Improved decoding of Folded Reed-Solomon and Multiplicity Codes

Swastik Kopparty[*]   Noga Ron-Zewi[†]   Shubhangi Saraf[‡]   Mary Wootters[§]

May 3, 2018

Abstract

In this work, we show new and improved error-correcting properties of folded Reed-Solomon codes and multiplicity codes. Both of these families of codes are based on polynomials over finite fields, and both have been the sources of recent advances in coding theory. Folded Reed-Solomon codes were the first explicit constructions of codes known to achieve list-decoding capacity; multivariate multiplicity codes were the first constructions of high-rate locally correctable codes; and univariate multiplicity codes are also known to achieve list-decoding capacity.

However, previous analyses of the error-correction properties of these codes did not yield optimal results. In particular, in the list-decoding setting, the guarantees on the list-sizes were polynomial in the block length, rather than constant; and for multivariate multiplicity codes, local list-decoding algorithms could not go beyond the Johnson bound.

In this paper, we show that Folded Reed-Solomon codes and multiplicity codes are in fact better than previously known in the context of list-decoding and local list-decoding. More precisely, we first show that Folded RS codes achieve list-decoding capacity with *constant* list sizes, independent of the block length; and that high-rate univariate multiplicity codes can also be list-recovered with constant list sizes. Using our result on univariate multiplicity codes, we show that multivariate multiplicity codes are high-rate, *locally* list-recoverable codes. Finally, we show how to combine the above results with standard tools to obtain capacity achieving locally list decodable codes with query complexity significantly lower than was known before.

1 Introduction

An error correcting code $C \subset \Sigma^n$ is a collection of *codewords c* of length n over an alphabet Σ. The goal in designing C is to enable the recovery of a codeword $c \in C$ given a corrupted version $\tilde{c}$ of c, while at the same time making C as large as possible. In the classical unique decoding problem, the goal is to efficiently recover c from any $\tilde{c} \in \Sigma^n$ so that c and $\tilde{c}$ differ in at most αn places; this requires that the *relative distance* δ of the code (that is, the fraction of places on which any two codewords differ) to be at least 2α.

Modern applications of error correcting codes, both in coding theory and theoretical computer science, have highlighted the importance of variants of the unique decoding problem, incuding *list decoding,* and *local decoding.* In list-decoding, the amount of error α is large enough that unique recovery of the codeword c is impossible (that is, $\alpha > \delta/2$), and instead the goal is to return a short list $\mathcal{L} \subset C$ with the guarantee that $c \in \mathcal{L}$. In local decoding, we still have $\alpha < \delta/2$, but the goal is to recover a single symbol c_i of a codeword c, after querying not too many positions of the corrupted codeword $\tilde{c}$. In a variant known as *local list-decoding,* we seek local information about a symbol even when $\alpha > \delta/2$. List-decoding, local decoding, and local list-decoding are important primitives in error correcting codes, with applications in coding theory, complexity theory, pseudorandomness and cryptography.

Algebraic codes have been at the heart of the study of list-decoding, local-decoding and local list-decoding. One classical example of this is Reed-Solomon (RS) codes, whose codewords are comprised of evaluations of low-degree polynomials.[1] In the late 1990's, Guruswami and Sudan [Sud97, GS99] gave an algorithm for efficiently list-decoding Reed-Solomon codes well beyond half the distance of the code, and this kicked off the field of algorithmic list-decoding. A second example is Reed-Muller (RM) codes, the multivariate analogue of Reed-Solomon codes. The structure of Reed-Muller codes is very amenable to local algorithms: a codeword of a Reed-Muller code corresponds to a multivariate low-degree polynomial, and considering the restriction of that polynomial to a line yields a univariate low-degree polynomial, *a.k.a.* a Reed-Solomon codeword. This local structure is the basis for Reed-Muller codes being locally testable [RS96] and locally decodable [Lip90, BFLS91]. Using this locality in concert with the Guruswami-Sudan algorithm leads to local list-decoding schemes [AS03, STV01] for these codes.

More recently, variants of Reed-Solomon and Reed-Muller codes have emerged to obtain improved list-decoding and local-decoding properties. Two notable examples, which are the focus of this work, are *Folded Reed-Solomon* (FRS) and *multiplicity codes.* Both of these constructions have led to recent advances in coding theory. We introduce these codes informally here, and give formal definitions in Section 2.

Folded Reed-Solomon codes, introduced by Guruswami and Rudra in [GR08], are a simple variant of Reed-Solomon codes. If the codeword of a Reed-Solomon code is $(c_0, c_2, \ldots, c_{n-1}) \in \Sigma^n$, then

---

[1]That is, a codeword of an RS code has the form $(f(x_0), f(x_1), \ldots, f(x_{n-1})) \in \mathbb{F}^n$ for some low-degree polynomial $f \in \mathbb{F}[X]$.

the folded version (with folding parameter s) is

$$\left(\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{s-1} \end{bmatrix}, \begin{bmatrix} c_s \\ c_{s+1} \\ \vdots \\ c_{2s-1} \end{bmatrix}, \ldots, \begin{bmatrix} c_{n-s} \\ c_{n-s+1} \\ \vdots \\ c_{n-1} \end{bmatrix} \right) \in (\Sigma^s)^{n/s}.$$

The main property of these codes that makes them interesting is that they admit much better list-decoding algorithms [GR08] than the original Guruswami-Sudan algorithm: more precisely, it allows for the error tolerance α to be much larger for a code of the same rate,[2] asymptotically obtaining the *optimal* trade-off.

Multiplicity codes, introduced in the univariate setting by Rosenbloom and Tsfasman in [RT97] and in the multivariate setting by Kopparty, Saraf and Yekhanin in [KSY14], are variants of polynomial codes that also include evaluations of derivatives. That is, while a symbol of a RS codeword is of the form $f(x) \in \mathbb{F}$ for some low-degree polynomial $f \in \mathbb{F}[X]$ and some $x \in \mathbb{F}$, a symbol in a univariate multiplicity code codeword is of the form $(f(x), f^{(1)}(x), f^{(2)}(x), \ldots, f^{(s-1)}(x)) \in \mathbb{F}^s$, where s is the *multiplicity parameter*. Similarly, while a symbol of an RM codeword is of the form $f(\mathbf{x})$ for $\mathbf{x} \in \mathbb{F}^m$ for some low-degree multivariate polynomial $f \in \mathbb{F}[X_1, \ldots, X_m]$, a symbol in a multivariate multiplicty code includes all partial derivatives of order less than s. Multivariate multiplicity codes were shown in [KSY14] to have strong locality properties, and were the first constructions known of high-rate locally decodable codes. Meanwhile, univariate multiplicity codes were shown in [Kop15, GW13] to be list-decodable in the same parameter regime as folded Reed-Solomon codes[3], also achieving asymptotically optimal trade-off between rate and error-tolerance.

In this work, we show that Folded Reed-Solomon codes, univariate multiplicity codes, and multivariate multiplicity codes are even more powerful than was previously known in the context of list-decoding and local list-decoding. Our motivations for this work are threefold:

1. First, FRS codes and multiplicity codes are basic and natural algebraic codes, central to many recent results in coding theory ([GR08, KSY14, Kop15, GW13, DL12, KMRS17, GKO+17], to name a few) and understanding their error-correcting properties is important in its own right.

2. Second, by composing our new results with known techniques, we obtain capacity-achieving locally list-decodable codes with significantly improved query complexity than previously known.

3. Third, while there have been improved constructions of list-decodable and locally list-decodable codes building on FRS and multiplicity codes (discussed more below), those constructions involve significant additional pseudorandom ingredients. Our results give simpler constructions of capacity achieving list-decodable and locally list-decodable codes with the best known parameters. In particular, we give the first constructions of linear[4] capacity-achieving list-decodable codes with constant alphabet size and constant output list size.

---

[2]The *rate* of a code $C \in \Sigma^n$ is defined as $R = \frac{1}{n} \log_{|\Sigma|}(|C|)$ and quantifies how much information can be sent using the code. We always have $R \in (0, 1)$, and we would like R to be as close to 1 as possible.

[3]They were previously shown to be list-decodable up to the Johnson bound by Nielsen [Nie01].

[4]Many codes in this paper have alphabet $\Sigma = \mathbb{F}_q^s$, where $\mathbb{F}_q$ is a finite field. For such "vector alphabet" codes, we use the term "linear" to mean "$\mathbb{F}_q$-linear".

We will state our results and contributions more precisely in Section 1.2 after setting up a bit more notation and surveying related work.

1.1 Related work

List-recoverable codes. While the discussion above focused on the more well-known problem of list-*decoding*, in this work we actually focus on a generalization of list-decoding known as list-*recovery*. Given a code $C \subseteq \Sigma^n$, an (α, ℓ, L)-list-recovery algorithm for C takes as input a sequence of lists $S_1, \ldots, S_n \subseteq \Sigma$, each of size at most ℓ, and returns a list $\mathcal{L}$ of all of the codewords $c \in C$ so that $c_i \in S_i$ for all but an α fraction of the coordinates i; the combinatorial requirement is that $|\mathcal{L}| \leq L$. List-decoding is the special case of list-recovery when $\ell = 1$.

Both list-recovery and list-decoding have been important in coding theory, especially in theoretical computer science, for the past several decades (see [Sud97, Vad12] for overviews). Initially, the generalization to list recovery was used as a building block towards constructions of list decodable and uniquely decodable codes [GI02, GI03, GI04, GI05, KMRS17, GKO+17, HRW17], although it has since found additional applications in algorithm design [INR10, NPR12, GNP+13].

The Guruswami-Sudan algorithm, mentioned above, is in fact a list-recovery algorithm as well as a list-decoding algorithm, and can efficiently list-recover Reed-Solomon codes up to radius $\alpha = 1 - \sqrt{\ell \cdot R}$, with polynomial list sizes L; this trade-off is known as the *Johnson bound.* It is a classical result that there are codes that go beyond the Johnson bound while keeping the output list size polynomial in n, or even constant: for large alphabet sizes, the "correct" limit (called the *list-decoding* or *list-recovering capacity*), is $\alpha = 1 - R$, provided q is sufficiently larger than ℓ, and this is achieved by uniformly random codes. There is a big difference between $1 - \sqrt{\ell \cdot R}$ and $1 - R$, especially when $\ell > 1$. In particular, the Guruswami-Sudan algorithm requires Reed-Solomon codes to have rate $R < 1/\ell$ to be (α, ℓ, L)-list-recoverable for nontrivial α, while a completely random code can achieve rates arbitrarily close to 1 (of course, without efficient decoding algorithms). For a decade it was open whether or not one could construct explicit codes which efficiently achieve list-decoding capacity.

In a breakthrough result, Guruswami and Rudra [GR08] (building on the work of Parvaresh and Vardy [PV05]) showed that the folding operation described above can make RS codes approach capacity with polynomial list-sizes. For some time, this was the only known route to capacity-achieving codes, until it was shown in [GW13, Kop15] that univariate multiplicity codes also do the job (again, with polynomial list sizes). Since then there has been a great deal of work aimed at reducing the list size and alphabet size of these constructions, both of which were polynomial in n (and both of which would ideally be independent of n). To reduce the alphabet size to constant, two high-level strategies are known to work: (1) swapping out the standard polynomial codes for Algebraic Geometry (AG) codes [GX12, GX13, GK16b], and (2) concatenation and *distance amplification* using expander graphs [AEL95, GI04, HW15, GKO+17, HRW17]. To reduce the list-size to constant, the known strategies involve passing to carefully constructing *subcodes* of Folded Reed-Solomon codes and univariate multiplicity codes, via pseudorandom objects such as *subspace evasive sets* or *subspace designs* [DL12, GW13, GX12, GX13, GK16b].

In this work, we show that in fact both folded Reed-Solomon codes and univariate multiplicity codes are *already* list-recoverable with constant list-sizes, with no additional modification needed!

3

Code	Alphabet size $\lvert\Sigma\rvert$	List size L	Explicit?	Linear?	Decoding time	Notes
Completely random code	$\ell^{O(1/\varepsilon)}$	$O(\ell/\varepsilon)$	No	No	-	
Random linear code [RW17]	$\ell^{O(1/\varepsilon)}$	$q^{O_\varepsilon(\log^2(\ell))}$	No	Yes	-	
Folded RS codes [GR08]	$\left(\frac{n}{\varepsilon^2}\right)^{O(\log(\ell)/\varepsilon^2)}$	$\left(\frac{n}{\varepsilon^2}\right)^{O(\log(\ell)/\varepsilon^2)}$	Yes	Yes	$n^{O(\log(\ell)/\varepsilon)}$	
Univariate Multiplicity [Kop15]	$\left(\frac{n}{\varepsilon^2}\right)^{O(\log(\ell)/\varepsilon^2)}$	$\left(\frac{n}{\varepsilon^2}\right)^{O(\log(\ell)/\varepsilon^2)}$	Yes	Yes	$n^{O(\log(\ell)/\varepsilon)}$	
Folded RS/Univariate Multiplicity [GW13]	$\left(\frac{n\ell}{\varepsilon^2}\right)^{O(\ell/\varepsilon^2)}$	$\left(\frac{n\ell}{\varepsilon}\right)^{O(\ell/\varepsilon)}$	Yes	Yes	$O(\ell\cdot n^2/\varepsilon)$	Output is a small subspace containing all nearby codewords.
Folded RS codes (This work, Theorem 3.1)	$\left(\frac{n\ell}{\varepsilon^2}\right)^{O(\ell/\varepsilon^2)}$	$\left(\frac{\ell}{\varepsilon}\right)^{O(\frac{1}{\varepsilon}\log(\ell/\varepsilon))}$	Yes	Yes	$\mathsf{poly}(n,L)$	
Univariate Multiplicity codes (This work, Theorem 4.1)	$\left(\frac{n\ell}{\varepsilon^2}\right)^{O(\ell/\varepsilon^2)}$	$\left(\frac{\ell}{\varepsilon}\right)^{O(\frac{1}{\varepsilon}\log(\ell/\varepsilon))}$	Yes	Yes	$\mathsf{poly}(n,L)$	For $d<q$ only.
Folded RS subcodes (via subspace evasive) [DL12]	$\left(\frac{n\ell}{\varepsilon^2}\right)^{O(\ell/\varepsilon^2)}$	$O\left(\frac{\ell}{\varepsilon}\right)^{O(\ell/\varepsilon)}$	Yes	No	$O_{\ell,\varepsilon}(n^2)$	
Folded AG (via subspace evasive) [GX12]	$\exp\left(\frac{\ell\log(\ell/\varepsilon)}{\varepsilon^2}\right)$	$O\left(\frac{\ell}{\varepsilon}\right)$	No	No	$\mathsf{poly}_{\ell,\varepsilon}(n)$	
Folded AG (via subspace designs) [GX13, GK16b]	$\exp\left(\frac{\ell\log(\ell/\varepsilon)}{\varepsilon^2}\right)$	$2^{2^{2^{O_{\varepsilon,\ell}(\log^*(n))}}}$	Yes	Yes	$O_{\ell,\varepsilon}(n^{O(1)})$	
Tensor products of AG subcodes, plus expander techniques [HRW17]	$\exp(\ell/\varepsilon^2)$	$2^{2^{2^{O_{\varepsilon,\ell}(\log^*(n))}}}$	Yes	Yes	$O_{\ell,\varepsilon}(n^{1.01})$	
Folded RS codes, plus expander techniques (This work, Corollary 6.6)	$(1+\ell)^{O(1/\varepsilon^5)}$	$O_{\varepsilon,\ell}(1)$	Yes	Yes	$\mathsf{poly}_{\ell,\varepsilon}(n)$	

Table 1: Constructions of (α,ℓ,L)-list-recoverable codes of rate $R^*-\varepsilon$, where $R^*=1-\alpha$ is list-recovering capacity (when $\lvert\Sigma\rvert\geq(1+\ell)^{\Omega(1/\varepsilon)}$). The top part of the table focuses on "simple" algebraic constructions; the bottom part has constructions which are involved. We assume that $R^*\in(0,1)$ is constant (independent of n,ε,ℓ).

The resulting codes still have large alphabet sizes, but this can be ameliorated by using the same expander-based techniques described above.

We summarize the state of affairs for list-recovery in Table 1, and discuss our contributions in more detail below in Section 1.2.

Locally list-recoverable codes. As mentioned above, local decoding has been an important theme in coding theory for the past several decades. Locality makes sense in the context of list-recovery as well. The definition of local list-recovery (given formally below as Definition 2.3 below) is a bit involved, but intuitively the idea is as follows. As with list-recovery, we have input lists $S=(S_1,\ldots,S_n)$, so that each S_i is of size at most ℓ. The goal is to obtain information about a single symbol c_i of a codeword i, given query access to S. More precisely, we will require that the decoder output a short list of randomized algorithms $A_1,\ldots,A_L$, each of which corresponds to a codeword c with $\lvert\{i:c_i\notin S_i\}\rvert\leq\alpha n$. The requirement is that if A_r corresponds to a codeword c,

4

Code	Alphabet size $|\Sigma|$	List size L	Locality t	Explicit?
Tensor products of AG Subcodes, plus expander techniques [HRW17]	$\exp(\ell/\varepsilon^2)$	$2^{2^{2^{O_{\varepsilon,\ell}(\log^*(n))}}}$	$O_{\ell,\varepsilon}(n^{0.01})$	Yes
Multivariate Multiplicity codes, plus expander techniques (This work, Theorem 6.2)	$(1+\ell)^{O(\ell/\varepsilon^{11})}$	$O_{\ell,\varepsilon}(1)$	$O_{\ell,\varepsilon}(n^{0.01})$	Yes
Multivariate Multiplicity codes, plus expander techniques (This work, Theorem 6.1)	$(1+\ell)^{O(1/\varepsilon^{10})}$	$\exp(\sqrt{\log(n)\log\log(n)})$	$\exp(\log^{3/4}(n)(\log\log(n))^{1/4})$	Yes

Table 2: Constructions of (t,α,ℓ,L)-locally-list-recoverable codes of rate $R^* - \varepsilon$, where $R^* = 1 - \alpha$ is list-recovering capacity (when $|\Sigma| \geq (1+\ell)^{\Omega(1/\varepsilon)}$). We assume that $R^* \in (0,1)$ is constant (independent of n, ε, ℓ).

then on input i, $A_r(i)$ outputs c_i with high probability, and using no more than t queries to S. If such a decoder exists, we say that the code is (t,α,ℓ,L)-locally-list-recoverable. Local list-decoding is the case special case where $\ell = 1$.

This definition may seem a bit convoluted, but it turns out to be the "right" definition for a number of settings. For example, local list-decoding algorithms are at the heart of algorithms in cryptography [GL89], learning theory [KM93], and hardness amplification and derandomization [STV01]. Locally list-*recoverable* codes have been desirable as a step towards obtaining efficient capacity-achieving local list-decoding algorithms. In particular, high-rate locally list-recoverable codes, combined with standard techniques, yield *capacity-achieving* locally list-decodable and locally list-recoverable codes.

However, until recently, we did not know of *any* high-rate locally list-recoverable codes. The first such construction was given recently in [HRW17]. The approach of [HRW17] is as follows: it takes a folded AG subcode from [GX13, GK16b] (which uses subspace designs to find the subcode); applies tensor products many times; and concatenates the result with a locally correctable code. Finally, to obtain capacity-achieving locally list-decodable/recoverable, codes, that work applies an expander-based technique of [AEL95] to pseudorandomly scramble up the symbols of the codewords to amplify the amount of error tolerated.

The reason that so much machinery was used in [HRW17] is that despite a great deal of effort, the "natural" algebraic approaches did not seem to work. Perhaps the most natural algebraic approach is via Reed-Muller codes, which have a natural local structure. As discussed above, a Reed-Muller codeword corresponds to a low-degree multivariate polynomial, and restricting such a polynomial to a line yields a low-degree univariate polynomial, which corresponds to a Reed-Solomon codeword. Using this connection, along with the Guruswami-Sudan algorithm for Reed-Solomon codes, Arora and Sudan [AS03] and Sudan, Trevisan and Vadhan [STV01] gave algorithms for locally list-decoding Reed-Muller codes up the the Johnson bound[5]. This algorithm also extends

---

[5]Technically these algorithms only came within a factor $\sqrt{2}$ of the Johnson bound. To go all the way to the

naturally to local list-recovery up to the Johnson bound [GKO⁺17], but this means that for large values of ℓ one cannot obtain high-rate codes.

One might hope to use a similar approach for multivariate multiplicity codes; after all, the univariate versions are list-recoverable to capacity. However, the fact that the list sizes were large was an obstacle to this approach, and again previous work on the local list-decodability of multivariate multiplicity codes also only worked up to the Johnson bound [Kop15].

In this work, we return to this approach, and—using our results on univariate multiplicity codes— show that in fact high-rate multivariate multiplicity codes are locally list-recoverable. Using our construction, combined with some expander-based techniques, we obtain capacity-achieving locally list-recoverable codes which improve on the state-of-the-art. The quantitative results are stated in Table 2, and we discuss them in more detail in the next section.

1.2 Our contributions

The main contribution of this work improved results on the (local)-list-recoverability of FRS codes and multiplicity codes. We discuss a few of the concrete outcomes below.

- **Constant list sizes for folded Reed-Solomon codes.** Theorem 3.1 says that a folded RS code of rate R and alphabet size $q^{O(\ell/\varepsilon^2)}$ is $(1 - R - \varepsilon, \ell, L)$-list-recoverable with $L = (\ell/\varepsilon)^{O\left(\frac{1}{\varepsilon} \log(\ell/\varepsilon)\right)}$. This improves over the previous best-known list size for this setting, which was $(n/\varepsilon)^{O\left(\frac{1}{\varepsilon^2} \log(\ell)\right)}$. In particular, when ε, ℓ are constant, the list size L improves from polynomial in n to a constant.

- **Constant list sizes for univariate multiplicity codes.** Theorem 4.1 recovers the same quantitative results as Theorem 3.1 for univariate multiplicity codes with degree d smaller than the characteristic of the underlying field.

  When the degree d is larger than the characteristic, which is what is relevant for the application to multivariate multiplicity codes, we obtain a weaker result. We no longer have capacity-achieving codes, but we obtain high-rate list-recoverable codes with constant list sizes. More precisely, Theorem 4.4 implies that rate R univariate multiplicity codes are efficiently (α, ℓ, L)-list-recoverable for $L = \ell^{O(\ell \log(\ell))}$ and $\alpha = O((1 - R)^2/\ell)$. In particular, Theorem 4.4 is nontrivial even for high-rate codes, while the Johnson bound only gives results for $R < 1/\ell$.

- **High-rate multivariate multiplicity codes are locally list-recoverable.** One reason to study the list-recoverability of univariate multiplicity codes is because list-recovery algorithms for univariate multiplicity codes can be used in local list-recovery algorithms for multivariate multiplicity codes. Theorems 5.1 and 5.2 show that high-rate multivariate multiplicity codes are locally list-recoverable. More precisely, in Theorem 5.1, we show that for constant ℓ, ε, a multivariate multiplicity code of length n with rate $1 - \varepsilon$: is efficiently (t, α, ℓ, L)-locally-list-recoverable for $\alpha = 1/\mathsf{polylog}(n)$, with list size L and query complexity t that are sub-polynomial in the block length n. In Theorem 5.2, we instantiate the same argument with

---

Johnson bound, one needs some additional ideas [BK09]; see [GK16a, Kop15] for further variations on this.

slightly different parameters to show a similar result where α and L are constant, but the query complexity t is of the form $t = O(n^{0.01})$.

- **Capacity-achieving locally list-recoverable codes over constant-sized alphabets.** Theorems 5.1 and 5.2 give high-rate locally-list-recoverable codes; however, these codes do not achieve capacity, and the alphabet sizes are quite large. Fortunately, following previous work, we can apply a series of by-now-standard expander-based techniques to obtain capacity-achieving locally list-recoverable codes over constant-sized alphabets. We do this in Theorems 6.1 and 6.2, respectively.

  The only previous construction of capacity-achieving locally list-recoverable codes (or even high-rate locally list-recoverable codes) is due to [HRW17], which achieved arbitrary polynomially small query complexity (and even subpolynomial query complexity $n^{O(1/\log\log n)}$) with slightly superconstant list size.

  Our codes in Theorem 6.1 achieve subpolynomial query complexity $\widetilde{\exp}(\log^{3/4} n)$ and subpolynomial list size. This brings the query complexity for capacity achieving local list-decodability close to the best known query complexity for locally decodable codes [KMRS17], which is $\widetilde{\exp}(\log^{1/2} n)$ (for the same codes).

  Our codes in Theorem 6.2 have arbitrary polynomially small query complexity, and constant list-size. This improves upon the codes of [HRW17].

  The quantitative details are shown in Table 2.

- **Deterministic constructions of capacity-achieving list-recoverable codes with constant alphabet size and list size.** Our result in Theorem 3.1 for Folded Reed-Solomon codes give capacity-achieving list-recoverable codes with constant list size, but with polynomial alphabet size. By running these through some standard techniques, we obtain in Corollary 6.6 efficient deterministic constructions of $\mathbb{F}_q$-linear, capacity-achieving, list-recoverable codes with constant alphabet size and list size, with a decoding algorithm that runs in time $n^{O(1)} \cdot \log(n)^{O_{\ell,\varepsilon}(1)}$.

  Codes with these properties do not seem to have been written down anywhere in the literature. Prior to our work, the same standard techniques could have also been applied to the codes of [DL12] (which are nonlinear subcodes of Folded Reed-Solomon codes) to construct nonlinear codes with the same behavior.

1.3 Overview of techniques

In this subsection, we give an overview of the proofs of our main results.

1.3.1 List recovery of folded Reed-Solomon and univariate multiplicity codes with constant output list size

Let $C \subseteq \Sigma^n$ be either a folded Reed-Solomon code or a univariate multiplicity code with constant distance $\delta > 0$. Suppose that s is the "folding parameter" or "multiplicity parameter," respectively, so that $\Sigma = \mathbb{F}_q^s$. We begin with a warm-up by describing an algorithm for *zero-error* list-recovery;

that is, when $\alpha = 0$. Here we are given "received lists" $S \in \binom{\Sigma}{\ell}^n$, and we want to find the list $\mathcal{L}$ of all codewords $c \in C$ such that $c_i \in S_i$ for each i. The groundbreaking work of [GR08] showed that for constant ℓ and large but constant s, $\mathcal{L}$ has size at most $q^{O_\ell(1)}$, and can be found in time $q^{O_\ell(1)}$. We now show that $\mathcal{L}$ is in fact of size at most $L = O_{\ell,\delta}(1)$, and can be found in time $\mathsf{poly}(q, L)$.

The starting point for our improved list-recovery algorithms for folded Reed-Solomon and univariate multiplicity codes is the *linear-algebraic approach* to list-recovering these codes that was taken in [GW13]. The main punchline of this approach is that the list $\mathcal{L}$ is contained in an $\mathbb{F}_q$ affine-subspace $v_0 + V$ of dimension at most $O_\varepsilon(\ell)$, and further that this subspace can be found in time $\mathsf{poly}(q)$ (this immediately leads to the previously known bound on $\mathcal{L}$). Armed with this insight, we now bring the received lists S back into play. How many elements c of the affine space $v_0 + V \subseteq C$ can have $c_i \in S_i$ for all $i \in [n]$? We show that there cannot be too many such c.

The proof is algorithmic: we will give a randomized algorithm PRUNE, which when given the low dimensional affine space $v_0 + V$, outputs a list of $K = O(1)$ elements of C, such that such that for any $c \in \mathcal{L}$, c is included in the output of A with high probability. This implies that $|\mathcal{L}| \leq O(K) = O(1)$.

The algorithm PRUNE works as follows. For some parameter $\tau = O(1)$, we pick coordinates $i_1, i_2, \ldots, i_\tau \in [n]$ uniformly at random. Then the algorithm iterates over all the ℓ^τ choices of $(y_1, \ldots, y_\tau) \in \prod_{j=1}^\tau S_{i_j}$. For each such $(y_1, \ldots, y_\tau)$, PRUNE checks if there is a unique element w of $v_0 + V$ such that $w_{i_j} = y_j$ for all $j \in [\tau]$. If so, we output that unique element w; otherwise (i.e., either there are either zero or greater than one such w's) we do nothing. Thus the algorithm PRUNE outputs at most $\ell^\tau = O(1)$ elements of C.

It remains to show that for any $c \in \mathcal{L}$, the algorithm outputs c with high probability. Fix such a c. By assumption, for every $i \in [n]$, $c_i \in S_i$. Thus there will be an iteration where the algorithm PRUNE takes $(y_1, \ldots, y_\tau) = (c_{i_1}, \ldots, c_{i_\tau})$. In this iteration, there will be at least one w (namely c) which has the desired property. Could there be more? If there was another $c' \in v_0 + V$ with this property, then the nonzero vector $c - c' \in V$ would have the property that $c - c'$ vanishes on all coordinates $i_1, \ldots, i_\tau$. It turns out that this can only happen with very low probability. Lemma 2 from [SY11] shows that that for any linear space V with dimension k and distance at least δ, for τ a large enough constant ($\tau = \Omega(k/\delta)$), it is very unlikely that there exists a nonzero element of V that vanishes at τ random coordinates $i_1, \ldots, i_\tau$. Thus with high probability, c is the unique w found in that iteration, and is thus included in the output of PRUNE. This completes the description and analysis of the algorithm PRUNE, and thus of our zero-error list-recovery algorithm.

One way to prove (a version of) Lemma 2 from [SY11] is as follows. First we note the following simple but important lemma:

Lemma 1.1. *Let $\Sigma = \mathbb{F}_q^s$. Let $W \subseteq (\Sigma)^n$ be an $\mathbb{F}_q$-subspace with $\dim(W) = t \geq 1$. Suppose W has minimum distance at least δ. Then:*

$$\mathbb{E}_{i \in [n]}[\dim(W \cap H_i)] \leq t - \delta,$$

where $H_i = \{v \in \Sigma^n \mid v_i = 0\}$.

Lemma 1.1 says that for any subspace $W \subseteq \Sigma^n$ of good distance, fixing a coordinate to 0 reduces the dimension a little in expectation. Iterating this, we see that fixing many coordinates is very likely to reduce the dimension down to zero, and this proves the result that we needed above.

With our warm-up complete, we turn to our main theorem on the list-recoverability of Folded Reed-Solomon codes (Theorem 3.1), which shows that the output list size is small even in the presence of an $\alpha = \delta - \varepsilon$ fraction of errors (for small $\varepsilon > 0$). Our approach generalizes the $\alpha = 0$ case described above. Let $\mathcal{L}$ be the list of $(\delta - \varepsilon)$-close codewords. Again, the linear-algebraic list decoder of [GW13] can produce a low dimensional affine subspace $v_0 + V$ such that $\mathcal{L} \subseteq v_0 + V$. Next, we show that the very same algorithm PRUNE described above (with a different setting of the parameter τ) does the desired list-recovery with at least some small constant probability p_0. This will imply that $|\mathcal{L}| \leq \frac{\ell^\tau}{p_0}$.

To see why this works, fix a codeword $c \in \mathcal{L}$. First observe that if we pick $i_1, \ldots, i_\tau$ uniformly at random, the probability that $c_{i_j} \in S_{i_j}$ for all $j = 1, \ldots, \tau$ is at least $p' = (1 - \delta + \varepsilon)^\tau$. This is small, but not too small; thus, there is some chance that at least one w (the correct one) is found by PRUNE.

Following the previous analysis, we now have to bound the probability that for random $i_1, \ldots, i_\tau \in [n]$, the space of codewords from V that vanish on all of $i_1, \ldots, i_{tau}$ has dimension at least one. This is the probability that strictly greater than one w is found by PRUNE. This time we will need a stronger (and much more specialized) version of Lemma 1.1, which shows that for subspaces W of the Folded Reed-Solomon code, fixing a random coordinate to 0 reduces the dimension by a lot: much more than the δ that we got from Lemma 1.1. Such a lemma was proved in [GK16b], although in a different language, and for a very different purpose. This lemma roughly shows that the expected dimension of $W \cap H_i$, for a random $i \in [n]$, is at most $(1 - \delta) \dim(W)$. Setting $\tau = O(\log(\dim(V))/\delta)$, with τ applications of this lemma, we get that the probability that the space of codewords from V that vanish on all of $i_1, \ldots, i_\tau$ has dimension at least one is at most $p'' = (1 - \delta)^\tau \dim(V)$. Note that this probability is tiny compared to p', and thus the probability that the algorithm PRUNE succeeds in finding c is at least $p' - p'' \approx p'$, as desired.

The description above was for folded RS codes, but same method works for univariate multiplicity codes whose degree d is smaller than the characteristic of the field $\mathbb{F}_q$. We state this in Theorem 4.1. The proof follows the same outline, using a different but analogous lemma from [GK16b].

For application to local list-recovery of multivariate multiplicity codes, however, we need to deal with univariate multiplicity codes where the degree d is larger than q. In Theorem 4.4, we show how to accomplish this when the fraction of errors α is very small. The algorithm and the outline of the analysis described above can again do the job for this setting, although the analysis is much more involved. The proof, which we give in Section 4, gives better quantitative bounds than the previous approach, and requires us to open up the relevant lemma from [GK16b]. At the end of the day, we are able to prove a reasonable version of this lemma for the case when $d > q$, and this allows the analysis to go through.

1.3.2   Local list-recovery of multivariate multiplicity codes

We now describe the high-level view of our local list-recovery algorithms. Our algorithm for local list-recovery of multivariate multiplicity codes follows the general paradigm for local list-decoding of Reed-Muller codes by Arora and Sudan [AS03] and Sudan, Trevisan and Vadhan [STV01]. In addition to generalizing various aspects of the paradigm, we need to introduce some further ideas

to account for the fact that we are in the high rate setting[6].

Local list-decoding of Reed-Muller codes is the following problem: we are given a function $r :$ $\mathbb{F}_q^m \to \mathbb{F}_q$ which is promised to be close to the evaluation table of some low degree polynomial $Q(X_1, \ldots, X_m)$. At the high level, the local list-decoding algorithm of [STV01] for Reed-Muller codes has two phases: generating advice, and decoding with advice. To generate the advice, we pick a uniformly random $\mathbf{a} \in \mathbb{F}_q^m$ and "guess" a value $z \in \mathbb{F}_q$ (this guessing can be done by going over all $z \in \mathbb{F}_q$). Our hope for this guess is that z equals $Q(\mathbf{a})$.

Once we have this advice, we see how to decode. We define an oracle machine $M^r[\mathbf{a}, z]$, which takes as advice $[\mathbf{a}, z]$, has query access to r, and given an input $\mathbf{x} \in \mathbb{F}_q^m$, tries to compute $Q(\mathbf{x})$. The algorithm first considers the line λ passing through $\mathbf{x}$ and the advice point $\mathbf{a}$, and list-decode the restriction of r to this line to obtain a list $\mathcal{L}_\lambda$ of univariate polynomials. These univariate polynomials are candidates for $Q|_\lambda$. Which of these univariate polynomials is $Q|_\lambda$? We use our guess z (which is suppose to be $Q(\mathbf{a})$): if there is a unique univariate polynomial in the list with value z at $\mathbf{a}$, then we deem that to be our candidate for $Q|_\lambda$, and output its value at the point $\mathbf{x}$ as our guess for $Q(\mathbf{x})$. This algorithm will be correct on the point $\mathbf{x}$ if (1) there are not too many errors on the line through $\mathbf{x}$ and $\mathbf{a}$, and (2) no other polynomnial in $\mathcal{L}_\lambda$ takes the same value at $\mathbf{a}$ as $Q|_\lambda$ does. The first event is low probability by standard sampling bounds, and the second is low probability using the random choice of $\mathbf{a}$ and the fact that $\mathcal{L}_\lambda$ is small. This algorithm does not succeed on all $\mathbf{x}$, but one can show that for random $\mathbf{a}$ and $z = Q(\mathbf{a})$, this algorithm does succeed on most $\mathbf{x}$. Then we can run a standard local correction algorithm for Reed-Muller codes to then convert it to an algorithm that succeeds on all $\mathbf{x}$ with high probability.

We are trying to locally list-recover a multivariate multiplicity code; the codewords are of the form $(Q^{(<s)}(\mathbf{y}))_{\mathbf{y} \in \mathbb{F}_q^m}$, where $Q^{(<s)}(\mathbf{y}) \in \mathbb{F}_q^{\binom{m+s-1}{m}} =: \Sigma_{m,s}$ is a tuple that consists of all partial derivatives of Q of order less than s, evaluated at $\mathbf{y}$. We are given query access to a function $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$, where $S(\mathbf{y}) \subset \Sigma_{m,s}$ is the received list for the coordinate indexed by $\mathbf{y}$. Suppose for the following discussion that $Q(\mathbf{X}) \in \mathbb{F}_q[X_1, \ldots, X_m]$ is a low-degree multivariate polynomial so that $|\{\mathbf{y} : Q^{(<s)}(\mathbf{y}) \notin S(\mathbf{y})\}| \leq \alpha q^m$. We want to describe an algorithm that, with high probability will output a randomized algorithm $A_j : \mathbb{F}_q^m \to \Sigma_{m,s}$ that will approximate $Q^{(<s)}$.

There are two main components to the algorithm again: generating the advice, and decoding with advice. The advice is again a uniformly random point $\mathbf{a} \in \mathbb{F}_q^m$, and a guess z which is supposed to equal $Q^{(<s^*)}(\mathbf{a})$, *a very high order evaluation of Q at $\mathbf{a}$*, for some $s^* \gg s$. We discuss how to generate z later, let us first see how to use this advice to decode.

To decode using the advice $[\mathbf{a}, z]$, we give an oracle machine $M^S[\mathbf{a}, z]$ which takes advice $[\mathbf{a}, z]$ and has query access to S. If $z = Q^{(<s^*)}(\mathbf{a})$, then $M^S[\mathbf{a}, z](\mathbf{x})$ will be equal to $Q^{(<s)}(\mathbf{x})$ with high probability over $\mathbf{x}$ and $\mathbf{a}$. This algorithm is discussed in Section 5.3. Briefly, the idea is to consider the line λ through $\mathbf{x}$ and $\mathbf{a}$ and again run the univariate list-recovery algorithm on the restrictions of S to this line to obtain a list $\mathcal{L}_\lambda$. We hope that $Q|_\lambda$ is in this list, and that $Q|_\lambda$ does not have the same *order s^** evaluation[7] on $\mathbf{a}$ as any other element of $\mathcal{L}_\lambda$ – this will allow us to identify it

---

[6]These ideas can also be used to improve the analysis of the [AS03] and [STV01] local list-decoders for Reed-Muller codes. In particular, they can remove the restriction that the degree d needs to be at most $1/2$ the size of the field $\mathbb{F}_q$ for the local list-decoder to work.

[7]This is why we take s^* large: it is much more unlikely that there will be a collision of higher order evaluations at the random point $\mathbf{a}$.

with the help of the advice $z = Q^{(<s^*)}(\mathbf{a})$. Once we identify $Q|_\lambda$, we output its value at $\mathbf{x}$ as our guess for $Q(\mathbf{x})$.

To generate the advice z, we give an algorithm RecoverCandidates, which takes as input a point $\mathbf{a} \in \mathbb{F}_q^m$, has query access to S, and returns a short list $Z \subset \Sigma_{m,s^*}$ of guesses for $Q^{(<s^*)}(\mathbf{a})$. Recall that we have s^* quite a bit larger than s. This algorithm is discussed in Section 5.2. Briefly, RecoverCandidates works by choosing random lines through $\mathbf{a}$ and running the (global) list-recovery algorithm for univariate multiplicity codes on the restriction of the lists S to these lines. Then it aggregates the results to obtain Z. This aggregation turns out to be a list-recovery problem for Reed-Muller codes evaluated on product sets. We describe this algorithm for list-recovery in Appendix D.

Summarizing, our local list-recovery algorithm works as follows. First, we run RecoverCandidates on a random point $\mathbf{a} \in \mathbb{F}_q^m$ to generate a short list $Z \subseteq \Sigma_{m,s^*}$ of possibilities for $Q^{(<s^*)}(\mathbf{a})$. Then, for each $z \in Z$, we will form the oracle machine $M^S[\mathbf{a}, z]$. We are not quite done even if the advice z is good, since $M^S[\mathbf{a}, z](\mathbf{x})$ may not be equal to $Q^{(<s)}(\mathbf{x})$; we know this probably happens for most $\mathbf{x}$'s, but not necessarily for the one that we care about. Fortunately, $M^S[\mathbf{a}, z]$ will agree with $Q^{(<s)}$ for many inputs $\mathbf{y}$, and so we can use the fact that multivariate multiplicity codes are locally correctable to finish the job [KSY14]. When we iterate over the advice $z \in Z$, this will give the list of randomized algorithms $A_1, \ldots, A_L$ that the local list-recovery algorithm returns.

1.3.3 Organization

We begin in Section 2 with notation and preliminary definitions. Once these are in place, we will prove Theorem 3.1 about Folded RS codes in Section 3. In Section 4, we extend our analysis of Folded RS codes to univariate multiplicity codes, and prove Theorems 4.1 and 4.4 for small and large degrees d respectively. In Section 5, we present our local list-recovery algorithm for multivariate multiplicity codes, and state Theorems 5.1 and 5.2 about high-rate local list-recovery of multivariate multiplicity codes. Finally in Section 6 we run our results through the expander-based machinery of [AEL95], to obtain Theorems 6.1 and 6.2 which give capacity-achieving locally list-recoverable codes over constant-sized alphabets.

2 Notation and Preliminaries

We begin by formally defining the coding-theoretic notions we will need, and by setting notation. We denote by $\mathbb{F}_q$ the finite field of q elements. For any pair of strings $x, y \in \Sigma^n$, the relative distance between x and y is the fraction of coordinates on which x and y differ, and is denoted by $\mathrm{dist}(x, y) := |\{i \in [n] : x_i \neq y_i\}| / n$. For a positive integer ℓ we denote by $\binom{\Sigma}{\ell}$ the set containing all subsets of Σ of size ℓ, and for any pair of strings $x \in \Sigma^n$ and $S \in \binom{\Sigma}{\ell}^n$ we denote by $\mathrm{dist}(x, S)$ the fraction of coordinates $i \in [n]$ for which $x_i \notin S_i$, that is, $\mathrm{dist}(x, S) := |\{i \in [n] : x_i \notin S_i\}| / n$. Throughout the paper, we use $\exp(n)$ to denote $2^{\Theta(n)}$. Whenever we use log, it is to the base 2. The notation $O_a(n)$ and $\mathsf{poly}_a(n)$ means that we treat a as a constant; that is, $\mathsf{poly}_a(n) = n^{O_a(1)}$.

2.1 Error-correcting codes

Let Σ be an alphabet and let n be a positive integer (the block length). A code is simply a subset $C \subseteq \Sigma^n$. The elements of a code C are called codewords. If $\mathbb{F}$ is a finite field and Σ is a vector space over $\mathbb{F}$, we say that a code $C \subseteq \Sigma^n$ is $\mathbb{F}$-linear if it is an $\mathbb{F}$-linear subspace of the $\mathbb{F}$-vector space Σ^n. In this work most of our codes will have alphabets $\Sigma = \mathbb{F}^s$, and we will use linear to mean $\mathbb{F}$-linear. The rate of a code is the ratio $\frac{\log |C|}{\log(|\Sigma|^n)}$, which for $\mathbb{F}$-linear codes equals $\frac{\dim_{\mathbb{F}}(C)}{n \cdot \dim_{\mathbb{F}}(\Sigma)}$. The relative distance $\mathrm{dist}(C)$ of C is the minimum $\delta > 0$ such that for every pair of distinct codewords $c_1, c_2 \in C$ it holds that $\mathrm{dist}(c_1, c_2) \geq \delta$.

Given a code $C \subseteq \Sigma^n$, we will occasionally abuse notation and think of $c \in C$ as a map $c : \mathcal{D} \to \Sigma$, where $\mathcal{D}$ is some domain of size n. With this notation, the map $c : \mathcal{D} \to \Sigma$ corresponds to the vector $(c(x))_{x \in \mathcal{D}} \in \Sigma^n$.

For a code $C \subseteq \Sigma^n$ of relative distance δ, a given parameter $\alpha < \delta/2$, and a string $w \in \Sigma^n$, the problem of decoding from α fraction of errors is the task of finding the unique $c \in C$ (if any) which satisfies $\mathrm{dist}(c, w) \leq \alpha$.

2.2 List-decodable and list-recoverable codes

List decoding is a paradigm that allows one to correct more than a $\delta/2$ fraction of errors by returning a small list of close-by codewords. More formally, for $\alpha \in [0, 1]$ and an integer L we say that a code $C \subseteq \Sigma^n$ is (α, L)-list-decodable if for any $w \in \Sigma^n$ there are at most L different codewords $c \in C$ which satisfy that $\mathrm{dist}(c, w) \leq \alpha$.

List recovery is a more general notion where one is given as input a small list of candidate symbols for each of the coordinates and is required to output a list of codewords that are consistent with many of the input lists. Formally we say that a code $C \subseteq \Sigma^n$ is (α, ℓ, L)-list-recoverable if for any $S \in \binom{\Sigma}{\ell}^n$ there are at most L different codewords $c \in C$ which satisfy that $\mathrm{dist}(c, S) \leq \alpha$. Note that list decoding corresponds to the special case of $\ell = 1$.

2.3 Locally correctable and locally list-recoverable codes

Locally correctable codes. Intuitively, a code is said to be locally correctable [BFLS91, STV01, KT00] if, given a codeword $c \in C$ that has been corrupted by some errors, it is possible to decode any coordinate of c by reading only a small part of the corrupted version of c. Formally, it is defined as follows.

Definition 2.1 (Locally correctable code (LCC)). *We say that a code $C \subseteq \Sigma^n$ is (t, α)-locally correctable if there exists a randomized algorithm A that satisfies the following requirements:*

- *__Input:__ A takes as input a coordinate $i \in [n]$ and also gets oracle access to a string $w \in \Sigma^n$ that is α-close to a codeword $c \in C$.*

- *__Query complexity:__ A makes at most t queries to the oracle w.*

- *__Output:__ A outputs c_i with probability at least $\frac{2}{3}$.*

Remark 2.2. By definition it holds that $\alpha < \text{dist}(C)/2$. The above success probability of $\frac{2}{3}$ can be amplified using sequential repetition, at the cost of increasing the query complexity. Specifically, amplifying the success probability to $1 - e^{-t}$ requires increasing the query complexity by a multiplicative factor of $O(t)$.

Locally list-recoverable codes. The following definition generalizes the notion of locally correctable codes to the setting of list decoding / recovery. In this setting the algorithm A is required to find all the nearby codewords in an implicit sense.

Definition 2.3 (Locally list-recoverable code)**.** *We say that a code $C \subseteq \Sigma^n$ is (t, α, ℓ, L)-locally list-recoverable if there exists a randomized algorithm A that satisfies the following requirements:*

- ***Input:** A gets oracle access to a string $S \in \binom{\Sigma}{\ell}^n$.*

- ***Query complexity:** A makes at most t queries to the oracle S.*

- ***Output:** A outputs L randomized algorithms $A_1, \ldots, A_L$, where each A_j takes as input a coordinate $i \in [n]$, makes at most t queries to the oracle S, and outputs a symbol in Σ.*

- ***Correctness:** For every codeword $c \in C$ for which $\text{dist}(c, S) \le \alpha$, with probability at least $\frac{2}{3}$ over the randomness of A the following event happens: there exists some $j \in [L]$ such that for all $i \in [n]$,*

$$\Pr[A_j(i) = c_i] \ge \frac{2}{3},$$

where the probability is over the internal randomness of A_j.

We say that A has running time T if A outputs the description of the algorithms $A_1, \ldots, A_L$ in time at most T and each A_j has running time at most T. We say that a code is (t, α, L)-locally list-decodable if it is $(t, \alpha, 1, L)$-locally list-recoverable.

2.4 Polynomials and derivatives

Let $\mathbb{F}_q[X]$ be the space of univariate polynomials over $\mathbb{F}_q$. We will often be working with linear and affine subspaces of $\mathbb{F}_q[X]$. We will denote linear subspaces of $\mathbb{F}_q[X]$ by the letters U, V, W, and affine subspaces of $\mathbb{F}_q[X]$ as $v_0 + V$, where $v_0 \in \mathbb{F}_q[X]$ and V is a linear subspace.

For polynomials $P_1, \ldots, P_s \in \mathbb{F}_q[X]$, we define their Wronskian, $W(P_1, \ldots, P_s)$, by

$$W(P_1, \ldots, P_s)(X) = \begin{pmatrix} P_1(X) & \cdots & P_s(X) \\ P_1^{(1)}(X) & \cdots & P_s^{(1)}(X) \\ \vdots & & \vdots \\ P_1^{(s-1)}(X) & \cdots & P_s^{(s-1)}(X) \end{pmatrix}.$$

For $i \in \mathbb{N}$, we define the i'th (Hasse) derivative $P^{(i)}(X)$ as the coefficient of Z^i in the expansion

$$P(X + Z) = \sum_i P^{(i)}(X) Z^i.$$

For multivariate polynomials $P \in \mathbb{F}_q[X_1, \ldots, X_m]$, we use the notation $\mathbf{X} = (X_1, \ldots, X_m)$ and $\mathbf{X}^{\mathbf{i}} = \prod_j X_j^{i_j}$ where $\mathbf{i} = (i_1, \ldots, i_m) \in \mathbb{Z}^m$. For $\mathbf{i} \in \mathbb{Z}^m$, we define the $\mathbf{i}$'th (Hasse) derivative $P^{(\mathbf{i})}(\mathbf{X})$ by

$$P(\mathbf{X} + \mathbf{Z}) = \sum_{\mathbf{i}} P^{(\mathbf{i})}(\mathbf{X})\mathbf{Z}^{\mathbf{i}}.$$

2.5 Some families of polynomial codes

In this section, we formally define the families of codes we will study: folded Reed-Solomon codes [GR08], univariate multiplicity codes [RT97, KSY14, GW13], and multivariate multiplicity codes [KSY14].

Folded Reed-Solomon codes. Let q be a prime power, and let s, d, n be nonnegative integers such that $n \leq (q-1)/s$. Let $\gamma \in \mathbb{F}_q$ be a primitive element of $\mathbb{F}_q$, and let $a_1, a_2, \ldots, a_n$ be distinct elements in $\{\gamma^{si} \mid 0 \leq i \leq (q-1)/s - 1\}$. Let $\mathcal{D} = \{a_1, \ldots, a_n\}$.

For a polynomial $P(X) \in \mathbb{F}_q[X]$ and $a \in \mathbb{F}_q$, let $P^{[s]}(a) \in \mathbb{F}_q^s$ denote the vector:

$$P^{[s]}(a) = \begin{bmatrix} P(a) \\ P(\gamma a) \\ \vdots \\ P(\gamma^{s-1}a) \end{bmatrix}.$$

The **folded Reed-Solomon code** $\mathrm{FRS}_{q,s}(n, d)$ is a code over alphabet $\mathbb{F}_q^s$. To every polynomial $P(X) \in \mathbb{F}_q[X]$ of degree at most d, there corresponds a codeword c:

$$c : \mathcal{D} \to \mathbb{F}_q^s,$$

where for each $a \in \mathcal{D}$:

$$c(a) = P^{[s]}(a).$$

Explicitly,

$$P(x) \mapsto \left(P^{[s]}(a_1), P^{[s]}(a_2), \ldots, P^{[s]}(a_n) \right)$$

$$= \left(\begin{bmatrix} P(a_1) \\ P(\gamma a_1) \\ \vdots \\ P(\gamma^{s-1}a_1) \end{bmatrix}, \begin{bmatrix} P(a_2) \\ P(\gamma a_2) \\ \vdots \\ P(\gamma^{s-1}a_2) \end{bmatrix}, \ldots, \begin{bmatrix} P(a_n) \\ P(\gamma a_n) \\ \vdots \\ P(\gamma^{s-1}a_n) \end{bmatrix} \right).$$

We denote the codeword of $\mathrm{FRS}_{q,s}(n, d)$ corresponding to the polynomial $P(X)$ by $\mathsf{FRSEnc}_s(P)$ (when the parameters q, n are clear from the context).

Note that **Reed-Solomon codes** correspond to the special case of $s = 1$. The following claim summarizes the basic properties of folded Reed-Solomon codes.

Claim 2.4 ([GR08]). *The folded Reed-Solomon code* $\mathrm{FRS}_{q,s}(n, d)$ *is an* $\mathbb{F}_q$*-linear code over alphabet* $\mathbb{F}_q^s$ *of block length* n, *rate* $(d+1)/(sn)$, *and relative distance at least* $1 - d/(sn)$.

Univariate multiplicity codes. Let q be a prime power, and let s, d, n be nonnegative integers such that $n \leq q$. Let $a_1, a_2, \ldots, a_n$ be distinct elements in $\mathbb{F}_q$. Let $\mathcal{D} = \{a_1, \ldots, a_n\}$.

For a polynomial $P(X) \in \mathbb{F}_q[X]$, let $P^{(<s)}(x) \in \mathbb{F}_q^s$ denote the vector:

$$P^{(<s)}(a) = \begin{bmatrix} P(a) \\ P^{(1)}(a) \\ \vdots \\ P^{(s-1)}(a) \end{bmatrix}.$$

The univariate multiplicity code $\mathsf{MULT}_{q,s}^{(1)}(n, d)$ is a code over alphabet $\mathbb{F}_q^s$. To every polynomial $P(X) \in \mathbb{F}_q[X]$ of degree at most d, there corresponds a codeword c:

$$c : \mathcal{D} \to \mathbb{F}_q^s,$$

where for each $a \in \mathcal{D}$:

$$c(a) = P^{(<s)}(a).$$

Explicitly,

$$P(x) \mapsto \left(P^{(<s)}(a_1), P^{(<s)}(a_2), \ldots, P^{(<s)}(a_n) \right)$$
$$= \left(\begin{bmatrix} P(a_1) \\ P^{(1)}(a_1) \\ \vdots \\ P^{(s-1)}(a_1) \end{bmatrix}, \begin{bmatrix} P(a_2) \\ P^{(1)}(a_2) \\ \vdots \\ P^{(s-1)}(a_2) \end{bmatrix}, \ldots, \begin{bmatrix} P(a_n) \\ P^{(1)}(a_n) \\ \vdots \\ P^{(s-1)}(a_n) \end{bmatrix} \right).$$

We denote the codeword of $\mathsf{MULT}_{q,s}^{(1)}(n, d)$ corresponding to the polynomial $P(X)$ by $\mathsf{MultEnc}_s(P)$ (when the parameters q, n are clear from the context).

Once again, Reed-Solomon codes correspond to the special case of $s = 1$.

Claim 2.5 ([KSY14], Lemma 9). *The univariate multiplicity code* $\mathsf{MULT}_{q,s}^{(1)}(n, d)$ *is an* $\mathbb{F}_q$-*linear code over alphabet* $\mathbb{F}_q^s$ *of block length* n, *rate* $(d+1)/(sn)$, *and relative distance at least* $1 - d/(sn)$.

Of particular importance is the setting where $q = n$ and $\mathcal{D}$ equals the whole field $\mathbb{F}_q$. We refer to this code as the *whole-field univariate multiplcity code*, and denote it by $\mathsf{MULT}_{q,s}^{(1)}(d)$. This will be relevant to multivariate multiplicity codes, which we define next.

Multivariate multiplicity codes. Multivariate multiplicity codes are a generalization of whole-field univariate multiplicity codes to the multivariate setting.

Let q be a prime power, and let s, d, m be nonnegative integers. Let $U_{m,s}$ denote the set $\{\mathbf{i} \in \mathbb{N}^m \mid \mathsf{wt}(\mathbf{i}) < s\}$. Note that $|U_{m,s}| = \binom{s+m-1}{m}$. Let $\Sigma_{m,s} = \mathbb{F}_q^{U_{m,s}}$.

For a polynomial $P(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$, and a point $\mathbf{a} \in \mathbb{F}_q^m$, define $P^{(<s)}(\mathbf{a}) \in \Sigma_{m,s}$ by:

$$P^{(<s)}(\mathbf{a}) = (P^{(\mathbf{i})}(\mathbf{a}))_{\mathbf{i} \in U_{m,s}}.$$

The multiplicity code $\mathsf{MULT}^{(m)}_{q,s}(d)$ is a code over alphabet $\Sigma_{m,s}$. To every polynomial $P(X_1, \ldots, x_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ of (total) degree at most d, there corresponds a codeword as

$$c : \mathbb{F}_q^m \to \Sigma_{m,s},$$

where for each $\mathbf{a} \in \mathbb{F}_q^m$,

$$c(\mathbf{a}) = P^{(<s)}(\mathbf{a}).$$

Note that Reed-Muller codes correspond to the special case of $s = 1$.

Claim 2.6 ([KSY14], Lemma 9)**.** *The multivariate multiplicity code $\mathsf{MULT}^{(m)}_{q,s}(d)$ is an $\mathbb{F}_q$-linear code over alphabet $\mathbb{F}_q^{\binom{m+s-1}{m}}$ of block length q^m, rate at least $(1 - m^2/s)(d/(sq))^m$, and relative distance at least $1 - d/(sq)$.*

3 List recovering folded Reed-Solomon codes with constant output list size

Our first main result shows that folded Reed-Solomon codes are list-recoverable (and in particular, list-decodable) up to capacity with constant output list size, independent of n.

Theorem 3.1 (List recovering FRS with constant output list size)**.** *Let q be a prime power, and let s, d, n be nonnegative integers such that $n \le (q-1)/s$. Let $\varepsilon > 0$ and $\ell \in \mathbb{N}$ be such that $16\ell/\varepsilon^2 \le s$. Then the folded Reed-Solomon code $\mathrm{FRS}_{q,s}(n, d)$ is (α, ℓ, L)-list-recoverable for $\alpha = 1 - d/(sn) - \varepsilon$ and $L = \left(\frac{\ell}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon} \log \frac{\ell}{\varepsilon}\right)}$.*

Moreover, there is a randomized algorithm that list recovers $\mathrm{FRS}_{q,s}(n, d)$ with the above parameters in time $\mathsf{poly}(\log q, s, d, n, (\ell/\varepsilon)^{\log(\ell/\varepsilon)/\varepsilon})$.

In particular, the $\ell = 1$ case yields the following statement about list-decoding.

Corollary 3.2 (List decoding FRS with constant output list size)**.** *Let q be a prime power, and let s, d, n be nonnegative integers such that $n \le (q - 1)/s$. Let $\varepsilon > 0$ be such that $16/\varepsilon^2 \le s$. Then the folded Reed-Solomon code $\mathrm{FRS}_{q,s}(n, d)$ is (α, L)-list decodable for $\alpha = 1 - d/(sn) - \varepsilon$ and $L = \left(\frac{1}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}\right)}$.*

Moreover, there is a randomized algorithm that list decodes $\mathrm{FRS}_{q,s}(n, d)$ with the above parameters in time $\mathsf{poly}(\log q, s, d, n, (1/\varepsilon)^{\log(1/\varepsilon)/\varepsilon})$.

The proof of Theorem 3.1 consists of two main steps. The first step, from [GW13], shows that the output list is contained in a low dimensional subspace. The second step, which relies on results from [GK16b], shows that the output list cannot contain too many codewords from a low dimensional subspace, and therefore is small. The two steps are presented in Sections 3.1 and 3.2, respectively, followed by the proof of Theorem 3.1 in Section 3.3.

3.1 Output list is contained in a low dimensional subspace

The following theorem from [GW13] shows that the output list is contained in a low dimensional subspace, which can also be found efficiently.

Theorem 3.3 ([GW13], Theorem 7). *Let q be a prime power, and let s, d, n, ℓ, r be nonnegative integers such that $n \leq (q-1)/s$ and $r \leq s$. Let $S : \mathcal{D} \to \binom{\mathbb{F}_q^s}{\ell}$ be an instance of the list-recovery problem for $\mathrm{FRS}_{q,s}(n, d)$. Suppose the decoding radius α satisfies:*

$$\alpha \leq 1 - \frac{\ell}{r+1} - \frac{r}{r+1} \cdot \frac{s}{s-r+1} \cdot \frac{d}{sn}. \tag{1}$$

Let

$$\mathcal{L} = \{P(X) \in \mathbb{F}_q[X] \mid \deg(P) \leq d \ and \ \mathrm{dist}(\mathsf{FRSEnc}_s(P), S) \leq \alpha\}.$$

There is a (deterministic) algorithm that given S, runs in time $\mathsf{poly}(\log q, s, d, n, \ell)$, and computes an affine subspace $v_0 + V \subseteq \mathbb{F}_q[X]$ such that:

1. $\mathcal{L} \subseteq V$,

2. $\dim(V) \leq r - 1$.

Remark 3.4. Theorem 7 of [GW13] only deals with the case where $a_i = \gamma^{s(i-1)}$ for all $i = 1, \ldots, n$, and $\ell = 1$. However, it can be verified that the proof goes through for any choice of distinct $a_1, a_2, \ldots, a_n$ in $\{\gamma^{si} \mid 0 \leq i \leq (q-1)/s - 1\}$, and $\ell \in \mathbb{N}$ (for the latter see discussion at end of Section 2.4 of [GW13]).

3.2 Output list cannot contain many codewords from a low dimensional subspace

To show that the output list $\mathcal{L}$ cannot contain too many elements from a low dimensional subspace (and to find $\mathcal{L}$ in the process), we first give a preliminary randomized algorithm $\mathsf{PruneListFRS}$ that outputs a constant size list $\mathcal{L}'$ such that any codeword of $\mathcal{L}$ appears in $\mathcal{L}'$ with a constant probability p_0. This implies that $|\mathcal{L}| \leq |\mathcal{L}'|/p_0$, proving the first part of Theorem 3.1. Now that we know that $|\mathcal{L}|$ is small, our final algorithm simply runs $\mathsf{PruneListFRS}$ $O(\frac{1}{p_0} \log |\mathcal{L}|)$ times and returns the union of the output lists. By a union bound, all elements of $\mathcal{L}$ will appear in the union of the output lists with high probability. This will complete the proof of the second part of Theorem 3.1.

We start by describing the algorithm $\mathsf{PruneListFRS}$ and analyzing it. The algorithm is given as input $S : \mathcal{D} \to \binom{\mathbb{F}_q^s}{\ell}$, an $\mathbb{F}_q$-affine subspace $v_0 + V \subseteq \mathbb{F}_q[X]$ consisting of polynomials of degree at most d and of dimension at most r, and a parameter $\tau \in \mathbb{N}$.

Algorithm $\mathsf{PruneListFRS}(S, v_0 + V, \tau)$

1. Initialize $\mathcal{L}' = \emptyset$.

2. Pick $b_1, b_2, \ldots, b_\tau \in \mathcal{D}$ independently and uniformly at random.

3. For each choice of $y_1 \in S(b_1), y_2 \in S(b_2), \ldots, y_\tau \in S(b_\tau)$:

   - If there is exactly one codeword $P(X) \in v_0 + V$ such that $P^{[s]}(b_j) = y_j$ for all $j \in [\tau]$, then:
   $$\mathcal{L}' \leftarrow \mathcal{L}' \cup \{P(X)\}.$$

4. Output $\mathcal{L}'$.

Lemma 3.5. *The algorithm* PruneListFRS *runs in time* $\mathsf{poly}(\log q, s, n, \ell^\tau)$*, and outputs a list* $\mathcal{L}'$ *containing at most* ℓ^τ *polynomials, such that any polynomial* $P(X) \in v_0 + V$ *with* $\mathrm{dist}(\mathsf{FRSEnc}_s(P), S) \leq \alpha$ *appears in* $\mathcal{L}'$ *with probability at least*

$$(1 - \alpha)^\tau - r \left(\frac{d}{(s-r)n} \right)^\tau.$$

Proof. We clearly have that $|\mathcal{L}'| \leq \ell^\tau$, and that the algorithm has the claimed running time. Fix a polynomial $\hat{P} \in v_0 + V$ such that $\mathrm{dist}(\mathsf{FRSEnc}_s(\hat{P}), S) \leq \alpha$, we shall show below that $\hat{P}$ belongs to $\mathcal{L}'$ with probability at least

$$(1 - \alpha)^\tau - r \left(\frac{d}{(s-r)n} \right)^\tau.$$

Let E_1 denote the event that $\hat{P}^{[s]}(b_j) \in S(b_j)$ for all $j \in [\tau]$. Let E_2 denote the event that for all nonzero polynomials $Q \in V$ there exists some $j \in [\tau]$ such that $Q^{[s]}(b_j) \neq 0$. By the assumption that $\mathrm{dist}(\mathsf{FRSEnc}_s(\hat{P}), S) \leq \alpha$, we readily have that

$$\Pr[E_1] \geq (1 - \alpha)^\tau.$$

Claim 3.6 below also shows that

$$\Pr[E_2] \geq 1 - r \left(\frac{d}{(s-r)n} \right)^\tau.$$

So both E_1 and E_2 occur with probability at least

$$(1 - \alpha)^\tau - r \left(\frac{d}{(s-r)n} \right)^\tau.$$

If E_2 occurs, then for every choice of $y_1 \in S(b_1), y_2 \in S(b_2), \ldots, y_\tau \in S(b_2)$, there can be at most one polynomial $P(X) \in v_0 + V$ such that $P^{[s]}(b_j) = y_j$ for all $j \in [\tau]$ (otherwise, the difference $Q = P_1 - P_2 \in V$ of two such distinct polynomials would have $Q^{[s]}(b_j) = 0$ for all $j \in [\tau]$, contradicting E_2). If E_1 also occurs, then in the iteration of Step 3 where $y_j = \hat{P}^{[s]}(b_j)$ for each $j \in [\tau]$, the algorithm will take $P = \hat{P}$, and thus $\hat{P}$ will be included in $\mathcal{L}'$. This completes the proof of the lemma. $\qquad\square$

It remains to prove the following claim.

Claim 3.6.
$$\Pr[E_2] \geq 1 - r \left(\frac{d}{(s-r)n} \right)^\tau .$$

The proof of the claim relies on the following theorem from [GK16b].

Theorem 3.7 ([GK16b], Theorem 14). *Let $W \subseteq \mathbb{F}_q[X]$ be a linear subspace of polynomials of degree at most d. Suppose $\dim(W) = t \leq s$. Let $a_1, a_2, \ldots, a_n$ be distinct elements in $\{\gamma^{si} \mid 0 \leq i \leq (q-1)/s - 1\}$, and for $i \in [n]$ let*

$$H_i = \{P(X) \in \mathbb{F}_q[X] \mid P(\gamma^j a_i) = 0 \ \forall j \in \{0, 1, \ldots, s-1\}\}.$$

Then

$$\sum_{i=1}^n \dim(W \cap H_i) \leq \frac{d}{s-t+1} \cdot t.$$

Proof of Claim 3.6. For $0 \leq j \leq \tau$, let

$$V_j := V \cap H_{i_1} \cap H_{i_2} \cap \ldots \cap H_{i_j},$$

and $t_j := \dim(V_j)$. Observe that $r = t_0 \geq t_1 \geq \ldots \geq t_\tau$, and that event E_2 holds if and only if $t_\tau = 0$.

By Theorem 3.7,

$$\mathbb{E}[t_{j+1} \mid t_j = t] = \mathbb{E}_{i \in [n]}[\dim(V_j \cap H_i) \mid \dim(V_j) = t] \leq \frac{t}{s-t+1} \cdot \frac{d}{n} \leq t \cdot \frac{d}{(s-r)n}.$$

Thus

$$\mathbb{E}[t_{j+1}] \leq \mathbb{E}[t_j] \cdot \frac{d}{(s-r)n},$$

and

$$\mathbb{E}[t_\tau] \leq \mathbb{E}[t_0] \cdot \left(\frac{d}{(s-r)n} \right)^\tau = r \left(\frac{d}{(s-r)n} \right)^\tau .$$

Finally, by Markov's inequality this implies in turn that

$$\Pr[E_2] = \Pr[t_\tau = 0] = 1 - \Pr[t_\tau \geq 1] \geq 1 - r \left(\frac{d}{(s-r)n} \right)^\tau .$$

$\square$

3.3   Proof of Theorem 3.1

We now prove Theorem 3.1 based on Theorem 3.3 and Lemma 3.5.

Proof of Theorem 3.1. Let $S : \mathcal{D} \to \binom{\mathbb{F}_q^s}{\ell}$ be the received sequence of input lists. We would like to find a list $\mathcal{L}$ of size $\left(\frac{\ell}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon}\log(\ell/\varepsilon)\right)}$ that contains all polynomials $P(X)$ of degree at most d with $\text{dist}(\mathsf{FRSEnc}_s(P), S) \leq \alpha$.

Let $v_0 + V$ be the subspace found by the algorithm of Theorem 3.3 for S and $r = \frac{4\ell}{\varepsilon}$ (so $r \leq \frac{1}{4}\varepsilon s$ by assumption that $s \geq 16\ell/\varepsilon^2$). Note that for this choice of r the RHS of (1) is at least

$$1 - \frac{\varepsilon}{4} - \frac{1}{1 - \varepsilon/4} \cdot \frac{d}{sn} \geq 1 - \frac{d}{sn} - \varepsilon = \alpha,$$

and so all polynomial $P(X)$ of degree at most d with $\text{dist}(\mathsf{FRSEnc}_s(P), S) \leq \alpha$ are included in V.

Next we invoke Lemma 3.5 with S, $v_0 + V$ and $\tau = O(\frac{1}{\varepsilon}\log(\ell/\varepsilon))$. Then the algorithm $\mathsf{PruneListFRS}$ returns a list $\mathcal{L}'$ of size at most ℓ^τ such that each polynomial $P(X)$ of degree at most d with $\text{dist}(\mathsf{FRSEnc}_s(P), S) \leq \alpha$ is included in $\mathcal{L}'$ with probability p_0, which is at least

$$
\begin{aligned}
(1 - \alpha)^\tau - r\left(\frac{d}{(s-r)n}\right)^\tau &\geq (1-\alpha)^\tau - r\left(\frac{1}{1-\varepsilon/4} \cdot \frac{d}{sn}\right)^\tau \\
&\geq (1-\alpha)^\tau - \frac{1}{2}\left(\frac{1+\varepsilon/4}{1-\varepsilon/4} \cdot (1-\alpha-\varepsilon)\right)^\tau \\
&\geq \frac{1}{2}(1-\alpha)^\tau,
\end{aligned}
$$

where the first inequality follows since $r \leq \frac{1}{4}\varepsilon s$, and the second inequality holds since $r = \frac{4\ell}{\varepsilon} \leq \frac{1}{2} \cdot (1 + \frac{\varepsilon}{4})^\tau$ and $\alpha = 1 - \frac{d}{sn} - \varepsilon$.

The above implies in turn that

$$|\mathcal{L}| \leq \frac{|\mathcal{L}'|}{p_0} \leq 2\left(\frac{\ell}{1-\alpha}\right)^\tau \leq \left(\frac{\ell}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon}\log(\ell/\varepsilon)\right)}.$$

Moreover, by running the algorithm $\mathsf{PruneListFRS}$ $O(\frac{1}{p_0}\log|\mathcal{L}|)$ times and returning the union of all output lists, by a union bound, all elements of $\mathcal{L}$ will appear in the union of the output lists with high probability (say, at least 0.99). This gives a randomized list recovery algorithm with output list size $\left(\frac{\ell}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon}\log(\ell/\varepsilon)\right)}$ and running time $\mathsf{poly}(\log q, s, d, n, (\ell/\varepsilon)^{\log(\ell/\varepsilon)/\varepsilon})$. $\qquad\square$

4   List recovering high-rate univariate multiplicity codes with constant output list size

In this section, we show that univariate multiplicity codes of high rate can be list recovered from constant-sized input lists with constant-sized output lists.

4.1 Small d

If the degree d of the univariate multiplicity code is less than $\mathsf{char}(\mathbb{F}_q)$, the characteristic of the field $\mathbb{F}_q$, then the proof from the previous section works verbatim. The only changes needed are as follows.

- First, use Theorem 17 from [GW13] instead of Theorem 7 from that paper, to show that the list is contained in a low-dimensional subspace.

- Second, use Theorem 17 from [GK16b] instead of Theorem 14 from that paper, to show that for a low dimensional subspace W, at a typical $a \in \mathcal{D}$ we have $\dim(\{P(X) \in W \mid P^{(<s)}(a) = 0\})$ is small.

The condition $d < \mathsf{char}(\mathbb{F}_q)$ is used in both steps. These changes lead to the following theorem.

Theorem 4.1 (List recovering univariate multiplicity codes over prime fields with $d < \mathsf{char}(\mathbb{F}_q)$). *Let q be a prime power, and let s, d, n be nonnegative integers such that $d < \mathsf{char}(\mathbb{F}_q)$ and $n < \mathsf{char}(\mathbb{F}_q)/s$.*

Let $\varepsilon > 0$ and $\ell \in \mathbb{N}$ be such that $16\ell/\varepsilon^2 \le s$. Then the univariate muliplicity code $\mathsf{MULT}_{q,s}^{(1)}(n, d)$ is (α, ℓ, L)-list recoverable for $\alpha = 1 - d/(sn) - \varepsilon$ and $L = \left(\frac{\ell}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon} \log \frac{\ell}{\varepsilon}\right)}$.

Moreover, there is a randomized algorithm that list recovers $\mathsf{MULT}_{q,s}^{(1)}(n, d)$ with the above parameters in time $\mathsf{poly}(\log q, s, d, n, (\ell/\varepsilon)^{\log(\ell/\varepsilon)/\varepsilon})$.

Remark 4.2 (Fields of characteristic zero.)**.** The exact same techniques also work over fields of characteristic 0. We state the the analogous combinatorial statement over $\mathbb{C}$, which may be of independent interest.

Theorem 4.3. *Let $\varepsilon > 0$. Let $d, n > 0$ be an integer. Let $\mathcal{D} \subseteq \mathbb{C}$ with $|\mathcal{D}| = n$. Let $s > \frac{16\ell}{\varepsilon^2}$ be an integer. Let $\alpha \le 1 - \frac{d}{sn} - \varepsilon$.*

Let $S : \mathcal{D} \to \binom{\mathbb{C}^s}{\ell}$ be arbitrary. Then:

$$\left| \left\{ P(X) \in \mathbb{C}[X] \mid \deg(P) \le d \text{ s.t. } |\{a \in \mathcal{D} \mid P^{(<s)}(a) \in S(a)\}| \ge (1 - \alpha)n \right\} \right| \le \left(\frac{\ell}{\varepsilon}\right)^{O\left(\frac{1}{\varepsilon} \log \frac{\ell}{\varepsilon}\right)}.$$

In particular, when $\delta > 0$ is a constant and we take $d = (1 - \delta)n$, $\varepsilon = \delta$, $s = O(\ell/\delta^2)$ and $\alpha = 0$, then the bound on L is independent of n.

4.2 Large d

Theorem 4.1 works when $d < q$, but for application to multivariate multiplicity codes, however, it is important that we can list-recover univariate multiplicity codes when the evaluation set $\mathcal{D}$ equals all of $\mathbb{F}_q$ (i.e., whole-field univariate multiplicity codes).

For the rest of this section we assume that $\mathcal{D} = \mathbb{F}_q$, and hence that $n = q$. In this setting, for the rate to be high, we would also like the degree d to be close to sq, and thus $\gg q \geq \mathsf{char}(\mathbb{F}_q)$. This precludes use of the Theorem 4.1.

Instead, we will dig deeper into the proof to see what can be salvaged when d is larger than q. For the first step, it turns out that if d is only moderately larger than $\mathsf{char}(\mathbb{F}_q)$, then the list can be captured inside a moderately small dimensional subspace. Thus if we make q prime, so that $q = \mathsf{char}(\mathbb{F}_q)$, then this step can still work for $d < sq$ and s not too large. The second step uses the $d < \mathsf{char}(\mathbb{F}_q)$ condition more essentially. By a reworking of several algebraic tools used in the proof of Theorem 17 from [GK16b], we prove a generalization of it to handle polynomials of degree $> q$. This generalization will only apply to subspaces W of a special kind ("(X^q, d)-closed" subspaces). The list-recoverability we show here is quantitatively weaker (in terms of the fraction of errors that can be tolerated) than the results we proved in Theorem 3.1 and Theorem 4.1. Nevertheless, this form of the result still suffices to needed to show high-rate *local* list-recoverability of multivariate multiplicity codes in the following section.

Theorem 4.4. *Let $\delta > 0$. Let q be a prime, and let s, d, ℓ be nonnegative integers such that $s < q$ and $d < (1 - \delta)sq$, and $1 \leq \ell < \frac{\delta^2 s}{16}$. Suppose $\alpha < \frac{1}{2s}$.*

Then the **whole-field** *univariate muliplicity code* $\mathsf{MULT}_{q,s}^{(1)}(d)$ *is (α, ℓ, L)-list recoverable for $L = \ell^{O(s \log s)} \cdot s^{O(1)}$.*

Moreover, there is a randomized algorithm that list recovers $\mathsf{MULT}_{q,s}^{(1)}(d)$ *with the above parameters in time* $\mathsf{poly}(q, \ell^{O(s \log s)})$.

The proof of this theorem is again in two steps.

First we use the linear-algebraic approach to list-recovering univariate multiplicity codes [GW13] to show that the list is contained in a subspace. Technically, we need to redo this proof using some additional algebraic ideas, because it will be important for us to deal with polynomials of degree $\gg q$, and [GW13] only worked for polynomials of degree at most $\mathsf{char}(\mathbb{F}_q)$. As a consequence, the low-dimensional subspace will be of noticeably higher dimension than in [GW13], but will be of a special form.

Next we show that the output list $\mathcal{L}$ cannot contain too many elements from a low-dimensional subspace of this special form. As before, we will do this via a randomized algorithm.

4.3 Output list is contained in a special subspace

Suppose we are given a received word $S : \mathbb{F}_q \to \binom{\mathbb{F}_q^s}{\ell}$ for the univariate multiplicity code with degree d and multiplicity parameter $s < q$.

The following theorem is essentially Lemma 14 from [GW13]. (This lemma is part of the proof of Theorem 17 of [GW13], which is used in the proof of our Theorem 4.1 above). The theorem gives a special affine subspace $v_0 + V$ of $\mathbb{F}_q[X]$ which contains all $f(X)$ whose codeword $\mathsf{MultEnc}_s(f)$ is close to S.

The main differences between the following theorem and Lemma 14 of [GW13] are: (1) we need to talk about list-recovery, not just list-decoding, and (2) we work with Hasse derivatives,

while [GW13] works with standard derivatives. Both differences are minor; for completeness we include a proof in the appendix.

Theorem 4.5. *Let $S : \mathbb{F}_q \to \binom{\mathbb{F}_q^s}{\ell}$. Let*

$$\alpha < 1 - \frac{\ell}{r+1} - \frac{r}{r+1} \cdot \frac{s}{s-r+1} \cdot \frac{d}{sq}. \tag{2}$$

Let

$$\mathcal{L} = \{g(X) \in \mathbb{F}_q[X] \mid \deg(g) \leq d \text{ and } \operatorname{dist}(\mathsf{MultEnc}_s(g), S) \leq \alpha\}.$$

There is an algorithm A, which when given as input r, finds polynomials $A(X), B_0(X), \ldots, B_{r-1}(X)$ such that the affine space:

$$v_0 + V = \{f(X) \in \mathbb{F}_q[X] \mid \deg(f) \leq d \text{ and } A(X) + \sum_{i=0}^{r-1} B_i(X) f^{(i)}(X) = 0\}$$

satisifies:

$$\mathcal{L} \subseteq v_0 + V.$$

4.4 Special subspaces

In this subsection, we study certain special linear spaces of polynomials. In the next subsection we show how this is relevant to the kinds of spaces $v_0 + V$ returned by the algorithm of Theorem 4.5.

Definition 4.6 ((X^q, d)-closed)**.** *A subspace $W \subseteq \mathbb{F}_q[X]$ consisting of polynomials of degree at most d is called (X^q, d)-closed if for every $f(X) \in W$ with $\deg(f) \leq d - q$, we have*

$$f(X) \cdot X^q \in W.$$

For a (X^q, d)-closed subspace W, we define the q-dimension by

$$\operatorname{qdim}(W) = |\{\deg(f) \mod q \mid f \in W \setminus \{0\}\}|.$$

Observe that if W is (X^q, d)-closed with $d \leq sq$, then

$$\dim(W) \leq s \cdot \operatorname{qdim}(W).$$

The next lemma gives a nice basis for every (X^q, d)-closed subspace.

Lemma 4.7. *Suppose $W \subseteq \mathbb{F}_q[X]$ is (X^q, d)-closed. Then there exist $f_1, \ldots, f_{t'} \in W$ such that:*

1. *$\deg(f_i) \not\equiv \deg(f_j) \mod q$ for all $i \neq j$.*

2. *Every $f \in W$ can be uniquely written as:*

$$f(X) = \sum_{i=1}^{t'} C_i(X^q) f_i(X),$$

where for all i, $C_i(Y) \in \mathbb{F}_q[Y]$ with $q \cdot \deg(C_i) + \deg(f_i) \leq d$.

Furthermore, we have $\mathrm{qdim}(W) = t'$.

Proof. Let f_1 be the lowest degree nonzero element of W, and let $M_1 = \{C_1(X^q)f_1(X) \mid C_1(Y) \in \mathbb{F}_q[Y]$ and $q \deg(C_1) + \deg(f_1) \leq d\}$. Note that $M_1 \subseteq W$. If $M_1 = W$ we are done with $r' = 1$.

Otherwise, we proceed. Let f_2 be the lowest degree nonzero element of $W \setminus M_1$. Observe that $\deg(f_2) \not\equiv \deg(f_1) \mod q$: otherwise for some $a \in \mathbb{F}_q$ and $b \geq 0$, we would have that

$$f_2'(X) = f_2(X) - aX^{bq}f_1(X) \in f_2 + M_1$$

is an element of $W \setminus M_1$ with even lower degree than $f_2(X)$, contradicting the choice of f_2. Define $M_2 = \{C_1(X^q)f_1(X) + C_2(X^q)f_2(X) \mid$ For $i \leq 2$ we have $C_i(Y) \in \mathbb{F}_q[Y]$ and $q \deg(C_i) + \deg(f_i) \leq d\}$. Note that $M_2 \subseteq W$. If $M_2 = W$, we are done with $r' = 2$.

Repeating this argument, we get polynomials $f_1, f_2, \ldots, f_{t'} \in \bar{V}$ with degrees $d_1, d_2, \ldots, d_{t'}$ satisfying:

$$d_i \not\equiv d_j \mod q \qquad \forall i \neq j$$

This implies that the polynomials $X^{cq}f_i(X)$ all have distinct degrees, and are thus linearly independent. Furthermore, we have that W is equal to

$$M_{t'} = \left\{ \sum_{i=1}^{t'} C_i(X^q)f_i(X) \mid \text{For } i \leq t' \text{ we have } C_i(Y) \in \mathbb{F}_q[Y] \text{ and } q \deg(C_i) + \deg(f_i) \leq d \right\}.$$

In particular, if we define $j_i = \left\lfloor \frac{d - \deg(f_i)}{q} \right\rfloor$, then

$$\{X^{cq}f_i(X) \mid q \leq i \leq t', 0 \leq c \leq j_i\}$$

is a basis for W. $\qquad\square$

The next theorem is a variant of Theorem 14 from [GK16b] for (X^q, d)-closed subspaces of polynomials of degree $d > q$.

Theorem 4.8. *Let $W \subseteq \mathbb{F}_q[X]$ be a (X^q, d)-closed linear subspace of polynomials of degree at most d. Suppose $\dim(W) = t$ and $\mathrm{qdim}(W) = t' \leq s$.*

Suppose $d \leq (s - t')q$. Then:

$$\mathbb{E}_{b \in \mathbb{F}_q}[\dim(W \cap H_b)] \leq (1 - 1/s) \cdot t.$$

Proof. Let $f_1, \ldots, f_{t'}$ be those given by Lemma 4.7. Let $d_i = \deg(f_i)$, and assume that $d_1 \leq d_2 \leq \ldots \leq d_{t'}$. Let $U = \mathsf{span}\{f_1, \ldots, f_{t'}\}$.

For $b \in \mathbb{F}_q$, consider the map $\Phi_b : W \to \mathbb{F}_q^s$ given by

$$\Phi(f) = f^{(<s)}(b).$$

Then $\ker(\Phi_b) = W \cap H_b$. Thus $\dim(W \cap H_b) = \dim(W) - \dim(\Phi(W))$. Similarly, $\dim(U \cap H_b) = \dim(U) - \dim(\Phi(U))$. Thus

$$\dim(W \cap H_b) \leq \dim(W) - \dim(U) + \dim(U \cap H_b).$$

Below we will show that

$$\dim(U) - \mathbb{E}_b[\dim(U \cap H_b)] \geq \frac{t}{s - t' + 1}. \tag{3}$$

Assuming this, we get that

$$\mathbb{E}[\dim(W \cap H_b)] \leq t - \frac{t}{s - t' + 1} \leq \left(1 - \frac{1}{s - t' + 1}\right) t \leq \left(1 - \frac{1}{s}\right) t,$$

which is what we wanted to prove.

We now prove Equation (3). Let $D = \sum_{i=1}^{t'} d_i$. By Item 2 of Lemma 4.7, we have that:

$$t = \sum_{i=1}^{t'} \left(1 + \left\lfloor \frac{d - d_i}{q} \right\rfloor\right)$$

$$\leq t'\left(1 + \frac{d}{q}\right) - \frac{D}{q}.$$

Let $Q(X)$ be the determinant of the Wronskian matrix of $f_1, \ldots, f_{t'}$. By Lemma A.1, $Q(X)$ is a nonzero polynomial. We have:

$$\deg(Q) \leq D.$$

By Claim 19 from [GK16b], we have that

$$\mathsf{mult}(Q, b) \geq (s - t' + 1) \cdot \dim(U \cap H_b).$$

Now using the fact that $\mathbb{E}_{b \in \mathbb{F}_q}[\mathsf{mult}(Q, b)] \leq \frac{\deg(Q)}{q} \leq \frac{D}{q}$, we get:

$$\mathbb{E}_{b \in \mathbb{F}_q}[\dim(U \cap H_b)] \leq \frac{1}{s - t' + 1}\mathbb{E}_{b \in \mathbb{F}_q}[\mathsf{mult}(Q, b)] \leq \frac{1}{s - t' + 1} \cdot \frac{D}{q}.$$

Thus

$$\dim(U) - \mathbb{E}_{b \in \mathbb{F}_q}[\dim(U \cap H_b)] \geq t' - \frac{1}{s - t' + 1}\frac{D}{q}.$$

Thus

$$(s - t' + 1) \cdot \left(\dim(U) - \mathbb{E}_{b \in \mathbb{F}_q}[\dim(U \cap H_b)]\right) \geq (s - t' + 1)t' - \frac{D}{q}$$

$$\geq t'\left(1 + \frac{d}{q}\right) - \frac{D}{q} \qquad \text{By the assumption } d \leq (s - t')q$$

$$\geq t.$$

This completes the proof of the lemma. $\qquad\square$

4.5 Properties of the space of solutions of a linear differential equation

Let $v_0 + V$ be the affine space of low degree solutions to a linear differential equation:

$$v_0 + V = \left\{ f(X) \in \mathbb{F}_q[X] \mid \deg(f) \le d \text{ and } A(X) + \sum_{i=0}^{r-1} B_i(X) f^{(i)}(X) = 0 \right\}.$$

We now prove some properties of V that will help us in the second step of our list-decoding algorithm.

First we note that V is (X^q, d)-closed.

Lemma 4.9. V is (X^q, d)-closed.

Proof. Observe that:

$$V = \left\{ f(X) \in \mathbb{F}_q[X] \mid \deg(f) \le d \text{ and } \sum_{i=0}^{r} B_i(X) f^{(i)}(X) = 0 \right\}.$$

Now take any $f(X) \in V$. We want to show that $g(X) = X^q f(X) \in V$. For any $i < q$, we have:

$$
\begin{aligned}
g^{(i)}(X) &= \sum_{i'=0}^{i} (X^q)^{(i')} \cdot f^{(i-i')}(X) \\
&= \sum_{i'=0}^{i} \binom{q}{i'} X^{q-i'} \cdot f^{(i-i')}(X) \\
&= X^q f^{(i)}(X).
\end{aligned}
$$

Thus (using the fact that $r < q$):

$$\sum_{i=0}^{r} B_i(X) g^{(i)}(X) = X^q \left(\sum_{i=0}^{r} B_i(X) f^{(i)}(X) \right) = 0.$$

We conclude that $g(X) \in V$, as desired. $\qquad\square$

Lemma 4.10.
$$\mathrm{qdim}(V) \le r - 1.$$

Proof. Apply Lemma 4.7 to V to obtain $f_1, \ldots, f_{r'} \in V$, where $r' = \mathrm{qdim}(V)$.

Suppose $r' \ge r$. Since $f_1, \ldots, f_r \in V$ we have that for each $j \in [r]$,

$$\sum_{i=0}^{r-1} B_i(X) f_j^{(i)}(X) = 0.$$

This means that the Wronskian matrix of $(f_1, \ldots, f_r)$ is singular. However, Lemma A.1 shows that the Wronskian matrix is nonsingular, a contradiction. This completes the proof. $\qquad\square$

The next two lemmas and the following corollary are trivial and we omit the proofs.

For $b \in \mathbb{F}_q$, let
$$H_b = \{f(X) \in \mathbb{F}_q[X] \mid \deg(f) \le d \text{ and } f^{(<s)}(b) = 0\}.$$

Lemma 4.11. *For each $b \in \mathbb{F}_q$, H_b is (X^q, d)-closed.*

Lemma 4.12. *The intersection of (X^q, d)-closed subspaces is (X^q, d)-closed.*

Corollary 4.13. *Given elements $b_1, \ldots, b_j \in \mathbb{F}_q$, the space $V \cap H_{b_1} \cap H_{b_2} \ldots H_{b_j}$ is (X^q, d)-closed.*

4.6 Pruning the list

Having developed the relevant machinery about (X^q, d)-closed subspaces, we can now describe and analyze the second part of the list-decoding algorithm for univariate multiplicity codes. Below we give the algorithm PruneListMULT, and after that we analyze it. The algorithm is given as input $S : \mathbb{F}_q \to \binom{\mathbb{F}_q^s}{\ell}$, an $\mathbb{F}_q$-affine subspace $v_0 + V \subseteq \mathbb{F}_q[X]$, and a parameter $\tau \in \mathbb{N}$.

Algorithm PruneListMULT$(S, v_0 + V, \tau)$

1. Initialize $\mathcal{L}' = \emptyset$.

2. Pick $b_1, b_2, \ldots, b_\tau \in \mathbb{F}_q$ independently and uniformly at random.

3. For each choice of $y_1 \in S(b_1), y_2 \in S(b_2), \ldots, y_\tau \in S(b_\tau)$:

   - If there is exactly one codeword $P(X) \in v_0 + V$ such that $P^{(<s)}(b_j) = y_j$ for all $j \in [\tau]$, then:
     $$\mathcal{L}' \leftarrow \mathcal{L}' \cup \{P(X)\}.$$

4. Output $\mathcal{L}'$.

Lemma 4.14. *Suppose $v_0 + V$ is an affine space of polynomials of degree at most d. Suppose V is (X^q, d)-closed and $\mathrm{qdim}(V) = r'$. Suppose $d \le (s - r')q$.*

Then the algorithm PruneListMULT runs in time $\mathsf{poly}(q, s, \ell^\tau)$, and outputs a list $\mathcal{L}'$ containing at most ℓ^τ polynomials, such that any polynomial $P(X) \in v_0 + V$ with $\mathrm{dist}(\mathsf{MultEnc}_s(P), S) \le \alpha$ appears in $\mathcal{L}'$ with probability at least
$$(1 - \alpha)^\tau - r's(1 - 1/s)^\tau.$$

Proof. We clearly have that $|\mathcal{L}'| \le \ell^\tau$, and that the algorithm has the claimed running time. Fix a polynomial $\hat{P} \in v_0 + V$ such that $\mathrm{dist}(\mathsf{MultEnc}_s(\hat{P}), S) \le \alpha$. Below we will show that $\hat{P}$ belongs to $\mathcal{L}'$ with probability at least
$$(1 - \alpha)^\tau - r's(1 - 1/s)^\tau.$$

Let E_1 denote the event that $\hat{P}^{(<s)}(b_j) \in S(b_j)$ for all $j \in [\tau]$. Let E_2 denote the event that for all nonzero polynomials $Q \in V$ there exists some $j \in [\tau]$ such that $Q^{(<s)}(b_j) \ne 0$. By assumption that

$\text{dist}(\mathsf{MultEnc}_s(\hat{P}), S) \leq \alpha$, we readily have that

$$\Pr[E_1] \geq (1 - \alpha)^\tau.$$

Claim 4.15 below also shows that

$$\Pr[E_2] \geq 1 - r's \cdot (1 - 1/s)^\tau.$$

So both E_1 and E_2 occur with probability at least

$$(1 - \alpha)^\tau - r's \cdot (1 - 1/s)^\tau.$$

If E_2 occurs, then for every choice of $y_1 \in S(b_1), y_2 \in S(b_2), \ldots, y_\tau \in S(b_2)$, there can be at most one polynomial $P(X) \in v_0 + V$ such that $P^{(<s)}(b_j) = y_j$ for all $j \in [\tau]$ (otherwise, the difference $Q = P_1 - P_2 \in V$ of two such distinct polynomials would have $Q^{(<s)}(b_j) = 0$ for all $j \in [\tau]$, contradicting E_2). If E_1 also occurs, then in the iteration of Step 3 where $y_j = \hat{P}^{(<s)}(b_j)$ for each $j \in [\tau]$, the algorithm will take $P = \hat{P}$, and thus $\hat{P}$ will be included in $\mathcal{L}'$. This completes the proof of the lemma. $\qquad\square$

It remains to prove the following claim.

Claim 4.15.

$$\Pr[E_2] \geq 1 - r's \cdot (1 - 1/s)^\tau.$$

Proof of Claim 4.15. We will use Theorem 4.8.

Since $\text{qdim}(V) \leq r'$, we have $\dim(V) \leq r's$. For $0 \leq j \leq \tau$, let

$$V_j := V \cap H_{b_1} \cap H_{b_2} \cap \ldots \cap H_{b_j},$$

and $t_j := \dim(V_j)$. Observe that $r's \geq t_0 \geq t_1 \geq \ldots \geq t_\tau$, and that event E_2 holds if and only if $t_\tau = 0$.

By Corollary 4.13, all the V_j are (X^q, d)-closed. Since $V_j \subseteq V$, we have $\text{qdim}(V_j) \leq \text{qdim}(V) = r'$.

We now study the distribution of $t_0, \ldots, t_\tau$. Since $\text{qdim}(V_j) \leq r'$ and $d \leq (s - r')q$, we may apply Theorem 4.8. We get:

$$\mathbb{E}[t_{j+1} \mid t_j = t] = \mathbb{E}_{b_{j+1} \in \mathbb{F}_q}[\dim(V_j \cap H_{b_j}|) \mid \dim(V_j) = t] \leq (1 - 1/s) \cdot t.$$

Thus

$$\mathbb{E}[t_{j+1}] \leq (1 - 1/s) \cdot \mathbb{E}[t_j],$$

and

$$\mathbb{E}[t_\tau] \leq (1 - 1/s)^\tau \cdot \mathbb{E}[t_0] \leq (1 - 1/s)^\tau \cdot r's.$$

Finally, by Markov's inequality this implies in turn that

$$\Pr[E_2] = \Pr[t_\tau = 0] = 1 - \Pr[t_\tau \geq 1] \geq 1 - r's (1 - 1/s)^\tau.$$

$\qquad\square$

4.7 Proof of Theorem 4.4

We now prove Theorem 4.4 based on Theorem 4.5 and Lemma 4.14.

Proof of Theorem 4.4. Let $S : \mathcal{D} \to \binom{\mathbb{F}_q^s}{\ell}$ be the received sequence of input lists. We would like to find a list $\mathcal{L}$ of size $\ell^{O(s \log s)} \cdot s^{O(1)}$ that contains all polynomials $P(X)$ of degree at most d with $\mathrm{dist}(\mathsf{MultEnc}_s(P), S) \leq \alpha$.

Let $v_0 + V$ be the affine subspace found by the algorithm of Theorem 4.5 for S and $r = 4\ell/\delta$ (and so $s > \frac{4r}{\delta}$ by our assumption that $s > 16\ell/\delta^2$). Note that for this choice of r the RHS of (2) is at least

$$1 - \frac{\delta}{4} - \frac{1}{1 - \delta/4}(1 - \delta) > \frac{\delta}{3} > \alpha,$$

and so all polynomial $P(X)$ of degree at most d with $\mathrm{dist}(\mathsf{MultEnc}_s(P), S) \leq \alpha$ are included in $v_0 + V$.

By Lemma 4.10, $\mathrm{qdim}(V) \leq r$. Since V is (X^q, d)-closed, $\mathrm{qdim}(V) \leq r$, and

$$d \leq (1 - \delta)sq \leq (s - 4r)q \leq (s - r)q,$$

we may invoke Lemma 4.14. It tells us that algorithm $\mathsf{PruneListMULT}$ with inputs S, $v_0 + V$ and $\tau = O(s \log(2rs))$ returns a list $\mathcal{L}'$ of size at most ℓ^τ such that each polynomial $P(X)$ of degree at most d with $\mathrm{dist}(\mathsf{MultEnc}_s(P), S) \leq \alpha$ is included in $\mathcal{L}'$ with probability p_0, where:

$$p_0 \geq (1 - \alpha)^\tau - rs\left(1 - \frac{1}{s}\right)^\tau$$
$$\geq (1 - \frac{1}{2s})^\tau - rs\left(1 - \frac{1}{s}\right)^\tau$$
$$\geq \frac{1}{2}(1 - \frac{1}{2s})^\tau,$$

where the first inequality holds since $\alpha \leq \frac{1}{2s}$, and the second inequality holds since

$$(1 - \frac{1}{2s}) > (1 + \frac{1}{2s})(1 - \frac{1}{s}) > e^{1/6s}(1 - \frac{1}{s}) > e^{\ln(2rs)/\tau}(1 - \frac{1}{s}).$$

The above implies in turn that

$$|\mathcal{L}| \leq \frac{|\mathcal{L}'|}{p_0} \leq 2\left(\frac{\ell}{1 - \frac{1}{2s}}\right)^\tau \leq \ell^{O(s \log s)} s^{O(1)}.$$

Moreover, by running the algorithm $\mathsf{PruneListMULT}$ $O(\frac{1}{p_0} \log |\mathcal{L}|)$ times and returning the union of all output lists, by a union bound, all elements of $\mathcal{L}$ will appear in the union of the output lists with high probability (say, at least 0.99). This gives a randomized list recovery algorithm with output list size $\ell^{O(s \log s)}$ and running time $\mathsf{poly}(q, s, \ell^{O(s \log s)})$. $\square$

5 Local list-recovery of multivariate multiplicity codes

In this section we show that multivariate multiplicity codes can be locally list recovered from constant sized input lists with small (or even constant) sized output lists.

Let $\delta > 0$ be a parameter, and m, s be integers. Let $d = (1 - \delta)sq$. Let $\mathsf{MULT}_{q,s}^{(m)}(d)$ be the corresponding multiplicity code. Given ℓ, α, we will demonstrate a local list recovery algorithm for multivariate multiplicity codes of degree d and multiplicity parameter s, with input list size ℓ and error tolerance α. Our main technical statement is Lemma 5.15, which we will state and prove later in this section. However, we first state two instantiations of Lemma 5.15, which show that (a) multivariate multiplicity codes are locally list-recoverable with sub-polynomial query complexity and sub-polynomial list size, and (b) multivariate multiplicity codes are locally list-recoverable with polynomial query complexity and constant list size.

Our first statement establishes sub-polynomial query complexity and list size.

Theorem 5.1. *Let $\varepsilon > 0$ be sufficiently small, and let $m, \ell > 0$ be any integers.*

Then there is a multivariate multiplicity code $C \subseteq \Sigma^N$ with distance $\delta = \varepsilon/2m$ and rate at least $1 - \varepsilon$ so that C is (t, α, ℓ, L)-locally-list-recoverable for

$$\alpha \leq \frac{\delta^2}{160\ell},$$

$$L = \ell^{O(\ell m^2 \log(\ell m/\varepsilon)/\varepsilon^2)} \cdot \left(\frac{m}{\varepsilon}\right)^{O(1)}$$

and

$$t = \left(\frac{Lm\ell}{\varepsilon}\right)^{O(m)}.$$

Moreover, we have

$$N = \left(\frac{Lm\ell}{\varepsilon}\right)^{O(m^2)}.$$

and

$$|\Sigma| = \left(\frac{Lm\ell}{\varepsilon}\right)^{O((\ell m^2/\varepsilon^2)^m)}.$$

The code is explicit, and can be locally-list-recovered in time $\mathsf{poly}(t)$.

In particular, if ε, ℓ are constant, and $\ell > 1$, we have

$$L = m^{O(m^2)}, \qquad t = m^{O(m^3)}, \qquad |\Sigma| = m^{O(m^{2m+2})}, \qquad N = m^{O(m^4)},$$

and so

$$L = \exp\left(\sqrt{\log(N) \cdot \log\log(N)}\right), \qquad t = \exp\left(\log^{3/4}(N) \cdot (\log\log(N))^{1/4}\right),$$

$$|\Sigma| = \exp\left(\exp\left(\log^{1/4}(N) \cdot \log\log^{3/4}(N)\right)\right).$$

Our second theorem establishes constant output list size with polynomial query complexity.

30

Theorem 5.2. *Let $\delta, \varepsilon > 0$ be sufficiently small, and choose $\tau \in (0, 1)$ and $\ell \in \mathbb{N}$.*

Then for infinitely many $N \in \mathbb{N}$, there is a multivariate multiplicity code $C \subseteq \Sigma^N$ with distance δ and rate at least $(1 - \delta)^{O(1/\tau)}(1 - \varepsilon)$ so that C is (t, α, ℓ, L)-locally-list-recoverable for some

$$\alpha = \Omega \left(\min \left\{ \frac{\delta^2}{\ell}, \tau^2 \varepsilon \right\} \right),$$

with

$$t = O_{\delta, \varepsilon, \ell, \tau}(N^\tau),$$
$$L = O_{\delta, \varepsilon, \ell, \tau}(1),$$

and

$$|\Sigma| = \mathsf{poly}_{\delta, \varepsilon, \ell, \tau}(N).$$

Notice that the alphabet size in Theorem 5.2 is large (polynomial in N) and in Theorem 5.1 is very large (super-polynomial in N). However, we will deal with this in Section 6 to obtain capacity-achieving codes with the roughly the same parameters t and L, and with constant alphabet size.

The rest of this section is devoted to the proof of Theorems 5.1 and 5.2, both of which follow from the more general Lemma 5.15 below. We give a short overview of the approach in Section 5.1, and then flesh out the details in the subsequent three subsections.

5.1 Overview and some more notation

We first introduce some terminology that will be useful for this section. Let $U_{m,s} = \{\mathbf{i} \in \mathbb{N}^m \mid \mathsf{wt}(\mathbf{i}) < s\}$. Let $\Sigma_{m,s} = \mathbb{F}_q^{U_{m,s}}$ be the alphabet of the multiplicity code. Let $w = |U_{m,s}| = \binom{m+s-1}{m}$. For an element $z \in \Sigma_{m,s}$, and a direction $\mathbf{b} \in \mathbb{F}_q^m$, we define the restriction of z to direction $\mathbf{b}$ (denoted $z|_{\mathbf{b}}$) to equal $h \in \Sigma_{1,s}$, where given by:

$$h^{(j)} = \sum_{\mathsf{wt}(\mathbf{j})=j} z^{(\mathbf{j})} \mathbf{b}^{\mathbf{j}}$$

for each j such that $0 \le j < s$.

The local list recovering algorithm has three main subroutines that we will describe and analyze in the next three subsections. Briefly, the three components are the following:

1. A subroutine RecoverCandidates, given in Section 5.2. RecoverCandidates takes as input a point $\mathbf{a} \in \mathbb{F}_q^m$, has query access to S, and returns a short list $Z \subset \Sigma_{m,s^*}$ of guesses for $Q^{(<s^*)}(\mathbf{a})$, where we will take s^* to be some parameter larger than s.

2. An oracle machine $M^S[\mathbf{a}, z]$, given in Section 5.3. The oracle machine $M^S[\mathbf{a}, z]$ is defined using an advice string $(\mathbf{a}, z)$ and has query access to S. If $z = Q^{(<s^*)}(\mathbf{a})$, then with high probability over the choice of a random point $\mathbf{a}$, we weill have that

$$\mathrm{dist}\left(M^S[\mathbf{a}, Q^{(<s^*)}(\mathbf{a})], Q^{(<s)} \right) \le 4\varepsilon.$$

3. The final local list-recovery algorithm LocalListRecoverMULT, given in Section 5.4. Recall that the goal is to output a list of randomized algorithms $A_1, \ldots, A_L$ so that for each codeword $c \in \mathsf{MULT}_{q,s}^{(m)}(d)$ with $\mathrm{dist}(c, S) \leq \alpha$, with probability at least $2/3$, there exists some j so that $\Pr[A_j(i) = c_i] \geq 2/3$. We arrive at these algorithms A_j as follows.

   First, the algorithms runs RecoverCandidates on a random point $\mathbf{a} \in \mathbb{F}_q^m$ to generate a short list $Z \subseteq \Sigma_{m,s^*}$ of possibilities for $Q^{(<s^*)}(\mathbf{a})$. Then, for each $z \in Z$, it forms the oracle machine $M^S[\mathbf{a}, z]$. At this point it would be tempting to output the list of these oracle machines, but we are not quite done: even if $z = Q^{(<s^*)}(\mathbf{a})$ corresponds to the correct advice and the choice of $\mathbf{a}$ is good, for some small fraction of points $\mathbf{x}$, we may still have $M^S[\mathbf{a}, z](\mathbf{x}) \neq Q^{(<s)}(\mathbf{x})$ with decent probability. Fortunately, for most $\mathbf{y}$ this will not be the case, and so we can implement the local correction algorithm of [KSY14] for multiplicity codes on top of $M^S[\mathbf{a}, z]$. This will give us our final list of randomized algorithms $A_1, \ldots, A_L$ that the local list-recovery algorithm returns.

We flesh out the details in the next three subsections.

5.2 The algorithm RecoverCandidates

As an important subroutine of the local list recovering algorithm, we will implement an algorithm which we call RecoverCandidates which will have the following features. It will have oracle access to a function $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$. Think of this function as assigning to each element of $\mathbb{F}_q^m$ a list of size ℓ of alphabet symbols of the multiplicity code. Now suppose that Q is an m-variate polynomial of degree at most d (think of Q to represent a true codeword of the multiplicity code) that "agrees" with at least $1 - \alpha$ fraction of these lists. On being input $\mathbf{x}$, a random element of $\mathbb{F}_q^m$, and for some parameter $\tilde{s}$ (think of $\tilde{s}$ to be much larger than s), the algorithm RecoverCandidates will make few queries to S and output a small list $Z \subseteq \Sigma_{m,\tilde{s}}$, such that with high probability (over the choice of $\mathbf{x}$ and the randomness of the algorithm), the list Z contains $Q^{(<\tilde{s})}(\mathbf{x})$.

The main feature of this algorithm is that given oracle access to small lists, that for most coordinates agree with the evaluations of order s derivatives of Q, can output for most coordinates, a small list that agrees with evaluations of order $\tilde{s}$ derivatives of Q.

Lemma 5.3. *Let q be a prime, let $\delta > 0$ and let s, d, m be nonnegative integers such that $d = (1 - \delta)sq$. Let α, α' be parameters such that $0 < \alpha < \alpha'$.*

Let $L = L(d, q, s, \ell, \alpha')$ be the list size for list recovering univariate multiplicity codes $\mathsf{MULT}_{q,s}^{(1)}(d)$ of degree d and multiplicity parameter s with input list size ℓ and error tolerance α', and T be the corresponding running time for list recovering univariate multiplicity codes of the same parameters.

Let $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$. Let $\tilde{s} > 0$ be a parameter. Suppose that $q > 100 \cdot \tilde{s} \cdot L \cdot m^2$, and suppose that $Q(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ is a polynomial of degree at most d such that:

$$\Pr_{\mathbf{x} \in \mathbb{F}_q^m}[Q^{(<s)}(\mathbf{x}) \in S(\mathbf{x})] > 1 - \alpha.$$

There is an algorithm RecoverCandidates which on input $x \in \mathbb{F}_q^m$ and $\tilde{s}$, and given oracle access to S, makes at most $\mathsf{poly}(q \cdot (L\tilde{s}m)^m)$ queries to S, runs in time at most $\mathsf{poly}(T \cdot q \cdot (L\tilde{s}m)^m)$, and

outputs a list $Z \subseteq$ of size at most $O(L)$ such that if $\mathbf{x} \in \mathbb{F}_q^m$ uniformly at random, then:

$$\Pr[Q^{(<\tilde{s})}(\mathbf{x}) \in Z] \geq 1 - \frac{12}{(\alpha' - \alpha)q},$$

where the probability is over the random choice of $\mathbf{x}$ and the random choices of the algorithm RecoverCandidates.

The high level idea of the algorithm is as follows. On input $\mathbf{x}$, we take several random lines passing through $\mathbf{x}$, and run the univariate multiplicity list-recovery algorithm on the restrictions of the received word to those lines. This gives us, for each of these lines, a list of univariate polynomials. For a given line, this list of univariate polynomials contains candidates for Q restricted to that line. In particular, this gives us candidate values for $Q(\mathbf{x})$ and the all higher order directional derivatives of Q at $\mathbf{x}$ in the directions of those lines. We combine this information about the different directional derivatives to reconstruct $Q^{(<\tilde{s})}(\mathbf{x})$.

This combination turns out to be a certain kind of polynomial list-recovery problem: namely list-recovery for tuples of polynomials. The following lemma, which is proved in Appendix D, shows how this can be done algorithmically.

Lemma 5.4 (Vector-valued Reed-Muller list recovery on a grid)**.** *Let $\ell, \tilde{s}, K, m$ be given parameters. Let $\mathbb{F}$ be a finite field. Suppose that $U \subseteq \mathbb{F}$ and $|U| \geq 2\ell\tilde{s}K$. Let $\alpha < 1 - \frac{1}{\sqrt{K}}$ be a parameter.*

Then for every $f : U^m \to \binom{\mathbb{F}^t}{\ell}$, if

$$\mathcal{L} = \{(Q_1, Q_2, \ldots, Q_t) \in (\mathbb{F}[Y_1, Y_2, \ldots, Y_t])^t \mid \forall i \in [t], \deg(Q_i) \leq \tilde{s} \text{ and}$$
$$\Pr_{\mathbf{u} \in U^m}[(Q_1(\mathbf{u}), Q_2(\mathbf{u}), \ldots, Q_t(\mathbf{u})) \notin f(\mathbf{u})] < \alpha\},$$

the following hold:

1. *$|\mathcal{L}| \leq 2K\ell$.*

2. *If $K \geq m^2$ and $\alpha < 1 - \frac{m}{\sqrt{K}}$, then there is a $\mathsf{poly}(|U|^m, t, \log|F|)$-time algorithm* VectorRMListRecover *which computes $\mathcal{L}$.*

We will use VectorRMListRecover as a subroutine of RecoverCandidates. To see why this is relevant, we make the following observation about Hasse derivatives.

Claim 5.5. *Let $Q(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$. Let $\mathbf{x}, \mathbf{b} \in \mathbb{F}_q^m$. Let $\tilde{s}$ be an integer. Let $\lambda(T) = \mathbf{x} + T\mathbf{b}$ be the line passing through $\mathbf{x}$ in direction $\mathbf{b}$.*

For each $j < \tilde{s}$, define polynomial

$$H_{\mathbf{x},j}(Y_1, \ldots, Y_m) = \sum_{\mathbf{j}:\mathsf{wt}(\mathbf{j})=j} Q^{(\mathbf{j})}(\mathbf{x})\mathbf{Y}^{\mathbf{j}},$$

and let $H_{\mathbf{x}}(Y_1, \ldots, Y_m) \in (\mathbb{F}_q[Y_1, \ldots, Y_m])^{\tilde{s}}$ be the tuple of polynomials:

$$H_{\mathbf{x}} = (H_{\mathbf{x},0}, H_{\mathbf{x},1}, \ldots, H_{\mathbf{x},\tilde{s}-1}).$$

Then we have

$$(Q \circ \lambda)^{(<\tilde{s})}(0) = Q^{(<\tilde{s})}(\mathbf{x})|_{\mathbf{b}} = H_{\mathbf{x}}(\mathbf{b}).$$

Proof. We have from the definition of the Hasse derivative that

$$\sum_i (Q \circ \lambda)^{(i)}(0) \cdot T^i = (Q \circ \lambda)(T)$$

$$= Q(\mathbf{x} + T\mathbf{b})$$

$$= \sum_{\mathbf{i}} Q^{(\mathbf{i})}(\mathbf{x}) \cdot \mathbf{b^i} \cdot T^{\mathsf{wt}(\mathbf{i})}$$

$$= \sum_i \left(\sum_{\mathbf{i}:\mathsf{wt}(\mathbf{i})=i} Q^{(\mathbf{i})}(\mathbf{x})\mathbf{b^i} \right) T^{\mathsf{wt}(\mathbf{i})},$$

and so by matching coefficients we have

$$(Q \circ \lambda)^{(j)}(0) = \sum_{\mathbf{j}:\mathsf{wt}(\mathbf{j})=j} Q^{(\mathbf{j})}(\mathbf{x})\mathbf{b^j} = H_{\mathbf{x},j}(\mathbf{b}).$$

This implies the claim. $\qquad\square$

Thus, given information about $(Q \circ \lambda)^{(<\tilde{s})}(0)$ for various lines λ and for some $\tilde{s}$, we have information about the tuple of polynomials $H_{\mathbf{x}}(\mathbf{Y})$, evaluated at many different points $\mathbf{b}$. It is on these polynomials that we will use Lemma 5.4.

Now we present our main subroutine RecoverCandidates, and analyze it below.

Main Subroutine RecoverCandidates.

- Oracle access to $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$.

- **INPUT:** $\mathbf{x} \in \mathbb{F}_q^m$, parameter $\tilde{s} \in \mathbb{N}$.

- The goal is to recover a small list of candidates for $Q^{(<\tilde{s})}(\mathbf{x})$.

1. Let $U \subseteq \mathbb{F}_q$ be a set of size $100\tilde{s}Lm^2$.

2. Let $\mathbf{b} \in \mathbb{F}_q^m$ be picked uniformly at random.

3. Let $B = \{\mathbf{b_u} = \mathbf{b} + \mathbf{u} \mid \mathbf{u} \in U^m\}$.

4. For each $\mathbf{u} \in U^m$:

   (a) Let $\lambda_{\mathbf{u}}(T)$ be the line $\lambda_{\mathbf{u}}(T) = \mathbf{x} + T\mathbf{b_u}$.
   (b) Consider the restriction $S_{\mathbf{u}} : \mathbb{F}_q \to \binom{\Sigma_{1,s}}{\ell}$ of S to $\lambda_{\mathbf{u}}$. Formally:

   $$S_{\mathbf{u}} = S \circ \lambda_{\mathbf{u}}(t) = \{z|_{\mathbf{b_u}} \mid z \in S(\lambda_{\mathbf{u}}(t))\}.$$

(c) Run the univariate list recovery algorithm on $S_{\mathbf{u}}$ with error-tolerance α' for degree d polynomials to obtain a list $\mathcal{L}_{\lambda_{\mathbf{u}}} \subseteq \mathbb{F}_q[T]$.

$.$

5. Define a function $f : U^m \to \binom{\mathbb{F}_q^{\tilde{s}}}{L}$ as follows. For each $\mathbf{u} \in U^m$, define

$$f(\mathbf{u}) = \{P^{(<\tilde{s})}(0) \mid P(T) \in \mathcal{L}_{\lambda_{\mathbf{u}}}\}.$$

6. Let $\mathcal{L}'$ be the set of all $\tilde{s}$-tuples of polynomials

$$(Q'_j(Y_1, \ldots, Y_m))_{j=0}^{\tilde{s}-1}$$

where Q'_j is homogeneous of degree j, and such that

$$(Q'_j(\mathbf{u}))_{j=0}^{\tilde{s}-1} \in f(\mathbf{u})$$

for at least $2/3$ fraction of the $\mathbf{u} \in U^m$.

Obtain this list $\mathcal{L}'$ by running the VectorRMListRecover (from Lemma 5.4) for $\tilde{s}$-tuples of polynomials of degree $\leq \tilde{s}$, where the evaluation points are U^m. Then prune the resulting list $\mathcal{L}$ to ensure that for each member

$$(Q'_j(Y_1, \ldots, Y_m))_{j=0}^{\tilde{s}-1}$$

of the list $\mathcal{L}$, and for each j such that $0 \leq j \leq \tilde{s} - 1$, Q'_j is homogeneous of degree j. This pruned list is $\mathcal{L}'$.

7. For each

$$(Q'_j(\mathbf{Y}))_{j=0}^{\tilde{s}-1} \in \mathcal{L}',$$

let

$$(P_j(\mathbf{Y}))_{j=0}^{\tilde{s}-1} = (Q'_j(\mathbf{Y} - \mathbf{b}))_{j=0}^{\tilde{s}-1},$$

and add this to a new list of tuples of polynomials that we call $\mathcal{L}''$.

8. Let

$$Z = \left\{ z \in \Sigma_{m,\tilde{s}} \mid \left(\sum_{\mathsf{wt}(\mathbf{i})=j} z^{(\mathbf{i})} \mathbf{Y}^{\mathbf{i}} \right)_{j=0}^{\tilde{s}-1} \in \mathcal{L}'' \right\}.$$

9. Return Z.

We now prove Lemma 5.3.

Proof of Lemma 5.3. Suppose $Q(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ is a polynomial of degree at most d such that:

$$\Pr_{\mathbf{x} \in \mathbb{F}_q^m}[Q^{(<s)}(\mathbf{x}) \in S(\mathbf{x})] > 1 - \alpha.$$

For each line λ in $\mathbb{F}_q^m$, let $\mathcal{L}_\lambda$ be the result of univariate list-recovering $S \circ \lambda$ with error-parameter α'. Let $\mathbf{b} \in \mathbb{F}_q^m$ be the random choices of RecoverCandidates. For $\mathbf{x}, \mathbf{b} \in \mathbb{F}_q^m$, and for $\mathbf{u} \in U^m$, recall that $\lambda_{\mathbf{u}}(T)$ denotes the line $\lambda_{\mathbf{u}}(T) = \mathbf{x} + T\mathbf{b_u}$, where $\mathbf{b_u} = \mathbf{b} + \mathbf{u}$. Then for each $\mathbf{u} \in U^m$, let $B_{\mathbf{u}}$ be the event that $Q \circ \lambda_{\mathbf{u}}(T)$ is not in $\mathcal{L}_{\lambda_{\mathbf{u}}}$.

Claim 5.6. *For each fixed $\mathbf{u} \in U^m$,*

$$\Pr_{\mathbf{x}, \mathbf{b} \in \mathbb{F}_q^m}[B_{\mathbf{u}}] \leq \frac{4}{(\alpha' - \alpha)q}.$$

Proof. Note that when $\mathbf{x}, \mathbf{b}$ are uniformly random elements of $\mathbb{F}_q^m$, then $\lambda_{\mathbf{u}}(T)$ is a uniformly random line. The event that $Q \circ \lambda_{\mathbf{u}}(T)$ is not in $\mathcal{L}_{\lambda_{\mathbf{u}}}$ is a subset of the event that $Q^{(<s)}(\mathbf{x}) \notin S(\mathbf{x})$ for more than α' fraction of points $\mathbf{x}$ on the line $\lambda_{\mathbf{u}}(T)$. The claim then follows from a standard application of Chebyshev's inequality, using the fact that the points on a uniformly random line are pairwise independent.

More precisely,

$$\Pr_{\mathbf{x}, \mathbf{b} \in \mathbb{F}_q^m}[B_{\mathbf{u}}] \leq \Pr_{\mathbf{x}, \mathbf{b} \in \mathbb{F}_q^m}\left[\sum_{t \in \mathbb{F}_q} \mathbf{1}\left\{Q^{(<s)}(\lambda_{\mathbf{u}}(t) \notin S(\lambda_{\mathbf{u}}(t))\right\} > \alpha'q\right]$$

$$=: \Pr_Y\left[\sum_{t \in \mathbb{F}_q} Y_t > \alpha'q\right]$$

$$= \Pr_Y\left[\sum_{t \in \mathbb{F}_q}(Y_t - \mathbb{E}Y_t) > (\alpha' - \alpha)q\right],$$

where the Y_t are pairwise independent $\{0,1\}$-valued random variables with $\mathbb{E}Y_t \leq \alpha$. Then by Chebyshev's inequality, this last quantity is at most

$$\frac{\sum_{t \in \mathbb{F}_q} \mathbb{E}(Y_t - \mathbb{E}Y_t)^2}{(\alpha' - \alpha)^2 q^2} \leq \frac{4}{(\alpha' - \alpha)^2 q}.$$

$\square$

Claim 5.7.

$$\Pr_{\mathbf{x}, \mathbf{b}_1, \ldots, \mathbf{b}_m}\left[\sum_{\mathbf{u} \in U^m} \mathbf{1}_{B_{\mathbf{u}}} > \frac{|U|^m}{3}\right] < \frac{12}{(\alpha' - \alpha)q}.$$

Proof. The proof is immediate from the previous claim and Markov's inequality. $\square$

Thus we conclude that with probability at least $1 - \frac{12}{(\alpha' - \alpha)q}$, when $\mathbf{x}$ is a uniformly random element of $\mathbb{F}_q^m$, for at least $1/3$ of the $\mathbf{u} \in U^m$, we have that

$$Q \circ \lambda_{\mathbf{u}}(T) \in \mathcal{L}_{\lambda_{\mathbf{u}}}.$$

We assume that this happens, and let $G \subseteq U^m$ be this set of $\mathbf{u}$.

Recall that $\mathcal{L}_{\lambda_{\mathbf{u}}}$ is a list of size L. Consider the function

$$f : U^m \to \binom{\mathbb{F}_q^{\tilde{s}}}{L},$$

where for each $\mathbf{u} \in U^m$,

$$f(\mathbf{u}) = \{P^{(<\tilde{s})}(0) \mid P(T) \in \mathcal{L}_{\lambda_{\mathbf{u}}}\}.$$

Fix any $\mathbf{u} \in G$. Then since

$$Q \circ \lambda_{\mathbf{u}}(T) \in \mathcal{L}_{\lambda_{\mathbf{u}}},$$

it holds that

$$(Q \circ \lambda_{\mathbf{u}})^{(<\tilde{s})}(0) \in f(\mathbf{u}).$$

Now observe that by Claim 5.5, we have

$$H_{\mathbf{x}}(\mathbf{b_u}) = (Q \circ \lambda_{\mathbf{u}})^{(<\tilde{s})}(0),$$

and thus

$$H_{\mathbf{x}}(\mathbf{b} + \mathbf{u}) = H_{\mathbf{x}}(\mathbf{b_u}) \in f(\mathbf{u}).$$

Since this happens for each $\mathbf{u} \in G$, we have that this happens for at least $1/3$ fraction of $\mathbf{u} \in U^m$.

Now by our assumption that $|U| \geq 100m^2 L\tilde{s}$, Lemma 5.4 implies[8] that the algorithm VectorRMListRecover on input f and error parameter $2/3$ will include $H_{\mathbf{x}}(\mathbf{b} + \mathbf{Y})$ in $\mathcal{L}'$ (here we also use the fact that $H_{\mathbf{x},j}$ is an m-variate polynomial of degree at most j).

In this event, $\mathcal{L}''$ will contain

$$H_{\mathbf{x}}(\mathbf{Y})$$

and then it follows that in Step 8 of RecoverCandidates, the list Z will contain $Q^{(<\tilde{s})}(\mathbf{x})$.

$\square$

5.3   The Oracle Machine M

Our final local list recovery algorithm will output a short list of oracle machines, each of which is defined by a piece of advice. In this case, the advice will be a point $\mathbf{a} \in \mathbb{F}_q^m$, and $z \in \Sigma_{m,s^*}$, which is meant to be a guess for $Q^{(<s^*)}(\mathbf{a})$. Given this advice, the oracle machine works as follows: on input $\mathbf{x}$, with corresponding input list $Y = S(\mathbf{x})$, it will run the univariate list-recovery algorithm on the line $\lambda(T) = \mathbf{x} + T(\mathbf{a} - \mathbf{x})$ through $\mathbf{x}$ and $\mathbf{a}$ to obtain a list $\mathcal{L}$ of univariate polynomials $P(T)$. We will show that with high probability (assuming the advice is good), there will be a unique polynomial $P(T)$ in $\mathcal{L}$ so that both $P^{(<s^*)}(1)$ is consistent with z, and $P^{(<s^*)}(0)$ is consistent with some element of Y. Then the oracle machine will output the symbol in Y that $P^{(<s^*)}(0)$ agrees with.

The key later will be that the advice z will not vary over all possibilities in Σ_{m,s^*}; this would result in too long a list. Rather, we will use RecoverCandidates in order to generate this advice.

---

[8]WARNING: we invoke the algorithm and statement of Lemma 5.4 with input list size equal to L; iė; we take $\ell = L$ (and $K = 100m^2$) in the statement when we invoke it.

Formally, we will prove the following lemma about our oracle machine, which we define below.

Lemma 5.8. *Let q be a prime, let $\delta > 0$ and let s, d, m be nonnegative integers such that $d = (1 - \delta)sq$. Let α, α' be parameters such that $0 < \alpha < \alpha'$.*

Let $L = L(d, q, s, \ell, \alpha')$ be the list size for list recovering univariate multiplicity codes $\mathsf{MULT}_{q,s}^{(1)}(d)$ of degree d and multiplicity parameter s with input list size ℓ and error tolerance α', and T be the corresponding running time for list recovering univariate multiplicity codes of the same parameters.

Let $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$. Let $s^* > 0$ be a parameter. Suppose that $q > C \cdot s^* \cdot L \cdot m^2$, for some absolute constant C, and suppose that $Q(X_1, \ldots, X_m) \in \mathbb{F}_q[X_1, \ldots, X_m]$ is a polynomial of degree at most d such that:

$$\Pr_{\mathbf{x} \in \mathbb{F}_q^m}[Q^{(<s)}(\mathbf{x}) \in S(\mathbf{x})] > 1 - \alpha.$$

Let

$$\varepsilon = \alpha + \frac{\ell s}{q} + \frac{4}{(\alpha' - \alpha)q} + \frac{sL}{s^*}.$$

There is an algorithm $M^S[\mathbf{a}, z](\mathbf{x})$ which on input $x \in \mathbb{F}_q^m$, given as advice a point $\mathbf{a} \in \mathbb{F}_q^m$, and $z \in \Sigma_{m,s^*}$, and given oracle access to S, makes at most q queries to S, runs in time at most $\mathsf{poly}\left(T, q, (s \cdot s^*)^m\right)$, and outputs an element of $\Sigma_{m,s} \cup \{\bot\}$ such that if $\mathbf{x}, \mathbf{a} \in \mathbb{F}_q^m$ are chosen uniformly at random, then:

$$\Pr_{\mathbf{a}, \mathbf{x} \in \mathbb{F}_q^m}\left[M^S[\mathbf{a}, Q^{(<s^*)}(\mathbf{a})](\mathbf{x}) = Q^{(<s)}(\mathbf{x})\right] \geq 1 - \varepsilon.$$

In particular, if we view $M^S[\mathbf{a}, Q^{(<s^*)}(\mathbf{a})]$ as a function from $\mathbb{F}_q^m \to \Sigma_{m,s} \cup \{\bot\}$, then with probability at least $3/4$ over the choice of $\mathbf{a} \in \mathbb{F}_q^m$,

$$\mathrm{dist}\left(M^S[\mathbf{a}, Q^{(<s^*)}(\mathbf{a})], Q^{(<s)}\right) \leq 4\varepsilon.$$

We will first decribe the algorithm and then show that it satisfies the required properties.

Oracle machine M.

- Oracle access to $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$.

- **INPUT:** $\mathbf{x} \in \mathbb{F}_q^m$.

- **ADVICE:** Point $\mathbf{a} \in \mathbb{F}_q^m$, and $z \in \Sigma_{m,s^*}$.

1. Let $Y = S(\mathbf{x})$.

2. Set $\mathbf{b}_* = \mathbf{a} - \mathbf{x}$.

---

3. Let $\lambda_{\mathbf{b}_*}$ be the line $\lambda_{\mathbf{b}_*}(T) = \mathbf{x} + T\mathbf{b}_*$.

4. Consider the restriction $S_{\mathbf{b}_*} : \mathbb{F}_q \to \binom{\Sigma_{1,s}}{\ell}$ of S to the line $\lambda_{\mathbf{b}_*}$, and list recover this with error-tolerance α' for degree d polynomials, and obtain the list $\mathcal{L}_{\lambda_{\mathbf{b}_*}} \subseteq \mathbb{F}_q[T]$.

5. If there exists exactly one $P(T) \in \mathcal{L}_{\lambda_{\mathbf{b}_*}}$ such that $P^{(<s^*)}(1) = z|_{\mathbf{b}_*}$, then set $P_{\mathbf{b}_*}(T)$ to equal that $P(T)$, otherwise output $\perp$ and exit.

6. If there exists exactly one $y \in Y$ for which $y|_{\mathbf{b}_*} = P_{\mathbf{b}_*}^{(<s)}(0)$, then output that y.

7. Otherwise output $\perp$.

We will now analyze the above algorithm and show that is satisfies the required properties.

Proof of Lemma 5.8. By the description of the oracle machine, it is clear that it makes at most q queries. Moreover its running time is at most $\mathsf{poly}\,(T, q, (s \cdot s^*)^m)$, since in addition to running the univariate list recovery algorithm, it needs to do some field calculations such as computing the restriction of S to a line, as well as computing for each $P(T) \in \mathcal{L}_*$, $P^{(<s^*)}(1)$ and $z|_{\mathbf{b}_*}$ and comparing the two. This takes time at most $\mathsf{poly}\,(q, (s \cdot s^*)^m)$.

It remains to show that when $\mathbf{a}$ and $\mathbf{x}$ are chosen uniformly at random from $\mathbb{F}_q^m$, then

$$\Pr_{\mathbf{a},\mathbf{x}\in\mathbb{F}_q^m} \left[M^S[\mathbf{a}, Q^{(<s^*)}(\mathbf{a})](\mathbf{x}) = Q^{(<s)}(\mathbf{x}) \right] \geq 1 - \varepsilon.$$

Claim 5.9. *Let $y_0 = Q^{(<s)}(\mathbf{x})$. With probability at least $1 - \alpha$ over the random choice of $x \in \mathbb{F}_q^m$, $y_0 \in Y$.*

Proof. Recalling that $Y = S(\mathbf{x})$, the proof is immediate since it is given to us that

$$\Pr_{\mathbf{x}\in\mathbb{F}_q^m}[Q^{(<s)}(\mathbf{x}) \in S(\mathbf{x})] > 1 - \alpha.$$

$\square$

Claim 5.10. *Let $y_0 = Q^{(<s)}(\mathbf{x})$. For any $y \in Y$ with $y \neq y_0$, with probability at least $1 - \frac{s}{q}$ over the random choice of $\mathbf{a} \in \mathbb{F}_q^m$, we have that*

$$y|_{\mathbf{b}_*} \neq y_0|_{\mathbf{b}_*}.$$

Proof. Recall that by definition, for an element $z \in \Sigma_{m,s}$, and a direction $\mathbf{b} \in \mathbb{F}_q^m$, $z|_{\mathbf{b}}$ is to equal $h \in \Sigma_{1,s}$, where:

$$h^{(j)} = \sum_{\mathsf{wt}(\mathbf{j})=j} z^{(\mathbf{j})} \mathbf{b}^{\mathbf{j}}$$

for each j such that $0 \leq j < s$. Note that $h^{(j)}$ can be viewed as a polynomial of degree at most j evaluated at $\mathbf{b}$, where the coefficients of the polynomial depend only on z.

Since $y \neq y_0$, the corresponding tuples of polynomials (each of degree at most s) will differ in at least one coordinate. Observe also that for any fixed choice of $\mathbf{x}$, the randomness of $\mathbf{a}$ implies that

39

$\mathbf{b}_*$ is a uniformly random element of $\mathbb{F}_q^m$. This in the coordinate where the tuples of polynomials differ, the evaluations at $\mathbf{b}_*$ will be distinct with probability at least $1 - \frac{s}{q}$ by the Schwartz-Zippel Lemma.

Thus with probability at least $1 - \frac{s}{q}$ over the random choice of $\mathbf{a} \in \mathbb{F}_q^m$, we have that

$$y|_{\mathbf{b}_*} \neq y_0|_{\mathbf{b}_*}.$$

□

Claim 5.11. *Let* $y_0 = Q^{(<s)}(\mathbf{x})$. *For any* $x \in \mathbb{F}_q^m$ *such that* $y_0 \in Y$, *with probability at least* $1 - \frac{\ell s}{q}$ *over the random choice of* $\mathbf{a} \in \mathbb{F}_q^m$, $y_0|_{\mathbf{b}*}$ *is unique element* y *of* Y *for which* $y|_{\mathbf{b}*} = Q \circ \lambda_{\mathbf{b}_*}^{(<s)}(0)$.

Proof. Clearly, by definition, $y_0|_{\mathbf{b}*} = Q \circ \lambda_{\mathbf{b}_*}^{(<s)}(0)$. Also, taking a union bound over all ℓ elements of Y, by Claim 5.10, $y_0|_{\mathbf{b}*} \neq y|_{\mathbf{b}*}$ for all other $y \in Y$ with probability at least $1 - \frac{\ell s}{q}$. □

Claim 5.9 and Claim 5.11 together imply that with probability at least $1 - \left(\alpha + \frac{\ell s}{q}\right)$ over the random choice of $\mathbf{a}$ and $\mathbf{x} \in \mathbb{F}_q^m$, $Q^{(<s)}(\mathbf{x})|_{\mathbf{b}_*}$ is the unique element y of Y for which $y|_{\mathbf{b}*} = Q \circ \lambda_{\mathbf{b}_*}^{(<s)}(0)$.

We will now show that with probability at least $1 - \left(\frac{4}{(\alpha'-\alpha)q} + \frac{sL}{s^*}\right)$ over the random choice of $\mathbf{a}$ and $\mathbf{x} \in \mathbb{F}_q^m$, $P_{\mathbf{b}_*}(T) = Q \circ \lambda_{\mathbf{b}_*}(T)$. Once we will have this, then it will immediately follow that the algorithm will output $Q^{(<s)}(\mathbf{x})$ with probability at least $1 - \left(\frac{4}{(\alpha'-\alpha)q} + \frac{sL}{s^*} + \frac{4}{(\alpha'-\alpha)q} + \frac{sL}{s^*}\right)$ over the random choice of $\mathbf{a}$ and $\mathbf{x} \in \mathbb{F}_q^m$

For each line λ in $\mathbb{F}_q^m$, let $\mathcal{L}_\lambda$ be the result of list-recovering $S \circ \lambda$ with error-parameter α'. For points $\mathbf{x}$ and $\mathbf{a}$ picked uniformly at random from $\mathbb{F}_q^m$, let $\mathbf{b}_* = \mathbf{a} - \mathbf{x}$, and let $\lambda_{\mathbf{b}_*}$ be the line $\lambda_{\mathbf{b}_*}(T) = \mathbf{x} + T\mathbf{b}_*$.

Let $B_{\lambda_{\mathbf{b}_*}}$ denote the event that $\mathcal{L}_{\lambda_{\mathbf{b}_*}}$ does not contain $Q \circ \lambda_{\mathbf{b}_*}(T)$. Let $C_{\lambda_{\mathbf{b}_*},\mathbf{a}}$ denote the event that there exist $P(T) \in \mathcal{L}_{\lambda_{\mathbf{b}_*}}$ with $P(T) \neq Q \circ \lambda_{\mathbf{b}_*}(T)$, but $P^{(<s^*)}(0) = (Q \circ \lambda_{\mathbf{b}_*})^{(<s^*)}(0)$. Thus $B_{\lambda_{\mathbf{b}_*}}$ is the event that there are too many errors on $\lambda_{\mathbf{b}_*}$. $C_{\lambda_{\mathbf{b}_*},\mathbf{a}}$ is the event that $\mathbf{a}$ is not a disambiguating point.

Claim 5.12.
$$\Pr[B_{\lambda_{\mathbf{b}_*}}] = \frac{4}{(\alpha' - \alpha)q}.$$

Proof. The proof is identical to that of Claim 5.6, and it follows from a standard application of Chebyshev's inequality, using the fact that the points on a uniformly random line are pairwise independent. □

Claim 5.13.
$$\Pr[C_{\lambda_{\mathbf{b}_*},\mathbf{a}}] < \frac{sL}{s^*}.$$

40

Proof. Because of the way $\mathbf{x}$, $\mathbf{a}$ and the line $\lambda_{\mathbf{b}_*}$ are sampled, equivalently one could let $\mathbf{x}$ be picked uniformly at random from $\mathbb{F}_q^m$, $\lambda_{\mathbf{b}_*}$ be a uniformly random line through $\mathbf{x}$ and $\mathbf{a}$ be a uniformly random point on $\lambda_{\mathbf{b}_*}$.

Now fix any polynomial $P(T) \in \mathcal{L}_{\lambda_{\mathbf{b}_*}}$ with $P(T) \neq Q \circ \lambda_{\mathbf{b}_*}(T)$. We want to bound the probability that $P^{(<s^*)}(\alpha) = (Q \circ \lambda_{\mathbf{b}_*})^{(<s^*)}(\alpha)$ where α is picked uniformly at random. But P and $Q \circ \lambda_{\mathbf{b}_*}$ are fixed distinct polynomials of degree at most sq. Thus the probability that they agree with multiplicity s^* on a random point of $\mathbb{F}_q$ is at most $\frac{sq}{s^*q} = \frac{s}{s^*}$.

The result follows from a union bound over all $P(T) \in \mathcal{L}_{\lambda_{\mathbf{b}_*}}$. $\qquad\square$

Claim 5.12 and Claim 5.13 together imply that with probability at least $1 - \left(\frac{4}{(\alpha'-\alpha)q} + \frac{sL}{s^*}\right)$ over the random choice of $\mathbf{a}$ and $\mathbf{x} \in \mathbb{F}_q^m$, neither $B_{\lambda_{\mathbf{b}_*}}$ nor $C_{\lambda_{\mathbf{b}_*},\mathbf{a}}$ occurs, and hence $P_{\mathbf{b}_*}(T) = Q \circ \lambda_{\mathbf{b}_*}(T)$.

Thus the result follows.

$\qquad\square$

5.4   Main local list-recovery algorithm

Together, Lemmas 5.3 and 5.8 inspire a local-list-recovery algorithm for multivariate multiplicity codes. The idea is that RecoverCandidates will first obtain a list of possibilities, Z, for $Q^{(<s^*)}(\mathbf{a})$. Then for each possibility $z \in Z$, we will create an oracle machine as in Lemma 5.8 which guesses $Q^{(<s^*)}(\mathbf{a}) = z$. Unfortunately, this will still have some amount of error; that is, there will be some small fraction of $\mathbf{x} \in \mathbb{F}_q^m$ so that the approach above will not be correct on $\mathbf{x}$. To get around this, we will wrap the whole thing in the local (unique) correction algorithm for multiplicity codes from [KSY14].

Theorem 5.14 ([KSY14], Theorem 3.6). *Let C be multiplicity code $\mathsf{MULT}_{q,s}^{(m)}(d)$. Let $\delta = 1 - \frac{d}{sq}$. Suppose that $q \geq \max\{10m, \frac{d+6s}{s}, 12(s+1)\}$. Then C is locally correctable from $\frac{\delta}{10}$-fraction of errors with $(O(s)^m \cdot q)$ queries. Moreover, the local corrector $\mathsf{SelfCorrect}^c(\mathbf{x})$, with query access to a codeword $c \in C$ running on a position $\mathbf{x} \in \mathbb{F}_q^m$, can be[9] made to run in time $O(s)^m \cdot q^{O(1)}$.*

With the self-correction algorithm for multiplicity codes in hand, we define our local-list-recovery algorithm as follows.

Algorithm LocalListRecoverMULT.

- Oracle access to $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$.

---

[9]This claim about the running time in [KSY14] was only proved for fields of small characteristic. There, in the discussion about "Solving the Noisy System" in Section 4.3, it was shown that the running time can be made $\mathsf{poly}(O(s)^m \cdot q)$ provided one could efficiently decode Reed-Muller codes over certain product sets in $\mathbb{F}_q$, and remarked that this was known over fields of small characteristic. Recently [KK17] showed that this Reed-Muller decoding problem could be solved over all fields. This justifies the running time claim over all fields.

1. Pick $\mathbf{a} \in \mathbb{F}_q^m$ uniformly at random.

2. Set $s^* = \frac{160 \cdot L \cdot s}{\delta}$.

3. Let Z be the output of $\mathsf{RecoverCandidates}^S(\mathbf{a}, s^*)$.

4. for $z \in Z$, define $\mathcal{A}_z$ by:

   - **INPUT: $\mathbf{x} \in \mathbb{F}_q^m$**

   (a) Let M denote the oracle machine $M^S[\mathbf{a}, z]$
   (b) Return $\mathsf{SelfCorrect}^M(\mathbf{x})$

5. Return $\mathcal{L} = \{\mathcal{A}_z \; : \; z \in Z\}$.

The following lemma shows that this algorithm works, assuming a list-recovery algorithm for univariate multiplicity codes. In the proof of Theorem 5.1, we will instantiate this with the list-recovery algorithm given in Section 4.

Lemma 5.15. *There is some constant $C > 0$ so that the following holds. Let q be a prime, let $\delta > 0$ and let s, d, m be nonnegative integers such that $d = (1 - \delta)sq$. Let α, α' be parameters such that $0 < \alpha < \alpha' < 1$.*

Let $L = L(d, q, s, \ell, \alpha')$ be the list size for list recovering univariate multiplicity codes $\mathsf{MULT}_{q,s}^{(1)}(d)$ of degree d and multiplicity parameter s with input list size ℓ and error tolerance α', and T be the corresponding running time for list recovering univariate multiplicity codes of the same parameters.

Let $S : \mathbb{F}_q^m \to \binom{\Sigma_{m,s}}{\ell}$. Suppose that

$$s^* \geq \frac{160 \cdot L \cdot s}{\delta}$$

and that

$$q \geq \max\left\{\frac{160\ell s}{\delta}, \frac{640}{(\alpha' - \alpha) \cdot \delta}, Cs^* Lm^2, \frac{20 \cdot Cm}{\alpha' - \alpha}, 10m, \frac{d + 6}{s}, 12(s + 1)\right\}$$

and that

$$\alpha \leq \frac{\delta}{160}.$$

Then for all $Q(X_1, \ldots, X_m) \in \mathbb{F}_q^m[X_1, \ldots, X_m]$ with degree at most d and so that

$$\Pr_{\mathbf{x} \in \mathbb{F}_q^m}[Q^{(<s)}(\mathbf{x}) \in S(\mathbf{x})] > 1 - \alpha,$$

with probability at least $2/3$ over the algorithm $\mathsf{LocalListRecoverMULT}$, the following holds. For all $\mathbf{x} \in \mathbb{F}_q^m$, there exists an oracle machine $\mathcal{A}_z \in \mathcal{L}$ so that

$$\Pr[\mathcal{A}_z(\mathbf{x}) = Q(\mathbf{x})] \geq 2/3.$$

Moreover, the output list $\mathcal{L}$ has size $|\mathcal{L}| = O(L)$; and $\mathsf{LocalListRecoverMULT}$ makes $\mathsf{poly}(q, (Ls^*m)^m)$ queries to S, and each $\mathcal{A}_z$ makes $O(s)^m \cdot q^2$ queries to S. Finally, the algorithm $\mathsf{LocalListRecoverMULT}$ runs in time $\mathsf{poly}(T, q, (Ls^*m)^m)$ and each $\mathcal{A}_z$ runs in time $O(s)^m \cdot \mathsf{poly}(q, T, (s \cdot s^*)^m)$.

Proof. Fix a polynomial $Q \in \mathbb{F}_q[X_1, \ldots, X_m]$ of degree at most d, so that $\mathrm{dist}(Q, S) \le \alpha$. We first establish the correctness of the algorithm LocalListRecoverMULT given above.

By Lemma 5.3, with probability at least $1 - \frac{20m}{(\alpha'-\alpha)q} \ge 1 - \frac{1}{C}$ over the randomness of both RecoverCandidates and $\mathbf{a}$, RecoverCandidates$^S(\mathbf{a}, s^*)$ returns a list Z of size at most $O(L)$ so that $Q^{(<s^*)}(\mathbf{a}) \in Z$. Let G_1 be the set of $\mathbf{a} \in \mathbb{F}_q^m$ so that $\mathrm{Pr}_{\mathsf{RecoverCandidates}}[Q^{(<s^*)}(\mathbf{x}) \in Z] \ge 1 - \frac{1}{\sqrt{C}}$. By Markov's inequality along with the conclusion of Lemma 5.3 above, G_1 has density at least $1 - 1/\sqrt{C}$. Now let G_2 be the set of $\mathbf{a} \in \mathbb{F}_q^m$ so that $\mathrm{dist}(M^S[\mathbf{a}, Q^{(<s^*)}(\mathbf{a}], Q^{(<s)}) \le 4\varepsilon$. By Lemma 5.8, G_2 has density at least $3/4$. Thus by the union bound, with probability at least $3/4 - 1/\sqrt{C}$ over the choice of $\mathbf{a}$, both events hold, and so with probability at least $3/4 - 2/\sqrt{C}$ over the choice of $\mathbf{a}$ and the randomness of RecoverCandidates, there is some $z \in Z$ so that

$$\mathrm{dist}(M^S[\mathbf{a}, z], Q^{(<s)}) \le 4\varepsilon \tag{4}$$

for any

$$\varepsilon \le \alpha + \frac{\ell s}{q} + \frac{4}{(\alpha'-\alpha)q} + \frac{sL}{s^*}.$$

By choosing $C \ge 24^2$ (as well as large enough so that Lemma 5.3 and 5.8 hold), we can ensure that (4) occurs with probability at least $2/3$. Suppose that this happens, and (4) does occur. Observe that our parameter choices above are made precisely so that

$$\frac{\delta}{40} \ge \alpha + \frac{\ell s}{q} + \frac{4}{(\alpha'-\alpha)q} + \frac{sL}{s^*}.$$

Thus, we may take $\varepsilon = \delta/40$ in the above, and conclude that in the favorable case of (4), we have

$$\mathrm{dist}(M^S[\mathbf{a}, z], Q^{(<s)}) \le \frac{\delta}{10}.$$

We may then apply Theorem 5.14 to the oracle machine $M = M^S(\mathbf{a}, z)$ in the algorithm above, and conclude that $\mathsf{SelfCorrect}^M(\mathbf{x})$ is a local-self-corrector for $\mathsf{MULT}_{q,s}^{(m)}(d)$. In particular, for all $\mathbf{x} \in \mathbb{F}_q^m$, with probability at least $2/3$, $\mathcal{A}_z(\mathbf{x}) = Q^{(<s)}(\mathbf{x})$, as desired.

Now that we have established that the algorithm is correct, we quickly work out the list size, query complexity, and runtime. The list size is clearly $O(L)$, because this is the list size returned by RecoverCandidates. For the query complexity, the algorithm LocalListRecoverMULT has the same query complexity as RecoverCandidates, while each $\mathcal{A}_z$ has query complexity which is the product of the query complexities of the oracle machines $M^S[\mathbf{a}, z]$ (which is q) and SelfCorrect (which is $O(s)^m \cdot q$), and together these give the reported values. The runtime calculation is similar. $\square$

Finally, we may choose parameters and use Theorem 4.4 to prove Theorems 5.1 and 5.2.

Proof of Theorem 5.1. The proof proceeds by setting parameters in Lemma 5.15. We will let $C = \mathsf{MULT}_{q,s}^{(m)}(d)$, where m is the parameter from the theorem statement. We will choose q, s, d below. Let $\delta = \varepsilon/(2m)$; we will verify below that δ is a bound on the distance of C.

Choose $s = \frac{16 \cdot \ell}{\delta^2}$, and $\alpha \le \frac{\delta^2}{160 \cdot \ell}$ as in the theorem statement. We will choose $\alpha' = 2\alpha$, so $\alpha' - \alpha = \alpha$. We note that these choices ensure that $\ell \le \frac{\delta^2 s}{16}$ and that $\alpha' < 1/2s$, both of which are required for

Theorem 4.4 to hold (when called with α' as the error parameter), as well as $\alpha < \frac{\delta}{160}$, as required by Lemma 5.15.

Now, with these choices Theorem 4.4 says that $\mathsf{MULT}_{q,s}^{(1)}(d)$ is (α', ℓ, L')-list-recoverable with

$$L' = \ell^{O(s \log(s))} \cdot s^{O(1)} = \ell^{O(\ell \log(\ell/\delta)/\delta^2)} \cdot \left(\frac{1}{\delta}\right)^{O(1)} = \ell^{O(\ell \cdot m^2 \log(m\ell/\varepsilon)/\varepsilon^2)} \cdot \left(\frac{m}{\varepsilon}\right)^{O(1)},$$

using our choice of s and δ. Since the list size in Lemma 5.15 grows by at most a constant factor, this establishes our choice of L in the theorem statement.

We will next choose q. We need q to be large enough so that Lemma 5.15 holds. It can be checked that of all of the requirements on q given in Lemma 5.15, the binding one is that $q = \Omega(s^* L m^2)$, where we chose $s^* = \Theta(Ls/\delta)$. We shall choose q safely larger than this, choosing a prime q so that

$$q := \Theta\left(\left(\frac{Lms^*}{\delta}\right)^m\right) = (Lm\ell/\varepsilon)^{O(m)}.$$

The reason for this choice is that this is the largest we may take q so that the query complexity expression

$$\mathsf{poly}(q \cdot (Ls^*m)^m)$$

from Lemma 5.15 does not substantially grow.

Now that we have chosen s and q, we will finally choose

$$d = \left(1 - \frac{\varepsilon}{2m}\right) sq$$

so that the distance of C is

$$\delta = 1 - \frac{d}{sq} = \frac{\varepsilon}{2m}$$

as claimed.

With this choice the query complexity given in Lemma 5.15 is

$$t = O(s)^m \cdot q^2 + q^{O(1)} \cdot \left(\frac{Ls^*m}{\delta}\right)^{O(m)},$$

which with our choices of s, δ and q is

$$t = \left(\frac{Lm\ell}{\varepsilon}\right)^{O(m)}$$

as claimed.

We now verify the rate. As per Claim 2.6, the rate of C is at least

$$\begin{aligned}
R &\geq \left(1 - \frac{m^2}{s}\right)(1 - \delta)^m \\
&= \left(1 - \frac{\varepsilon^2}{16\ell}\right)\left(1 - \frac{\varepsilon}{2m}\right)^m \\
&\geq \left(1 - \frac{\varepsilon^2}{16\ell}\right)\left(1 - \frac{2\varepsilon}{3}\right) \\
&\geq 1 - \varepsilon,
\end{aligned}$$

44

where the last two lines hold for sufficiently small ε. We first note that our choice of $\alpha \leq \frac{\delta^2}{160\ell}$ satisfies $\alpha \leq \frac{\delta}{160}$, which was required in Lemma 5.15.

Finally, we note that the length of the code C is

$$N = q^m = \left(\frac{Lm\ell}{\varepsilon}\right)^{O(m^2)}$$

and that the alphabet size is similarly

$$|\Sigma| = q^{s^m} = \left(\frac{Lm\ell}{\varepsilon}\right)^{O(s^m)},$$

which results in the expression given in the theorem statement. Finally, the running time for the list-recovery algorithm guaranteed by Lemma 5.15 is dominated by the $(Ls^*m)^m$ term, which is $\mathsf{poly}(t)$. $\qquad\square$

Proof of Theorem 5.2. Again, we set parameters in Lemma 5.15. Let $\delta, \varepsilon, \tau, \ell$ be as in the statement of Theorem 5.2. We will choose $C = \mathsf{MULT}_{q,s}^{(m)}(d)$, and we set parameters below. First, we choose

$$s = \max\left\{\frac{16\ell}{\delta^2}, \frac{c^2}{\tau^2\varepsilon}\right\},$$

where c is some universal constant that will be chosen below. We will choose

$$m = \frac{c}{\tau},$$

and d so that $d = (1-\delta)sq$, ensuring that the relative distance of the code is at least δ. Now we choose

$$\alpha' \leq \min\left\{\frac{\delta^2}{160\ell}, \frac{\tau^2\varepsilon}{2c^2}\right\},$$

and $\alpha = \alpha'/2$. which ensures that $\alpha \leq \delta/160$ (as is required for Lemma 5.15) and that $\alpha' \leq 1/2s$, which is required for Theorem 4.4. We also have $(\alpha' - \alpha) = \Omega_{\delta,\varepsilon,\ell,\tau}(1)$.

Notice that all of the requirements on the size of q in Lemma 5.15 simply require $q = \Omega_{\ell,\varepsilon,\delta,\tau}(1)$, so we choose any prime q sufficiently large, and let $N = q^m$ is be the length of the multiplicity code.

By Claim 2.6, C has rate at least

$$\left(1 - \frac{m^2}{s}\right)\left(\frac{d}{sq}\right)^m \geq (1-\varepsilon)(1-\delta)^m = (1-\varepsilon)(1-\delta)^{c/\tau},$$

which is what was claimed.

Then by Theorem 4.4, the univariate multiplicity code $\mathsf{MULT}_{q,s}^{(1)}(d)$ is (α, ℓ, L)-list-recoverable with

$$L = \ell^{O(s\log(s))}s^{O(1)} = O_{\ell,\delta,\varepsilon,\tau}(1)$$

in time $\mathsf{poly}_{\ell,\delta,\varepsilon,\ell}(q)$.

Now we choose $s^* = 160Ls/\delta = O_{\ell,\delta,\varepsilon,\tau}(1)$, and Lemma 5.15 concludes that C is (t, α, ℓ, L')-list-recoverable for $L' = O(L) = O_{\ell,\delta,\varepsilon,\tau}(1)$ and for

$$t = q^c \cdot (Ls^*m)^{O}(m) = O_{\ell,\delta,\varepsilon,\tau}(q^c)$$

for some constant c. (This defines the constant c). Now, since $N = q^m = q^{c/\tau}$, we have $t = O_{\ell,\delta,\varepsilon,\tau}(N^\tau)$, as desired. Finally, Lemma 5.15 further implies that the running time of the local list-recovery algorithm is $\mathsf{poly}(t)$, where the exponent in the polynomial does not depend on $\ell, \delta, \varepsilon$, or τ.

$\square$

6 Capacity-achieving codes over constant-sized alphabets

Theorems 5.1 and 5.2 show that high-rate multivariate multiplicity codes are efficiently locally list-recoverable. However, the alphabet sizes for both of these constructions are large, and they only tolerate a small amount of error. Fortunately, via standard techniques, we can both boost the error tolerance and improve the alphabet size without substantially impacting the locality or list size. We will prove the following theorems, based on Theorems 5.1 and 5.2 respectively.

First we give a statement with sub-polynomial query complexity and list size.

Theorem 6.1. Let $R > 0$. Let $\varepsilon > 0$ be sufficiently small, and let $m, \ell > 0$ be integers. Suppose that ε, ℓ are constants, independent of m, and that $R \in (\varepsilon, 1 - 2\varepsilon)$. There is a code $C \subseteq \Sigma^N$ with rate R that is $(t, 1 - R - \varepsilon, \ell, L)$-list-recoverable for

$$L = m^{O_{\ell,\varepsilon}(m^2)}$$

$$t = m^{O_{\ell,\varepsilon}(m^3)}$$

$$N = m^{O_{\ell,\varepsilon}(m^4)}$$

$$|\Sigma| = O_{\ell,\varepsilon}(1)$$

which can be locally list-recovered in time $\mathsf{poly}(t)$. Moreover, C has a deterministic encoding algorithm which runs in time $\mathsf{poly}(N)$.

In particular, solving for $m \approx \left(\frac{\log(N)}{\log\log(N)}\right)^{1/4}$, we have that

$$L = \exp\left(\sqrt{\log(N)\log\log(N)}\right) \qquad t = \exp\left(\log^{3/4}(N) \cdot (\log\log(N))^{1/4}\right).$$

Next we give a statement with polynomial query complexity but constant list size.

Theorem 6.2. Let $R > 0$. Let $\varepsilon, \tau > 0$ be sufficiently small, and let $\ell > 0$ be an integer. Suppose that ε, τ, ℓ are constants, and that $R \in (\varepsilon, 1 - 2\varepsilon)$. Then for infinitely many N, there is a code $C \subseteq \Sigma^N$ of rate R that is $(t, 1 - R - \varepsilon, \ell, L)$-list-recoverable for

$$L = O_{\ell,\varepsilon,\tau}(1)$$

$$t = O_{\ell,\varepsilon,\tau}(N^{\tau})$$

$$|\Sigma| = O_{\ell,\varepsilon,\tau}(1),$$

which can be locally list-recovered in time $\mathsf{poly}(t)$. *Moreover,* C *has a deterministic encoding algorithm that runs in time* $\mathsf{poly}(N)$.

The proof of Theorems 6.1 and 6.2 will follow from an expander-based construction [AEL95] which has been used in similar settings to reduce alphabet sizes and improve the rate/distance trade-offs (and in particular in [GR08, GKO+17, HRW17] in the context of list-recovery). We state a general transformation below.

Theorem 6.3. *Choose* $\varepsilon, \gamma, \zeta, R \in (0, 1)$. *Let* $C_1 \subseteq \Sigma_1^{n_1}$ *be a code of rate* $1 - \zeta$. *Suppose that there exists a code* $C_0 \subseteq \Sigma_0^{n_0}$ *of rate* R *which is* $(1 - R - \varepsilon, \ell, \ell_1)$*-list-recoverable in time* $T(C_0)$, *which can be deterministically constructed in time* $T_{construct}(C_0)$.

Then there exists a code $C \subseteq \Sigma^N$ *of rate* $(1 - \zeta) \cdot R$ *over an alphabet of size*

$$|\Sigma| = |\Sigma_0|^{O(1/(\varepsilon^3 \cdot \gamma))}$$

and block length

$$N = O\left(\frac{n_1 \log |\Sigma_1| \varepsilon^3 \gamma}{R \log |\Sigma_0|}\right).$$

so that C *can be deterministically constructed in time* $T_{construct}(C_0) + \mathsf{poly}(|\Sigma_1|, n_1, 1/\varepsilon, 1/\gamma)$ *and so that:*

- *If* C_1 *is* (γ, ℓ_1, L)*-list-recoverable in time* $T(C_1)$, *then* C *is* $(1 - R - 4\varepsilon, \ell, L)$*-list-recoverable in time* $O(n_1 T(C_0) + T(C_1))$.

- *If* C_1 *is* (t, γ, ℓ_1, L)*-list-recoverable in time* $T'(C_1)$, *then* C *is* $(t', 1 - R - 4\varepsilon, \ell, L)$*-locally list-recoverable in time*
$$O\left(\frac{T(C_0) + T'(C_1)}{\varepsilon^3 \gamma}\right),$$
  where
$$t' = O\left(\frac{t \log |\Sigma_1|}{R \log |\Sigma_0|}\right).$$

The proof of Theorem 6.3 is by now standard, and we include it in Appendix E for completeness. The basic idea is to concatenate C_1 with C_0, and then to scramble up and re-aggregate the symbols of the resulting concatenated code using a bipartite expander graph.

We will use Theorem 6.3 three times: once with C_0 as a random linear code and C_1 as the concatenation of two folded RS codes; and the next two times with C_0 as the code produced by the first application of Theorem 6.3 and with C_1 as a multivariate multiplicity code from Theorem 5.1 and Theorem 5.2, respectively.

For the list-recoverability of a random linear code, we use a result of [RW17].

Theorem 6.4 (Follows from Theorem 6.1 in [RW17]). *Choose $R \in (0,1)$ be constant, and let $\varepsilon, \ell > 0$. There is some $q_0 = (1+\ell)^{O(1/\varepsilon)}$ and*

$$\ell_1 = \left(\frac{q\ell}{\varepsilon}\right)^{O(\log(\ell)/\varepsilon^3)}$$

so that the following holds. Let q be a prime power, and let C_0 be a random linear code over $\mathbb{F}_q$ of rate R and length n_0. Then with high probability, C_0 is $(1 - R - \varepsilon, \ell, \ell_1)$-list-recoverable.

We will also use the following corollary of Theorem 3.1.

Corollary 6.5. *Let $\varepsilon > 0$ and $\ell \in \mathbb{N}$ be constants. Then for infinitely many values of n, there is a code $C \subseteq \Sigma^n$ of rate $1 - \varepsilon$, which is $(\varepsilon^2/16, \ell, L)$-list-recoverable in time $\mathsf{poly}(n, L)$, for $L = O_{\ell,\varepsilon}(1)$, and which has $|\Sigma| = \mathsf{poly}_{\ell,\varepsilon}(\log(n))$.*

Proof. The proof follows by concatenating two folded Reed-Solomon codes. More precisely, let $C_0 \subseteq \Sigma_0^{n_0}$ be a folded RS code of rate $1 - 2\gamma$ which is (γ, ℓ, ℓ_1)-list-recoverable for $\ell_1 = O_{\ell,\gamma}(1)$, which has alphabet size $|\Sigma_0| = \mathsf{poly}_{\ell,\gamma}(n_0)$ and is list-recoverable in time $\mathsf{poly}(n_0, \ell_1)$; this exists by Theorem 3.1. Then let $C_1 \subseteq \Sigma_1^{n_1}$ be another code of rate $1 - 2\gamma$ which is (γ, ℓ_1, L)-list-recoverable for $L = O_{\ell_1,\gamma}(1) = O_{\ell,\gamma}(1)$ and has $|\Sigma_1| = \mathsf{poly}_{\ell,\gamma}(n_1)$ and is list-recoverable in time $\mathsf{poly}(n_1, L)$; again this exists by Theorem 3.1. Since $|C_0| = n_0^{O_{\ell,\gamma}(n_0)}$ and $|\Sigma_1| = \mathsf{poly}_{\ell,\gamma}(n_1)$, there is a choice of n_1 so that $n_0 = O_{\ell,\gamma}(\log(n_1))$ so that $|C_0| \geq |\Sigma_1|$.

Now consider the code C which is the concatenation of C_1 with C_0. The length of the code is $n = n_1 \cdot n_0$. The alphabet size of C is $|\Sigma_0| = \mathsf{poly}_{\ell,\gamma}(n_0) = \mathsf{poly}_{\ell,\gamma}(\log(n))$, and the rate is $(1 - 2\gamma)^2 \geq 1 - 4\gamma$. Finally, it is not hard to see that the composition of two list-recoverable codes is again list-recoverable (see, eg, [HRW17], Lemma 7.4), and we conclude that C is (γ^2, ℓ, L)-list-recoverable in time $\mathsf{poly}(N, L)$. Setting $\varepsilon = 4\gamma$ completes the proof. $\square$

Next, we instantiate Theorem 6.3 using the codes from Corollary 6.5 as the outer code, and a random linear code as the inner code.

Corollary 6.6. *Let $R \in (0,1), \ell, \varepsilon > 0$ be constants so that $\varepsilon < R < 1 - 2\varepsilon$, and let $\alpha < 1 - R - \varepsilon$. Then there is a code $C \subset \Sigma^n$ of rate R, constructable in time $\mathsf{poly}_{\ell,\varepsilon}(n)$, which is (α, ℓ, L)-list-recoverable in time $O_{\ell,\varepsilon}(n^{O(1)}) + O(n \cdot \log(n)^{O_{\ell,\varepsilon}(1)})$ with*

$$|\Sigma| = (1+\ell)^{O(1/\varepsilon^6)} = O_{\ell,\varepsilon}(1)$$

and

$$|L| = O_{\ell,\varepsilon}(1).$$

Proof. The proof follows by applying Theorem 3.1 in Theorem 6.3. Let the outer code $C_1 \subseteq \Sigma_1^{n_1}$ be a code of rate $1 - \varepsilon/4$ which is (α, ℓ_1, L)-list-recoverable in time $\mathsf{poly}(n_1 \cdot O_{\ell,\varepsilon}(1))$ for $\alpha = \Omega(\varepsilon^2)$, and which has $|\Sigma_1| = \mathsf{poly}_{\ell,\varepsilon}(\log(n_1))$, as guaranteed by Corollary 6.5.

For the inner code, we use a random linear code, choosing $|\Sigma_0| = (1+\ell)^{O(1/\varepsilon)}$. By Theorem 6.4, for any n_0, there exists a linear code $C_0 \subseteq \Sigma_0^{n_0}$ of rate $R + \varepsilon/2$ that is $(1 - R - \varepsilon, \ell, \ell_1)$-list-recoverable, for

$$\ell_1 = \left(\frac{\ell}{\varepsilon}\right)^{O(\log(\ell)/\varepsilon^4)} = O_{\ell,\varepsilon}(1).$$

We have $|C_0| = (1+\ell)^{O(n_0/\varepsilon)}$, so there is a choice of $n_0 = O_{\ell,\varepsilon}(\log\log(n_1))$ so that $|C_0| \geq |\Sigma_1| = \mathsf{poly}_{\ell,\varepsilon}(\log(n_1))$, and we make this choice. Thus, we may use C_0 and C_1 in the construction in Theorem 6.3 to construct a code $C \subseteq \Sigma^N$ of rate $(1 - \varepsilon/4)(R + \varepsilon/2) \geq R$ that is $(1 - R - 4\varepsilon, \ell, L)$-list-recoverable, where $N = O_\varepsilon(n_0 \cdot n_1)$. The final alphabet size is $|\Sigma| = |\Sigma_0|^{O(1/\varepsilon^3 \alpha)} = (1+\ell)^{1/\varepsilon^6}$.

Finally, we consider how long it takes to construct and decode C. To construct C_0 we iterate over all possible generator matrices and verify their list-recovery properties. There are at most $|\Sigma_0|^{Rn_0^2} = |C_0|^{\varepsilon \log_\ell(|C_0|)}$ linear codes, and checking the list-recoverability of any one of them takes time $O(|\Sigma_0|^{\ell n_0} \cdot |C_0|^{\ell_1+1} \cdot n_0\ell)$, the time to search over all lists $S_1, \dots, S_{n_0}$ of size ℓ, and all subsets of $L + 1$ codewords and compute their distance. Since n_0 will be much larger than ℓ, ε, this is dominated by the $|\Sigma_0|^{Rn_0^2}$ term, and the time it takes to find the generator matrix of such a code is

$$T_{construct}(C_0) = O(|\Sigma_0|^{R \cdot n_0^2}) = O(|C_0|^{n_0}) = \log(n_1)^{O_{\ell,\varepsilon}(\log\log(n_1))} = \mathsf{poly}_{\ell,\varepsilon}(n_1) = \mathsf{poly}_{\ell,\varepsilon}(N).$$

Thus time to construct the whole code $C \subseteq \Sigma^N$ is also $\mathsf{poly}(N)$.

The time to perform list-recovery on the inner code C_0 by brute force is $\mathsf{poly}(|C_0|) = \mathsf{poly}_{\ell,\varepsilon}(\log(n_1))$. The time to perform list-recovery is then the time to run the list-recovery algorithm for C_1 (which is $\mathsf{poly}(N, O_{\ell,\varepsilon}(1))$), plus the time to brute-force decode C_0 n_1 times, which is $O(N \cdot \log(N)^{O_{\ell,\varepsilon}(1)})$.

Instantiating Theorem 6.3 with these choices yields the corollary. $\qquad\square$

Remark 6.7. The reason to concatenate folded RS codes with themselves to obtain the outer code C_1 above is to make the alphabet size small enough that a brute-force search over all generator matrices for the inner code is still polynomial time. If one omits this step, then the construction above still works with a quasipolynomial-time construction and a better list size. It may be possible to create a version of Corollary 6.6 which has a significantly smaller list size (close to the one guaranteed by Theorem 3.1) by using a folded RS code as C_1 and a derandomization of existing Monte-Carlo constructions of capacity-achieving list-recoverable codes as the inner code C_0. However, since a list size of $O_{\ell,\varepsilon}(1)$ is sufficient for our applications going forward, we stick with the simpler machinery.

One might try to prove Theorem 6.1 in the same way, with multivariate multiplicity codes as C_1 and a random linear code as C_0. However, in this case the alphabet size is so large that doing exhaustive search to decode C_0 would yield a super-polynomial decoding time, and concatenating the outer code with smaller versions of itself until the alphabet size is smaller will yield too large a list size. Therefore, we instead use for C_0 the code we have just created in Corollary 6.6 instead.

There is one more catch, which is that the codes from Theorem 5.1 don't meet list-decoding capacity, since they have rate $1 - \varepsilon$, but can only handle up to $O(\varepsilon^2/\ell m^2)$ fraction of errors, which in our parameter regime is sub-constant. If we applied Theorem 6.3 directly, we would need to take γ in that theorem to be sub-constant, which would result in a super-constant alphabet size. Thus,

before we apply Theorem 6.3 to reduce the alphabet size, we amplify the distance to a constant, by applying a different version of the expander-based argument stated in Lemma 6.9 below. This will very slightly increase the alphabet size, but not so much that it will affect the asymptotics, and then we can apply Theorem 6.3.

Remark 6.8. We believe it is possible to combine the two expander-based constructions into only one (with only one expander), which would give a slight improvement in the parameters. However, our approach here (using both Lemma 6.9 and Theorem 6.3 in serial) is more modular and still yields the desired asymptotic result, so we stick with it for simplicity of exposition.

We use the following lemma from [GKO$^+$17].

Lemma 6.9 ([GKO$^+$17], Distance amplification for local list-recovery)**.** *For any constants $\delta_1, \alpha_1, \gamma > 0$, there exists an integer $d \leq \mathsf{poly}(1/\delta_1, 1/\alpha_1, 1/\gamma)$ so that the following holds.*

- *Let $C_1 \subseteq \Sigma_1^{n_1}$ have rate R_1 and and distance δ_1 and be (t, α_1, ℓ_1, L)-locally-list-recoverable in time $T(C_1)$.*

- *Let $C_0 \subseteq \Sigma_0^{n_0}$ have rate R_0 and be (α_0, ℓ, ℓ_1)-globally-list-recoverable in time $T(C_0)$.*

- *Further suppose that $n_0 \geq d, |\Sigma_1| = |C_0|$.*

Then there exists a code $C \subseteq (\Sigma_0^{n_0})^{n_1}$ of block length n_1 over $\Sigma = \Sigma_0^{n_0}$ with rate $R_0 \cdot R_1$ that is $(t', \alpha_0 - \gamma, \ell, L)$-locally-list-recoverable for

$$t' = t \cdot n_0^2 \cdot \log(n_0).$$

Moreover there is a local list-recovery algorithm for C which runs in time $O(T(C_1)) + O(t \cdot T(C_0)) + \mathsf{poly}(t, n_0, \ell)$. Further, if both codes can be constructed in time $T_{construct}(C_0)$ and $T_{construct}(C_1)$ (in the sense that this is the time it take to generate a short description which suffices for polynomial-time encoding), then the final code C can be constructed in time $O(T_{construct}(C_0) + T_{construct}(C_1))$.

Remark 6.10. The statement of this lemma in [GKO$^+$17] is slightly different in that both the hypotheses and the conclusion are slightly stronger. They both specify linear codes, and the both have an additional "soundness" parameter which we will not need. However, an inspection of the proof shows that it goes through if these additional requirements and conclusions are dropped.

Corollary 6.11. *Let $\varepsilon > 0$ be sufficiently small, and let $m, \ell > 0$ be any integers. Suppose that ε, ℓ are constant, and m is growing. Then there is a code $C \subseteq \Sigma^N$ with rate at least $1 - 3\varepsilon$ so that C is $(t, \varepsilon, \ell, L)$-locally-list-recoverable for*

$$L = m^{O_{\ell,\varepsilon}(m^2)}$$

$$N = m^{O_{\ell,\varepsilon}(m^4)}$$

$$t = m^{O_{\ell,\varepsilon}(m^3)}$$

$$|\Sigma| = m^{O_{\ell,\varepsilon}(m^{2m+2})}$$

Further, C can be locally list recovered in time $\mathsf{poly}(t)$, and can be deterministically encoded in time $\mathsf{poly}(N)$.

Proof. We instantiate Lemma 6.9 using the code from Corollary 6.6 with rate $1 - 2\varepsilon$ as C_0 and a multivariate multiplicity code, from Theorem 5.1, as C_1. Thus, $C_0 \subseteq \Sigma_0^{n_0}$ has rate $R_0 = 1 - 2\varepsilon$ and is $(1 - \varepsilon, \ell, \ell_1)$-globally list-recoverable in time $\mathsf{poly}_{\ell,\varepsilon}(n_0)$, with $\ell_1 = O_{\ell,\varepsilon}(1)$. Meanwhile, $C_1 \subseteq \Sigma_1^{n_1}$ is (α, ℓ_1, L)-list-recoverable with

$$\alpha = O_{\ell,\varepsilon}(1/m^2),$$

$$L = m^{O_{\ell_1,\varepsilon}(m^2)} = m^{O_{\ell,\varepsilon}(m^2)},$$

$$N = m^{O_{\ell_1,\varepsilon}(m^4)} = m^{O_{\ell,\varepsilon}(m^4)},$$

$$t = m^{O_{\ell_1,\varepsilon}(m^3)} = m^{O_{\ell,\varepsilon}(m^3)},$$

$$|\Sigma| = m^{O_{\ell_1,\varepsilon}(m^{2m+2})}, = m^{O_{\ell,\varepsilon}(m^{2m+2})},$$

where we have used the fact that $\ell_1 = O_{\ell,\varepsilon}(1)$ to turn $O_{\ell_1,\varepsilon}(\cdot)$ into $O_{\ell,\varepsilon}(\cdot)$. The fact that $|C_0| = |\Sigma_1|$ thus implies that

$$n_0 = O(\log|\Sigma_1|) = O_{\ell,\varepsilon}(m^{2m+2}\log(m)).$$

Observe that n_0 is much larger than the $\mathsf{poly}(1/\alpha, 1/\varepsilon, 1/\delta_1) = \mathsf{poly}_{\ell,\varepsilon}(m)$ that is require by Lemma 6.9, for sufficiently large m. Now we check the conclusions. We immediately have the desired expressions for $N = n_1$ and L. The final alphabet size is $|\Sigma| = |\Sigma_1|^{1/R_0} = \mathsf{poly}(|\Sigma_1|) = m^{O_{\ell,\varepsilon}(m^3)}$, as before. The query complexity is

$$t' = t \cdot n_0^2 \log(n_0) = m^{O_{\ell,\varepsilon}(m^3)} \cdot m^{O_{\ell,\varepsilon}(m)} = m^{O_{\ell,\varepsilon}(m^3)}.$$

The dominating term in the local list-recovery time is $\mathsf{poly}(t)$, so the running time is still $\mathsf{poly}(t)$. And finally the time to construct a generator matrix for the inner code is $\mathsf{poly}(n_0) = \mathsf{poly}(N)$. Thus we may treat C as a deterministic code whose encoding map performs the search for C_0 in polynomial time and then encodes the message in polynomial time. $\square$

Finally we are ready to prove Theorem 6.1.

Proof of Theorem 6.1. Again we will use Theorem 6.3.

Let $C_0 \subseteq \Sigma_0^{n_0}$ be the code from Corollary 6.6, with rate $R + 4\varepsilon$. Thus, we have

$$|\Sigma_0| = (1 + \ell)^{O(1/\varepsilon^6)},$$

and C_0 is $(1 - R - 5\varepsilon, \ell, \ell_1)$-list-recoverable in time $\mathsf{poly}_{\ell,\varepsilon}(n_0)$, for $\ell_1 = O_{\ell,\varepsilon}(1)$. Moreover, C_0 can be constructed in time $n_0^{O_{\ell,\varepsilon}(\log(n_0))}$.

We choose $C_1 \subseteq \Sigma_1^{n_1}$ to be the code from Corollary 6.11, so that C_1 is a code of rate $(1 - 3\varepsilon)$, which is $(t, \varepsilon, \ell_1, L)$-list-recoverable, for

$$L = m^{O_{\ell,\varepsilon}(m^2)}$$

$$n_1 = m^{O_{\ell,\varepsilon}(m^4)}$$

$$t = m^{O_{\ell,\varepsilon}(m^3)}$$

$$|\Sigma_1| = m^{O_{\ell,\varepsilon}(m^{2m+2})}$$

where as in the proof of Corollary 6.11, above we have used the fact that $\ell_1 = O_{\ell,\varepsilon}(1)$ to hide dependence on ℓ_1 in the notation $O_{\ell,\varepsilon}(\cdot)$.

Now we apply Theorem 6.3, which concludes that there exists a code $C \subset \Sigma^N$ of rate $(1 - 3\varepsilon) \cdot (R + 4\varepsilon) \geq R$ which is $(t', 1 - R - 9\varepsilon, \ell, L)$-locally list-recoverable in time

$$O_{\ell,\varepsilon}(T(C_0) + T(C_1)) = \mathsf{poly}_{\ell,\varepsilon}(n_0) + m^{O_{\ell,\varepsilon}(m^3)} = m^{O_{\ell,\varepsilon}(m)} + m^{O_{\ell,\varepsilon}(m^3)} = m^{O_{\ell,\varepsilon}(m^3)} = \mathsf{poly}(t),$$

where

$$t' = O_{\ell,\varepsilon}(t \log|\Sigma_1|) = m^{O_{\ell,\varepsilon}(m^3)}.$$

Moreover, we have

$$N = O_{\ell,\varepsilon}(n_1 \cdot \log|\Sigma_1|) = m^{O_{\ell,\varepsilon}(m^3)}$$

and

$$|\Sigma| = |\Sigma_0|^{O(1/\varepsilon^4)} = (1 + \ell)^{O(1/\varepsilon^{10})} = O_{\ell,\varepsilon}(1).$$

Finally, to deterministically encode a message in C, we run the construction algorithm for C_0 (in time $n_0^{O(\log(n_0))} = m^{O(m^2 \log(m))} = \mathsf{poly}(N)$), and then use the polynomial-time encoding algorithm for C_1 which exists from Corollary 6.11.

Now applying the proof above with $\varepsilon/9$ instead of ε gives the theorem statement. $\qquad\square$

Finally, we prove Theorem 6.2.

Proof of Theorem 6.2. Choose $\delta, \tau, \varepsilon > 0$ constant and sufficiently small, and $\ell > 0$ constant. Let $C_0 \subseteq \Sigma_0^{n_0}$ be a code from Corollary 6.6, so that the rate of C_0 is $R + 2\varepsilon$, $|\Sigma_0| = (1 + \ell)^{O(1/\varepsilon^6)} = O_{\varepsilon,\ell}(1)$, and so that C_0 is $(1 - R - 3\varepsilon, \ell, \ell_1)$-list-recoverable in time $\mathsf{poly}_{\ell,\varepsilon}(n_0)$.

Now choose $C_1 \subseteq \Sigma_1^{n_1}$ which is (t, α, ℓ_1, L)-locally list-recoverable in time $\mathsf{poly}(t)$ for $\alpha = \Omega_{\varepsilon,\ell,\tau,\ell}(1)$, so that C_1 has rate $1 - \varepsilon$, query complexity $t = O_{\ell,\delta,\tau,\varepsilon}(n_1^{\tau/2})$, $L = O_{\ell,\delta,\tau,\varepsilon}(1)$, and $|\Sigma_1| = \mathsf{poly}_{\ell,\delta,\tau,\varepsilon}(N)$. Such a code exists by Theorem 5.2 (where we have used the fact that $\ell_1 = O_{\varepsilon,\ell}(1)$).

Now we apply Theorem 6.3, and conclude that there exists a code $C \subseteq \Sigma^N$, so that the rate of C is at least $(1 - \varepsilon)(R + 2\varepsilon) \geq R$ and so that

$$|\Sigma| = |\Sigma_0|^{O(1/(\varepsilon^3 \alpha))} = O_{\ell,\delta,\tau,\varepsilon}(1),$$

so that C is $(t', 1 - R - 7\varepsilon, \ell, L)$-list-recoverable for

$$t' = O\left(\frac{t \log|\Sigma_1|}{(R + 2\varepsilon) \log|\Sigma_0|}\right) = O_{\ell,\delta,\tau,\varepsilon}(N^{\tau/2} \log(N)) = O_{\ell,\delta,\tau,\varepsilon}(N^\tau),$$

in time

$$O_{\ell,\delta,\tau,\varepsilon}(T(C_0) + T'(C_1)) = O_{\ell,\delta,\tau,\varepsilon}(N^{O(\tau)}).$$

Moreover, C has rate $(1 - \varepsilon)(R + 2\varepsilon) \geq R$. The stated result follows by replacing ε with $\varepsilon/7$ in the above analysis. $\qquad\square$

7 Conclusion

We have shown that folded Reed-Solomon codes and multiplicity codes perform better than previously known in the context of (local) list-recovery. In addition to improving our knowledge about these codes, our results also lead to new and improved constructions of locally-list-recoverable codes. However, there is still much left to do, and we conclude with some open questions.

1. Theorem 3.1 shows that the list size for folded Reed-Solomon codes is $(\ell/\varepsilon)^{O\left(\frac{1}{\varepsilon}\log(\ell/\varepsilon)\right)}$. However, it is known that it is possible for codes to achieve a list size of $O(\ell/\varepsilon)$. It would be very interesting to strengthen our result to this bound, or even to reduce the list size to $\mathsf{poly}(\ell, 1/\varepsilon)$.

2. It would be very interesting to improve the list size in Theorem 4.4 on univariate multiplicity codes with large d to be $s^{O_{\ell,1/\varepsilon}(1)}$, rather than the current bound of $\ell^{O(s\log(s))} \cdot s^{O(1)}$. Beyond intrinsic interest, such an improvement would lead to an improvement in the query complexity of local list-recovery of multivariate multiplicity codes.

3. The algorithm given in Theorem 3.1 is a randomized algorithm. It is a very interesting open problem to design a deterministic list-decoding algorithm for folded RS codes with fixed polynomial running time that works up to list-decoding capacity.

4. We give a construction of a high-rate locally list-recoverable code with sub-polynomial query complexity. But we do not know if this is the best we could do; for example, could one get away with polylogarithmic query complexity in the same setting? Any lower bounds would be extremely interesting.

Acknowledgements

We would like to thank Atri Rudra and Venkatesan Guruswami for helpful discussions.

References

[AEL95]   Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 512–519. IEEE Computer Society, 1995.

[AS03]   Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 21–31. ACM Press, 1991.

[BK09]   K. Brander and S. Kopparty. List-decoding Reed-Muller over large fields upto the Johnson radius. *Manuscript*, 2009.

[DL12]   Zeev Dvir and Shachar Lovett. Subspace evasive sets. In *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*, pages 351–358. ACM Press, 2012.

[GI02]   Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 812–821. ACM Press, 2002.

[GI03]   Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *STOC*, pages 126–135, 2003.

[GI04]   Venkatesan Guruswami and Piotr Indyk. Linear-time list decoding in error-free settings. In *ICALP*, volume 3142, pages 695–707. Springer, 2004.

[GI05]   Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.

[GK16a]   Alan Guo and Swastik Kopparty. List-decoding algorithms for lifted codes. *IEEE Transactions on Information Theory*, 62(5):2719–2725, 2016.

[GK16b]   Venkatesan Guruswami and Swastik Kopparty. Explicit subspace designs. *Combinatorica*, 36(2):161–185, 2016.

[GKO⁺17]   Sivakanth Gopi, Swastik Kopparty, Rafael Oliveira, Noga Ron-Zewi, and Shubhangi Saraf. Locally testable and locally correctable codes approaching the gilbert-varshamov bound. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2073–2091. SIAM, 2017.

[GKS13]   Alan Guo, Swastik Kopparty, and Madhu Sudan. New affine-invariant codes from lifting. In *proceedings of the 4th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 529–540. ACM Press, 2013.

[GL89]   Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 25–32. ACM, 1989.

[GNP⁺13]   Anna C. Gilbert, Hung Q. Ngo, Ely Porat, Atri Rudra, and Martin J. Strauss. ℓ_2/ℓ_2-foreach sparse recovery with low risk. In *Automata, Languages, and Programming*, volume 7965 of *Lecture Notes in Computer Science*, pages 461–472. Springer Berlin Heidelberg, 2013.

[GR08]   Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

[GS99]   Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Trans. Information Theory*, 45(6):1757–1767, 1999.

[Gur04]   Venkatesan Guruswami. *List decoding of error-correcting codes: winning thesis of the 2002 ACM doctoral dissertation competition*, volume 3282. Springer Science & Business Media, 2004.

[GW13]   Venkatesan Guruswami and Carol Wang. Linear-algebraic list decoding for variants of reed-solomon codes. *IEEE Transactions on Information Theory*, 59(6):3257–3268, 2013.

[GX12]   Venkatesan Guruswami and Chaoping Xing. Folded codes from function field towers and improved optimal rate list decoding. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 339–350. ACM, 2012.

[GX13]   Venkatesan Guruswami and Chaoping Xing. List decoding reed-solomon, algebraic-geometric, and gabidulin subcodes up to the singleton bound. In *Proceedings of the 45th annual ACM symposium on Theory of Computing (STOC)*, pages 843–852. ACM Press, 2013.

[HOW15]   Brett Hemenway, Rafail Ostrovsky, and Mary Wootters. Local correctability of expander codes. *Information and Computation*, 243:178–190, 2015.

[HRW17]   Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes and applications. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2017.

[HW15]   Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. In *proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 9134 of *LNCS*, pages 701–712. Springer, 2015.

[INR10]   Piotr Indyk, Hung Q. Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 1126–1142, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics.

[KK17]   John Y Kim and Swastik Kopparty. Decoding reed–muller codes over product sets. *Theory of Computing*, 13(21):1–38, 2017.

[KM93]   Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993.

[KMRS17]   Swastik Kopparty, Or Meir, Noga Ron-Zewi, and Shubhangi Saraf. High-rate locally correctable and locally testable codes with sub-polynomial query complexity. *Journal of ACM*, 64(2):11:1–11:42, 2017.

[Kop15]   Swastik Kopparty. List-decoding multiplicity codes. *Theory of Computing*, 11(5):149–182, 2015.

[KSY14]   Swastik Kopparty, Shubhangi Saraf, and Sergey Yekhanin. High-rate codes with sublinear-time decoding. *Journal of ACM*, 61(5):28, 2014.

[KT00]   Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86. ACM Press, 2000.

[Lip90]   Richard J. Lipton. Efficient checking of computations. In *Proceedings of the 7th Annual ACM Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 207–215. Springer, 1990.

[Nie01]   R. R. Nielsen. *List decoding of linear block codes.* PhD thesis, Technical University of Denmark, 2001.

[NPR12]   Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently Decodable Compressed Sensing by List-Recoverable Codes and Recursion. In Christoph Dürr and Thomas Wilke, editors, *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 230–241, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[PV05]   Farzad Parvaresh and Alexander Vardy. Correcting errors beyond the guruswami-sudan radius in polynomial time. In *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*, pages 285–294. IEEE, 2005.

[RS96]   Ronitt Rubinfeld and Madhu Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[RT97]   M. Yu. Rosenbloom and M. A. Tsfasman. Codes for the m-metric. *Problemy Peredachi Informatsii*, 33(1):55–63, 1997.

[RW17]   Atri Rudra and Mary Wootters. Average-radius list-recovery of random linear codes. *arXiv preprint arXiv:1704.02420*, 2017.

[STV01]   Madhu Sudan, Luca Trevisan, and Salil P. Vadhan. Pseudorandom generators without the xor lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.

[Sud97]   Madhu Sudan. Decoding of reed solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[SY11]   Shubhangi Saraf and Sergey Yekhanin. Noisy interpolation of sparse polynomials, and applications. In *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, pages 86–92. IEEE, 2011.

[Vad12]   Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1–3):1–336, 2012.

A   A Wronskian lemma

In this section, we prove a lemma that shows that certain Wronskian determinants needed in Section 4 are nonzero.

Lemma A.1. *Suppose q is prime. Suppose $f_1, \ldots, f_t \in \mathbb{F}_q[X]$ are such that $\deg(f_i) \not\equiv \deg(f_j)$ mod q for $i \neq j$. Then $W(f_1, \ldots, f_t)$ is nonsingular.*

Proof. Let $d_j = \deg(f_j)$.

For $0 \leq i \leq t-1$ and $j \in [t]$, the (i, j) entry of the Wronskian matrix is $f_j^{(i)}(X)$, whose leading term is $\binom{d_j}{i} X^{d_j - i}$.

Thus the determinant of the Wronskian matrix has degree at most $D = \sum_{j=1}^{t} d_j - \sum_{i=0}^{t-1} i$. Furthermore, the coefficient of X^D in the Wronskian determinant equals the determinant of the matrix whose (i, j) entry (for $0 \leq i < t$, $1 \leq j \leq t$) equals $\binom{d_j}{i}$. This latter determinant is essentially a Vandermonde determinant, and by our hypotheses on the d_j, is nonzero in $\mathbb{F}_q$. Thus the determinant of the Wronskian matrix is a polynomial of degree exactly D, and in particular is nonzero. $\qquad\square$

B   Proof of Theorem 4.5

Here we prove Theorem 4.5, which adapts a theorem of [GW13] to our setting.

Proof of Theorem 4.5. We begin by giving the algorithm.

Algorithm FindPolys.

- **INPUT:** A parameter r, and access to $S : \mathbb{F}_q \to \binom{\mathbb{F}_q^s}{\ell}$

- **OUTPUT:** An affine subspace $v_0 + V$ that contains all polynomials that are close to S.

1. Set $D = (s - r + 1)(1 - \alpha)q - 1$.

2. By solving a linear system of equations over $\mathbb{F}_q$, find a nonzero $(A(X), B_0(X), \ldots, B_r(X)) \in (\mathbb{F}_q[X])^{r+1}$ such that:

   (a) $\deg(A) \leq D$, and for all i, $\deg(B_i) \leq D - d$.
   (b) For each λ with $0 \leq \lambda \leq s - r$, for each $x \in \mathbb{F}_q$, for each $y \in S(x)$:

   $$A(x) + \sum_{i=0}^{r-1} \sum_{j=0}^{\lambda} \binom{i+j}{i} y^{(i+j)} B^{(\lambda-j)}(x) = 0.$$

3. Let $v_0 + V$ be the affine space

   $$v_0 + V = \{f(X) \mid A(X) + \sum_{i=0}^{r-1} f^{(i)}(X) B_i(X) = 0\}.$$

4. Output $v_0 + V$.

We need to show:

1. The linear system has a nonzero solution,

2. $\mathcal{L} \subseteq v_0 + V$,

To see that the linear system has a nonzero solution, we show that the homogeneous system of linear equations in Step 2 of the algorithm has more variables than constraints. The total number of free coefficients in $A(X), B_0(X), \ldots, B_r(X)$ equals:

$$
\begin{aligned}
(D+1) + r(D-d+1) = &\; (D+1)(r+1) - d \cdot r \\
= &\; (s-r+1)(1-\alpha)q(r+1) - dr \\
> &\; (s-r+1)\left(\frac{\ell}{r+1} + \frac{r}{r+1}\frac{d}{(s-r+1)q}\right)q(r+1) - dr \\
= &\; (s-r+1)\ell q + dr - dr \\
= &\; (s-r+1)\ell q.
\end{aligned}
$$

The total number of constraints equals:

$$
q \cdot (s-r+1) \cdot \ell.
$$

By choice of D, the number of free coefficients is larger than the number of constraints. This proves that the algorithm can find a nonzero solution in Step 2.

Now take any $g(X) \in \mathcal{L}$. We will show that $g(X)$ is an element of the affine space $v_0 + V$ that is output by the algorithm.

Define $Q(X) = A(X) + \sum_{i=0}^{r-1} g^{(i)}(X)B_i(X)$. Observe that $\deg(Q) \leq D$.

Now take any $x \in \mathbb{F}_q$ and $y \in S(x)$ for which

$$
g^{(<s)}(x) = y. \tag{5}
$$

Let λ be an integer with $0 \leq \lambda \leq s-r$. Then by the chain rule for Hasse derivatives:

$$
\begin{aligned}
Q^{(\lambda)}(x) &= A^{(\lambda)}(x) + \sum_{i=0}^{r-1} \left(g^{(i)} \cdot B_i\right)^{(\lambda)}(x) \\
&= A^{(\lambda)}(x) + \sum_{i=0}^{r-1}\sum_{j=0}^{\lambda} \left(g^{(i)}\right)^{(j)}(x)B^{(\lambda-j)}(x) \\
&= A^{(\lambda)}(x) + \sum_{i=0}^{r-1}\sum_{j=0}^{\lambda} \binom{i+j}{i} g^{(i+j)}(x)B^{(\lambda-j)}(x) \\
&= A^{(\lambda)}(x) + \sum_{i=0}^{r-1}\sum_{j=0}^{\lambda} \binom{i+j}{i} y^{(i+j)}B^{(\lambda-j)}(x) \\
&= 0
\end{aligned}
$$

Since this holds for every λ with $0 \leq \lambda \leq s-r$, we get that:

$$
\mathsf{mult}(Q, x) \geq s-r+1.
$$

By assumption on g, there are at least $(1-\alpha)q$ values of $x \in \mathbb{F}_q$ such that there exists some $y \in S(x)$ for which Equation (5) holds. Thus there are at least $(1-\alpha)q$ points where Q vanishes with multiplicity at least $s - r + 1$. Since $\deg(Q) \le D < (s - r + 1)(1 - \alpha)q$, we conclude that $Q(X) = 0$.

By definition of $Q(X)$ and $v_0 + V$, this implies that $g(X) \in v_0 + V$, as desired. $\qquad\square$

C   Coordinate restrictions of subspaces

In this section, we discuss the relationship between the list-recovery results for Folded Reed-Solomon codes and univariate multiplicity codes, as well as the ideas that go into their proofs.

First we point out that we could use Lemma 1.1 to analyze Algorithm PruneListFRS and Algorithm PruneListMULT. This can be used in place of Theorem 3.7 and Theorem 4.8, and would give a proof of the list recoverability of Folded Reed-Solomon and univariate multiplicity codes *from very small error*. However, this approach is not able to reproduce the capacity achieving list-decodability in Theorem 3.1 and Theorem 4.1, and only gives a quantiatively weaker version of Theorem 4.4[10].

The statement of Theorem 4.4 on list-recovery of whole field univariate multiplicity codes is noticeably weaker than the statement of Theorem 3.1 on list-recovery of Folded Reed-Solomon codes: the former only gives list-recoverability in the presence of very few errors. The proof of the former is also noticeably more involved. Inspecting the components of the proofs, we see that this difference arises from the significantly different quantitative natures of the analyses of algorithm PruneListFRS and PruneListMULT.

The following example shows that this difference is not just an artifact of the analysis: there are instances where the algorithm PruneListMULT (which is exactly analogous to the algorithm PruneListFRS) really requires the error fraction to be very small, and produces an output list size which is exponentially large in s.

Example C.1. Let $V \subseteq \mathbb{F}_q[X]$ be given by:

$$V = \{ \sum_{i < d/q} a_i X^{iq} \mid a_i \in \mathbb{F}_q \}.$$

Let τ be any integer $< d/q - 1$, and let $b_1, \ldots, b_\tau \in \mathbb{F}_q$ be distinct. Then:

$$V \cap \bigcap_{j=1}^{\tau} H_{b_j} = \left\{ \prod_{j=1}^{\tau}(X - b_j)^q \cdot \left(\sum_{i < d/q - \tau} c_i X^{iq} \right) \mid c_i \in \mathbb{F}_q \right\}.$$

In particular:
$$\dim(V \cap H_{b_1} \cap H_{b_2} \cap \ldots \cap H_{b_\tau}) = \dim(V) - \tau \ge 1.$$

---

[10] The version proved using Lemma 1.1 requires $\alpha < \frac{1}{s^2}$, and gives an output list-size of $\ell^{O(s^2 \log s)}$. This in turn would be sufficient to give a local list-recovery algorithm for length N multivariate multiplicity codes with query complexity $\exp(\log^{5/6}(N))$.

This means that when we run the algorithm PruneListMULT on V as input, the step where we search for $P(X) \in v_0 + V$ will NEVER find a unique solution.

Thus for the algorithm PruneListMULT to succeed with positive probability we must have $\tau \geq d/q$. For the constant rate setting, this means that $\tau = \Omega(s)$. With $\tau = \Omega(s)$, the success probability of PruneListMULT is at most $(1 - \alpha)^{\Omega(s)}$, and the output list-size is at least $\ell^{\Omega(s)}$. Thus the analysis of the algorithm PruneListMULT in Lemma 4.14 cannot be improved.

D List recovering Reed-Muller codes on product sets

In this section, we prove Lemma 5.4 about list-recovery of Reed-Muller codes that we need for RecoverCandidates and Lemma 5.3.

We will prove Lemma 5.4 by reducing the case of tuples of polynomials to the case of a single polynomial over a large field. We go through the details in the next two subsections.

D.1 Replacing vector values with big field values

Let $\mathbb{F}$ be a finite field, and let $\mathbb{K}$ be the degree t field extension of $\mathbb{F}$. Let $\phi : \mathbb{F}^t \to \mathbb{K}$ be an arbitrary $\mathbb{F}$-linear bijection. Let $U \subseteq \mathbb{F}$ (and thus $U \subseteq \mathbb{K}$).

Then to every function $f : U^m \to \mathbb{F}^t$, we can associate a function $\tilde{f} : U^m \to \mathbb{K}$, where $\tilde{f} = \phi \circ f$. This identifies the underlying Hamming metric spaces. The key observation is that under this identification, f is the evaluation table of a tuple of t polynomials in $\mathbb{F}[X_1, \ldots, X_m]$ of degree $\leq d$ if and only if $\tilde{f} : U^m \to \mathbb{K}$ is the evaluation table of a degree $\leq d$ polynomial in $\mathbb{K}[X_1, \ldots, X_m]$. Thus questions about decoding (list-decoding, list-recovering) vector valued polynomial codes reduce to questions about decoding (list-decoding, list-recovering) scalar valued polynomial codes over larger fields.

Through this connection, Lemma 5.4 is a consequence of the following lemma (and the fact that $\mathbb{K}$ can be constructed in randomized $\mathsf{poly}(\log |\mathbb{K}|)$ time).

Lemma D.1 (Reed-Muller list recovery on a grid)**.** *Let $\ell, \tilde{s}, K, m$ be given parameters. Let $\mathbb{K}$ be a finite field. Suppose that $U \subseteq \mathbb{K}$ and $|U| \geq 2\ell\tilde{s}K$. Let $\alpha < 1 - \frac{1}{\sqrt{K}}$ be a parameter.*

Then for every $f : U^m \to \binom{\mathbb{K}}{\ell}$, if

$$\mathcal{L} = \{Q(Y_1, \ldots, Y_m) \in \mathbb{K}[Y_1, \ldots, Y_m] \mid \deg(Q) \leq \tilde{s} \text{ and } \Pr_{\mathbf{u} \in U^m}[Q(\mathbf{u}) \notin f(\mathbf{u})] < \alpha\},$$

we have:

1. $|\mathcal{L}| \leq 2K\ell$.

2. *Suppose further that $K > m^2$ and $\alpha \leq 1 - \frac{m}{\sqrt{K}}$, then there is a $\mathsf{poly}(|U|^m, \log |K|)$ time algorithm to compute $\mathcal{L}$.*

D.2   Proof of Lemma D.1

Proof of Lemma D.1. Item 1 will follow from the Johnson bound for list-recovery, which is a general statement implying good list recoverability for codes with large distance. Specifically, Lemma V.2 in [GKO+17] (see also Corollary 3.7 in [Gur04]) states that a code of distance δ is (α, ℓ, L) list-recoverable for $\alpha > 1 - \sqrt{\ell(1-\delta)}$ and $L = \frac{\ell}{(1-\alpha)^2 - \ell(1-\delta)}$. In our setting, the code of polynomials of degree at most $\tilde{s}$ on U^m has distance δ at least $1 - \frac{\tilde{s}}{|U|} \geq 1 - \frac{1}{2K\ell}$. Thus for $\alpha < 1 - \frac{1}{\sqrt{K}}$, the Johnson bound for list-recovery implies that we can take $L = 2K\ell$, as desired.

We prove Item 2 by induction on m. The $m = 1$ case is simply the Sudan list-recovery algorithm [Sud97] for Reed-Solomon codes, which works with the claimed parameters (since the total number of points $n = \ell \cdot |U|$, the number of agreement points A is at least $(1 - \alpha)n \geq \frac{n}{\sqrt{K}}$, and so $A \geq 2\sqrt{n\tilde{s}}$, which is the requirement for the Sudan algorithm to work).

For general m, we first do list-recovery on $m - 1$ dimensional grids, and then combine the results using list-recovery for vector-valued univariate polynomials. We crucially use the previous combinatorial bound on the list size to ensure that the intermediate list size is under control (as the recursion unfolds).

More concretely, the algorithm proceeds as follows:

1. First, for each setting of $u \in U$, we consider the received word $f_u : U^{m-1} \to \binom{\mathbb{K}}{\ell}$, given by $f_u(\mathbf{y}) = f(\mathbf{y}, u)$.

   Now list-recover f_u to radius $\beta = \left(1 - \frac{m-1}{\sqrt{K}}\right)$ to find the set of nearby $m - 1$-variate polynomials $\mathcal{L}_u$. By the previous combinatorial bound, we can assume (after confirming that all elements of $\mathcal{L}_u$ are indeed close to f_u) that:

   $$|\mathcal{L}_u| \leq 2K\ell.$$

2. Next we combine all $\mathcal{L}_u$. Let $M_1(Y_1, \ldots, Y_{m-1}), \ldots, M_t(Y_1, \ldots, Y_{m-1})$ be all the $(m-1)$-variate monomials of total degree at most $\tilde{s}$. Define a function

   $$g : U \to \binom{\mathbb{K}^t}{2K\ell}$$

   as follows: for each $u \in U$ and each element P of $\mathcal{L}_u$, include the vector of coefficients of P into $g(u)$.

   Then, using a vector-valued Sudan list-recovery algorithm for univariate polynomials (obtained from the standard scalar-valued Sudan list-recovery algorithm via the connection in Section D.1), we find all tuples of univariate polynomials $\mathbf{P}(Z) = (P_1(Z), \ldots, P_t(Z)) \in (\mathbb{K}[Z])^t$ such that:

   $$\Pr_{u \in U}[\mathbf{P}(u) \notin g(u)] < \gamma,$$

   where $\gamma = 1 - \frac{1}{\sqrt{K}}$.

3. For each $\mathbf{P}(Z) \in (\mathbb{K}[Z])^t$ found in the previous step, we construct the polynomial:

$$R(Y_1, \ldots, Y_{m-1}, Y_m) = \sum_{i=1}^{t} M_i(Y_1, \ldots, Y_{m-1}) P_i(Y_m).$$

If this polynomial has total degree at most $\tilde{s}$ and is α-close to f, then we include it in the output list.

To prove correctness of this algorithm, consider any $Q(Y_1, \ldots, Y_m) \in \mathcal{L}$. Let $Q_u(Y_1, \ldots, Y_{m-1}) = Q(Y_1, \ldots, Y_{m-1}, u)$. Then we have:

$$\mathbb{E}_{u \in U}[\text{dist}(Q_u, f_u)] = \text{dist}(Q, f) < \alpha.$$

Thus

$$\Pr_{u \in U}[\text{dist}(Q_u, f_u) \geq \beta] \leq \frac{\alpha}{\beta} < \frac{1 - m/\sqrt{K}}{1 - (m-1)/\sqrt{K}} \leq 1 - \frac{1}{\sqrt{K}} = \gamma.$$

This implies that for at most γ-fraction of $u \in U$, we have that $Q_u \notin \mathcal{L}_u$.

Write $Q(Y_1, \ldots, Y_m)$ as $\sum_{i=1}^{t} M_i(Y_1, \ldots, Y_{m-1}) G_i(Y_m)$. Then the above discussion means that for at most γ fraction of $u \in U$, we have that $(G_1(u), \ldots, G_t(u)) \notin g(u)$. This implies that $(G_1(u), \ldots, G_t(u))$ will be included in the list returned by the univariate list-recovery algorithm in Step 2, and thus that Q will be included in the output of the algorithm in Step 3.

This completes the proof of correctness. The bound on the running time follows immediately from the description of the algorithm (using the fact that $t \leq (\tilde{s} + m)^m \leq |U|^m$). $\square$

E Proof of Theorem 6.3

In this section, we prove Theorem 6.3. Our proof is based on a construction first attributed to [AEL95], which has since been used in many works to improve the parameters of list-recoverable and locally list-recoverable codes. We include the proof here for completeness.

Proof of Theorem 6.3. The construction uses three ingredients: a bipartite expander graph G; the code C_1 guaranteed in the problem statement; and an inner code C_0 as in the theorem statement. Notice that the size and rate of C_0 implies that $n_0 = \left\lceil \frac{\log |\Sigma_1|}{R \cdot \log |\Sigma_0|} \right\rceil$. Choosing $|\Sigma_0| = (\max\{2, \ell\}^{O(1/\varepsilon)}$, this reads

$$n_0 = O\left(\frac{\log |\Sigma_1| \cdot \varepsilon}{R \cdot (1 + \log(\ell))}\right).$$

It is known that the double-cover of a Ramanujan graph has the properties we want; we state these properties formally in the following claim.

Claim E.1. *[See [KMRS17], Lemma 2.7] Let $\xi, \varepsilon, R \in [0, 1]$, so that ξ and ε are sufficiently small. For infinitely many integers $N > 0$, there exists a $D = O(1/\xi\varepsilon^2)$ so that the following holds. There exists a bipartite expander graph $G = (V_L, V_R, E)$ be a bipartite expander graph with N vertices on*

each side, with degree D, and with the following property: for any set $Y \subseteq V_R$ of right-hand-vertices with $|Y| \geq (R + 4\varepsilon)N$, we have

$$|\{v \in V_L \ : \ |\Gamma(v) \cap Y| < (R + 3\varepsilon)D\}| \leq \xi N,$$

where $\Gamma(v) \subseteq V_R$ is the set of neighbors of v in G.

We will instantiate Claim E.1 with the ε from the guarantee in C_0, and with $\xi := \gamma \cdot \varepsilon$. Thus, we have $D = O(1/(\varepsilon^3 \gamma))$. With these ingredients C_0 and G in hand, let C_1 be as in the theorem statement, and let $\bar{C} = C_0 \circ C_1$ be the concatenation of C_0 and C_1. Thus, a codeword in $\bar{C}$ has the form

$$\bar{c} = (C_0(x_1), C_0(x_2), \ldots, C_0(x_{n_1})) \in (\Sigma_0^{n_0})^{n_1}$$

for $(x_1, \ldots, x_{n_1}) \in C_1$. Suppose without loss of generality that D divides n_0. (Otherwise, we may pad the codewords of C_0 with zeros to make this be the case). Then break up the codewords $\bar{c}$ into $N = n_0 n_1/D$ blocks of length D:

$$\bar{c} = (y^{(1)}, y^{(2)}, \ldots, y^{(N)}) \in (\Sigma_0^D)^N.$$

We will form our final code $C \subseteq (\Sigma_0^D)^N$ as follows: for each codeword $(y^{(1)}, \ldots, y^{(N)}) \in \bar{C}$ (thought of as an element of $(\Sigma_0^D)^N$), define a codeword $c = (c^{(1)}, \ldots, c^{(N)}) \in C \subseteq (\Sigma_0^D)^N$ by

$$c_\ell^{(j)} = y_r^{(i)},$$

where $j = \Gamma_r(V_L[i])$ and $i = \Gamma_\ell(V_R[j])$ and where the notation $\Gamma_r(v)$ denotes the r'th neighbor of vertex v (according to some arbitrary order) and $V_L[i]$ denotes the i'th vertex in V_L (again according to an arbitrary order).

The code $C \subseteq (\Sigma_0^D)^N$ will be the set of all codewords obtained this way. Notice that the rate of C is the same as that of $\bar{C}$, since the operation above just permutes the symbols of a codeword. Thus, the rate of C is

$$(1 - \zeta) \cdot R,$$

as claimed.

Global list-recovery. We first argue that if C_1 is efficiently (ε, ℓ_1, L)-list-recoverable, then C is efficiently $(1 - R - 4\varepsilon, \ell, L)$-list-recoverable. Suppose that $S_1, \ldots, S_N \subset \Sigma_0^D$ have $|S_i| \leq \ell$, and suppose that $c \in C$ has $c^{(i)} \in S_i$ for all $i \in Y$, for some set $Y \subseteq [N]$ of size at least $(R + 2\gamma)N$. Suppose that c is obtained as above from $(y^{(1)}, \ldots, y^{(N)}) \in \bar{C}$, which is obtained by concatenation from $(x_1, \ldots, x_{n_1}) \in C_1$. Suppose that z is the original message so that $C_1(z) = (x_1, \ldots, x_{n_1})$. Thus, our goal is to recover a short list S of size at most L, so that $z \in S$.

For $i \in [N]$ and $r \in [D]$, let

$$T_{i,r} = \{\alpha \in \Sigma_0 \ : \ \exists \beta \in S_j \subseteq \Sigma_0^D, \beta_\ell = \alpha, j = \Gamma_r(V_L[i]), i = \Gamma_\ell(V_R[j])\}.$$

That is, $T_{i,r}$ is the list of symbols in Σ_0 that $y_r^{(i)}$ could be that is consistent with the lists $S_1, \ldots, S_N$.

The decoding algorithm for C is then straightforward: given $S_1, \ldots, S_N$, compute the lists $T_{i,r}$, and then run the list-recovery algorithm for C_0 on each block $C_0(x_t)$ to obtain a list $S_t' \subset \Sigma_1$ of possible

values of x_t. Since C_0 is obtained via a random coding argument, there is not an efficient algorithm for this; however, C_0 is small enough that the brute-force decoding algorithm will do. Next, we run the list-recovery algorithm for C_1 on the lists S'_t, to obtain our final list S of size at most L.

Before we show that this is correct, consider the run-time of this algorithm. The dominating term in the running time is the time to run the list-recovery algorithms of C_0 and C_1. The time to list-recover C_1 is given by $T(C_1)$, and the time to list-recover each of the n_1 copies of C_0 is bounded above by $O(|C_0|) = O(|\Sigma_1|)$. Together, these expressions give the runtime bound claimed in the theorem.

Next, we argue that this algorithm is correct. Let $c \in C$ be as above, so that $c^{(i)} \in S_i$ for all $i \in Y$, for some set $Y \subseteq [N]$ of size at least $(R + 4\varepsilon)N$. By the expansion property of G, the set $S \subseteq [N]$ of indices i so $V_L[i]$ has at most $(R + 3\varepsilon)D$ neighbors in Y has size $|W| \leq \xi N = \gamma\varepsilon N$. Thus, for each $i \notin W$, $y_r^{(i)} \in T_{i,r}$ for at least $(R + 3\varepsilon)D$ values of $r \in [D]$.

Now consider the blocks $\bar{c} = (C_0(x_1), \ldots, C_0(x_{n_1}))$. Each $C_0(x_t)$ is made up of n_0/D blocks $y^{(i)} \in \Sigma_0^D$. By an averaging argument, since $|W| \leq \gamma\varepsilon N$, at most an γ-fraction of the blocks $C_0(x_t)$ have more than an ε-fraction of its constituent length-D blocks in W.

Suppose that the block $C_0(x_t)$ is one of the $(1-\gamma)$-fraction of the blocks for which this does not hold; that is, at most an ε-fraction of the blocks $y^{(i)}$ in $C_0(x_t)$ have $i \in W$. Then the number of symbols $y_r^{(i)} \in \Sigma_0$ that make up $C_0(x_t)$ so that $y_r^{(i)} \in T_{i,r}$ is at least

$$|\{(i,r) \ : \ r \in [D], y^{(i)} \text{ is contained in } C_0(x_t), \text{ and } y_r^{(i)} \in T_{i,r}\}| \geq (R+3\varepsilon) \cdot D \cdot (1-\varepsilon)\frac{n_0}{D} \geq (R+\varepsilon)n_0.$$

Since C_0 is $(1-R-\varepsilon, \ell, \ell_1)$-list-recoverable, this implies that for all such t, the list-recovery algorithm for C_0 returns a list S'_t of length at most ℓ_1 so that $x_t \in S'_t$.

Now since there are at least $(1 - \gamma)n_1$ such blocks, the list-recovery algorithm for C_1 will return a list S of size at most L, so that the original message z is guaranteed to be contained in S. Thus, the algorithm is correct.

Local list-recovery. Suppose that C_1 is (t, γ, ℓ_1, L)-locally list-recoverable via an algorithm A, which expects advice $\xi \in [L]$. Now we may use exactly the same construction as above to obtain a locally list-recoverable code.

More precisely, for any $x \in C_1$ resulting in a codeword $c \in C$ that agrees with a $1 - R - 4\varepsilon$ fraction of the lists S_i, the argument above shows that, for a $1 - \gamma$ fraction of the indices $i \in [n_1]$, we may obtain a list $S'_i \subseteq \Sigma_1$ of size at most L, so that $x_i \in S'_i$, using $n_0 = O(\log_{|\Sigma_0|}|\Sigma_1|/R)$ queries to the input lists S_i. This immediately implies the following local list-recovery algorithm:

Algorithm LocalListRecoveryAEL

- **INPUT:** Query access to the lists $S_1, \ldots, S_N \subseteq \Sigma_0^D$, and an index $i \in [N]$

- **ADVICE:** $\xi \in [L]$

---

1. For each $j \in \Gamma(V_R[i])$:

   - Let $r \in [n_1]$ be the index so that the block $y^{(j)}$ is contained in $C_0(x_r)$.
   - Run C_1's local list-recovery algorithm A with advice ξ, simulating query access to the lists S'_s by using n_0 queries to the lists $S_1, \ldots, S_N$, as described above. This returns x_r.
   - Use C_0's encoder (which in this case we may treat as a look-up table) to obtain $C_0(x_r)$ and hence $y^{(j)}$.

2. Given $y^{(j)}$ for each $j \in \Gamma(V_R[i])$, assemble $c^{(i)}$ and return it.

The correctness of this algorithm follows from the correctness of the query-simulation procedure, which was shown above.

The query complexity is $D \cdot t \cdot n_0$, because for each of D values of j, we need to simulate t queries to the lists S'_r, each of which requires n_0 queries to the lists S_r. Plugging in our settings of D and n_0 gives the query complexity claimed in the theorem.

The running time for each simulated query to a list S'_r is dominated by the time to correct C_0, which is $O(|C_0|) = O(|\Sigma_1|)$ by using a brute-force algorithm, plus the time used by A, which is $T(C_1)$. There are $D = O(1/(\varepsilon^3 \gamma))$ such queries which gives the running time claimed in the theorem statement. $\qquad \square$