2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Improved decoding of Folded Reed-Solomon and Multiplicity Codes

Swastik Kopparty*, Noga Ron-Zewi†, Shubhangi Saraf* and Mary Wootters‡
*Department of Mathematics and Department of Computer Science,
Rutgers University, Piscataway, NJ, USA, Email: swastik.kopparty@gmail.com, shubhangi.saraf@gmail.com,
†Department of Computer Science, University of Haifa, Haifa, Israel, Email: noga@cs.haifa.ac.il
‡Department of Computer Science and Department of Electrical Engineering,
Stanford University, Stanford, CA, USA, Email: marykw@stanford.edu

***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.

***Index Terms*—list decodable codes; locally decodable codes; folded Reed-Solomon codes; multiplicity code;**

The research is supported in part by NSF grants CCF-1253886, CCF-1540634, CCF-1350572, and CCF-1657049, and an NSF-BSF grant CCF-1814629 and 2017732.

I. 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

2575-8454/18/$31.00 ©2018 IEEE
DOI 10.1109/FOCS.2018.00029

low-degree polynomials.[1] In the late 1990's, Guruswami and Sudan [1], [2] 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 [3] and locally decodable [4], [5]. Using this locality in concert with the Guruswami-Sudan algorithm leads to local list-decoding schemes [6], [7] 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 II.

Folded Reed-Solomon codes, introduced by Guruswami and Rudra in [8], 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 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 [8] 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.

[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 $f \in \mathbb{F}[X]$.

[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.

Multiplicity codes, introduced in the univariate setting by Rosenbloom and Tsfasman in [9] and in the multivariate setting by Kopparty, Saraf and Yekhanin in [10], 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 [10] to have strong locality properties, and were the first constructions known of high-rate locally decodable codes. Meanwhile, univariate multiplicity codes were shown in [11], [12] 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 ([8], [10], [11], [12], [14], [15], [16], 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

[3]They were previously shown to be list-decodable up to the Johnson bound by Nielsen [13].

linear[4] capacity-achieving list-decodable codes with constant alphabet size and constant output list size.

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

A. Related work

a) 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 [1], [17] for overviews). Initially, the generalization to list recovery was used as a building block towards constructions of list decodable and uniquely decodable codes [18], [19], [20], [21], [15], [16], [22], although it has since found additional applications in algorithm design [23], [24], [25].

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 [8] (building on the work of Parvaresh and Vardy [26]) 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 [12], [11] 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 [27], [28], [29], and (2) concatenation and *distance amplification* using expander graphs [30], [20], [31], [16], [22]. 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* [14], [12], [27], [28], [29].

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

b) 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 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 $|\{i : c_i \notin S_i\}| \leq \alpha n$. The requirement is that if A_r corresponds to a codeword c, 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 $\ell = 1$.

[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".

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 [32], learning theory [33], and hardness amplification and derandomization [7]. 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 [22]. The approach of [22] is as follows: it takes a folded AG subcode from [28], [29] (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 [30] 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 [22] 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 [6] and Sudan, Trevisan and Vadhan [7] gave algorithms for locally list-decoding Reed-Muller codes up the the Johnson bound[5]. This algorithm also extends naturally to local list-recovery up to the Johnson bound [16], 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 [11].

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.

B. 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 IV.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.** We recover the same quantitative results as Theorem IV.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, we show 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, this result 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. We show that high-rate multivariate multiplicity codes are locally list-recoverable. More precisely, we show that for constant ℓ, ε, a multivariate multiplicity code of length n with rate $1-\varepsilon$ is effi-

[5]Technically these algorithms only came within a factor $\sqrt{2}$ of the Johnson bound. To go all the way to the Johnson bound, one needs some additional ideas [34]; see [35], [11] for further variations on this.

ciently (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. We also instantiate the same argument with 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.** The aforementioned results 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.
  The only previous construction of capacity-achieving locally list-recoverable codes (or even high-rate locally list-recoverable codes) is due to [22], 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 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 [15], which is $\widetilde{\exp}(\log^{1/2} n)$ (for the same codes). We can also achieve arbitrary polynomially small query complexity, and constant list-size. This improves upon the codes of [22].
- **Deterministic constructions of capacity-achieving list-recoverable codes with constant alphabet size and list size.** Our result in Theorem IV.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 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 $n^{O(1)} \cdot \log(n)^{O_{\ell,\varepsilon}(1)}$.[6]
  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 [14] (which are nonlinear subcodes of Folded Reed-Solomon codes) to construct nonlinear codes with the same behavior.

[6]Unfortunately, the dependency of alphabet size, list size, and running time on ℓ that we obtain is far from optimal. It is an interesting open problem to improve this dependency.

C. Overview of techniques

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

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 relative 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 [8] 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 [12]. 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 for any $c \in \mathcal{L}$, c is included in the output of PRUNE 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 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 [36] shows that for any linear space V with dimension k and relative 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 [36] is as follows. First we note the following simple but important lemma:

Lemma I.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 relative 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 I.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 IV.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 [12] 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 I.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 I.1. Such a lemma was proved in [29], 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 $\mathbb{F}_q$. The proof follows the same outline, using a different but analogous lemma from [29].

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. 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 gives better quantitative bounds than the previous approach, and requires us to open up the relevant lemma from [29]. 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.

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 [6] and Sudan, Trevisan and Vadhan [7]. 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[7].

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 [7] 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 supposed 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 polynomial in $\mathcal{L}_\lambda$ takes the same value at $\mathbf{a}$ as $Q|_\lambda$ does. The first event is high probability by standard sampling bounds, and the second is high 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.

[7]These ideas can also be used to improve the analysis of the [6] and [7] 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.

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 $\mathbf{a}$. 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[8] on $\mathbf{a}$ as any other element of $\mathcal{L}_\lambda$ – this will allow us to identify it 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 $Q^{<s}(\mathbf{x})$.

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. 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.

Summarizing, our local list-recovery algorithm works as follows. First, we run RecoverCandidates on a random

[8]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 $\mathbf{a}$.

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 $\mathbf{x}$, and so we can use the fact that multivariate multiplicity codes are locally correctable to finish the job [10]. 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.

3) Organization: Due to space limitation in the rest of the paper we only present our results on list recovery of Folded RS codes.

II. 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, $\mathsf{poly}_a(n) = n^{O_a(1)}$.

A. 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 $\mathrm{dist}(c, w) \leq \alpha$.

B. 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 $\ell = 1$.

III. 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 III.1 ([8]). *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 $1 - d/(sn)$.*

IV. 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 IV.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 IV.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 IV.1 consists of two main steps. The first step, from [12], shows that the output list is contained in a low dimensional subspace. The second step, which relies on results from [29], 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 IV-A and IV-B, respectively, followed by the proof of Theorem IV.1 in Section IV-C.

A. Output list is contained in a low dimensional subspace

The following theorem from [12] shows that the output list is contained in a low dimensional subspace, which can also be found efficiently.

Theorem IV.3 ([12], Theorem 7). *Let q be a prime power, and let s, d, n, ℓ, r be nonnegative integers such that $n \le (q-1)/s$ and $r \le s$. 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:*

$$\alpha \le 1 - \frac{\ell}{r+1} - \frac{r}{r+1} \cdot \frac{s}{s-r+1} \cdot \frac{d}{sn}. \qquad (1)$$

Let

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

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) $\dim(V) \le r - 1$.

Remark IV.4. Theorem 7 of [12] 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 $\{\gamma^{si} \mid 0 \le i \le (q-1)/s - 1\}$, and $\ell \in \mathbb{N}$ (for the latter see discussion at end of Section 2.4 of [12]).

B. 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 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 IV.1. Now that we know that $|\mathcal{L}|$ is small, our final algorithm simply runs 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 IV.1.

We start by describing the algorithm 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 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 IV.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 IV.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. □

It remains to prove the following claim.

Claim IV.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 [29].

Theorem IV.7 ([29], 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 IV.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 IV.7,

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

Thus

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

and

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

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

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

□

C. Proof of Theorem IV.1

We now prove Theorem IV.1 based on Theorem IV.3 and Lemma IV.5.

Proof of Theorem IV.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 $\mathrm{dist}(\mathsf{FRSEnc}_s(P), S) \le \alpha$.

Let $v_0 + V$ be the subspace found by the algorithm of Theorem IV.3 for S and $r = \frac{4\ell}{\varepsilon}$ (so $r \le \frac{1}{4}\varepsilon s$ by assumption that $s \ge 16\ell/\varepsilon^2$). Note that for this choice of r the RHS of (1) is at least

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

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

Next we invoke Lemma IV.5 with S, $v_0 + V$ and $\tau = O(\frac{1}{\varepsilon}\log(\ell/\varepsilon))$. Then the algorithm PruneListFRS returns a list $\mathcal{L}'$ of size at most ℓ^τ such that each polynomial $P(X)$ of degree at most d with $\mathrm{dist}(\mathsf{FRSEnc}_s(P), S) \le \alpha$ is included in $\mathcal{L}'$ with probability p_0, which is at least

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

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

The above implies in turn that

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

Moreover, by running the algorithm 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 $\mathsf{poly}(\log q, s, d, n, (\ell/\varepsilon)^{\log(\ell/\varepsilon)/\varepsilon})$.

□

ACKNOWLEDGEMENTS

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

REFERENCES

[1] M. Sudan, "Decoding of reed solomon codes beyond the error-correction bound," *Journal of Complexity*, vol. 13, no. 1, pp. 180–193, 1997.

[2] V. Guruswami and M. Sudan, "Improved decoding of reed-solomon and algebraic-geometry codes," *IEEE Trans. Information Theory*, vol. 45, no. 6, pp. 1757–1767, 1999. [Online]. Available: http://dx.doi.org/10.1109/18.782097

[3] R. Rubinfeld and M. Sudan, "Robust characterizations of polynomials with applications to program testing," *SIAM J. Comput.*, vol. 25, no. 2, pp. 252–271, 1996. [Online]. Available: http://dx.doi.org/10.1137/S0097539793255151

[4] R. J. Lipton, "Efficient checking of computations," in *Proceedings of the 7th Annual ACM Symposium on Theoretical Aspects of Computer Science (STACS)*. Springer, 1990, pp. 207–215.

[5] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy, "Checking computations in polylogarithmic time," in *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 1991, pp. 21–31. [Online]. Available: http://doi.acm.org/10.1145/103418.103428

[6] S. Arora and M. Sudan, "Improved low-degree testing and its applications," *Combinatorica*, vol. 23, no. 3, pp. 365–426, 2003.

[7] M. Sudan, L. Trevisan, and S. P. Vadhan, "Pseudorandom generators without the xor lemma," *Journal of Computer and System Sciences*, vol. 62, no. 2, pp. 236–266, 2001.

[8] V. Guruswami and A. Rudra, "Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy," *IEEE Transactions on Information Theory*, vol. 54, no. 1, pp. 135–150, 2008.

[9] M. Y. Rosenbloom and M. A. Tsfasman, "Codes for the m-metric," *Problemy Peredachi Informatsii*, vol. 33, no. 1, pp. 55–63, 1997.

[10] S. Kopparty, S. Saraf, and S. Yekhanin, "High-rate codes with sublinear-time decoding," *Journal of ACM*, vol. 61, no. 5, p. 28, 2014.

[11] S. Kopparty, "List-decoding multiplicity codes," *Theory of Computing*, vol. 11, no. 5, pp. 149–182, 2015.

[12] V. Guruswami and C. Wang, "Linear-algebraic list decoding for variants of reed-solomon codes," *IEEE Transactions on Information Theory*, vol. 59, no. 6, pp. 3257–3268, 2013.

[13] R. R. Nielsen, "List decoding of linear block codes," Ph.D. dissertation, Technical University of Denmark, 2001.

[14] Z. Dvir and S. Lovett, "Subspace evasive sets," in *Proceedings of the 44th Symposium on Theory of Computing Conference (STOC)*. ACM Press, 2012, pp. 351–358.

[15] S. Kopparty, O. Meir, N. Ron-Zewi, and S. Saraf, "High-rate locally correctable and locally testable codes with sub-polynomial query complexity," *Journal of ACM*, vol. 64, no. 2, pp. 11:1–11:42, 2017. [Online]. Available: http://doi.acm.org/10.1145/3051093

[16] S. Gopi, S. Kopparty, R. Oliveira, N. Ron-Zewi, and S. 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)*. SIAM, 2017, pp. 2073–2091.

[17] S. P. Vadhan, "Pseudorandomness," *Foundations and Trends in Theoretical Computer Science*, vol. 7, no. 1–3, pp. 1–336, 2012.

[18] V. Guruswami and P. 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)*. ACM Press, 2002, pp. 812–821.

[19] ——, "Linear time encodable and list decodable codes," in *STOC*, 2003, pp. 126–135.

[20] ——, "Linear-time list decoding in error-free settings," in *ICALP*, vol. 3142. Springer, 2004, pp. 695–707.

[21] ——, "Linear-time encodable/decodable codes with near-optimal rate," *IEEE Transactions on Information Theory*, vol. 51, no. 10, pp. 3393–3400, 2005.

[22] B. Hemenway, N. Ron-Zewi, and M. 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.

[23] P. Indyk, H. Q. Ngo, and A. Rudra, "Efficiently decodable non-adaptive group testing," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '10. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2010, pp. 1126–1142. [Online]. Available: http://portal.acm.org/citation.cfm?id=1873692

[24] H. Q. Ngo, E. Porat, and A. Rudra, "Efficiently Decodable Compressed Sensing by List-Recoverable Codes and Recursion," in *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), C. Dürr and T. Wilke, Eds., vol. 14. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 230–241. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2012/3401

[25] A. C. Gilbert, H. Q. Ngo, E. Porat, A. Rudra, and M. J. Strauss, "ℓ_2/ℓ_2-foreach sparse recovery with low risk," in *Automata, Languages, and Programming*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7965, pp. 461–472. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39206-1_39

[26] F. Parvaresh and A. Vardy, "Correcting errors beyond the guruswami-sudan radius in polynomial time," in *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*. IEEE, 2005, pp. 285–294.

[27] V. Guruswami and C. 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*. ACM, 2012, pp. 339–350.

[28] ——, "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)*. ACM Press, 2013, pp. 843–852.

[29] V. Guruswami and S. Kopparty, "Explicit subspace designs," *Combinatorica*, vol. 36, no. 2, pp. 161–185, 2016.

[30] N. Alon, J. Edmonds, and M. Luby, "Linear time erasure codes with nearly optimal recovery," in *proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 1995, pp. 512–519.

[31] B. Hemenway and M. Wootters, "Linear-time list recovery of high-rate expander codes," in *proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP)*, ser. LNCS, vol. 9134. Springer, 2015, pp. 701–712.

[32] O. Goldreich and L. A. Levin, "A hard-core predicate for all one-way functions," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. ACM, 1989, pp. 25–32.

[33] E. Kushilevitz and Y. Mansour, "Learning decision trees using the fourier spectrum," *SIAM Journal on Computing*, vol. 22, no. 6, pp. 1331–1348, 1993.

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

[35] A. Guo and S. Kopparty, "List-decoding algorithms for lifted codes," *IEEE Transactions on Information Theory*, vol. 62, no. 5, pp. 2719–2725, 2016.

[36] S. Saraf and S. Yekhanin, "Noisy interpolation of sparse polynomials, and applications," in *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*. IEEE, 2011, pp. 86–92.