Decoding Reed-Muller codes over product sets

John Kim[*] Swastik Kopparty[†]

September 29, 2017

Abstract

We give a polynomial time algorithm to decode multivariate polynomial codes of degree d up to half their minimum distance, when the evaluation points are an arbitrary product set S^m, for every $d < |S|$. Previously known algorithms could achieve this only if the set S had some very special algebraic structure, or if the degree d was significantly smaller than $|S|$. We also give a near-linear time randomized algorithm, based on tools from list-decoding, to decode these codes from nearly half their minimum distance, provided $d < (1 - \epsilon)|S|$ for constant $\epsilon > 0$.

Our result gives an m-dimensional generalization of the well known decoding algorithms for Reed-Solomon codes, and can be viewed as giving an algorithmic version of the Schwartz-Zippel lemma.

1 Introduction

Error-correcting codes based on polynomials have played an important role throughout the history of coding theory. The mathematical phenomenon underlying these codes is that distinct low-degree polynomials have different evaluations at many points. More recently, polynomial-based error-correcting codes have had a big impact on complexity theory, exploiting the intimate relation between polynomials and computation. Notable applications include PCPs, interactive proofs, polynomial identity testing and property testing.

Our main result is a decoding algorithm for multivariate polynomial codes over product sets. Let $\mathbb{F}$ be a field, $S \subseteq \mathbb{F}$, $d < |S|$ and $m \geq 1$. Let $\mathbb{F}[X_1, \ldots, X_m]$ denote the ring of m-variate polynomials with coefficients in $\mathbb{F}$. Let $\deg(P)$ denote the *total* degree of the polynomial P. Consider the code of all m-variate polynomials of total degree at most d, evaluated at all points of S^m:

$$\mathcal{C} = \{\langle P(\mathbf{a})\rangle_{\mathbf{a}\in S^m} \mid P(X_1, \ldots, X_m) \in \mathbb{F}[X_1, \ldots, X_m], \deg(P) \leq d\}.$$

When $m = 1$, this code is known as the Reed-Solomon code [3], and for $m > 1$ this code is a Reed-Muller code [1, 2]. The family of Reed-Muller codes also includes polynomial evaluation codes where the total degree d is larger than $|S|$, and the individual degree is capped to be at most $|S| - 1$. We do not consider the $d \geq |S|$ case in this paper.

The code $\mathcal{C}$ above is a subset of $\mathbb{F}^{S^m}$, which we view as the space of functions from S^m to $\mathbb{F}$. Given two functions $f, g : S^m \to \mathbb{F}$, we define their (Hamming) distance $\Delta(f, g) = |\{\mathbf{a} \in S^m \mid f(\mathbf{a}) \neq g(\mathbf{a})\}|$. To understand the error-correcting properties of $\mathcal{C}$, we recall the following well known lemma, often called the Schwartz-Zippel lemma:

Lemma 1.1. *Let $\mathbb{F}$ be a field, and let $P(X_1, \ldots, X_m)$ be a nonzero polynomial over $\mathbb{F}$ with degree at most d. Then for every $S \subseteq \mathbb{F}$,*

$$\Pr_{\mathbf{a}\in S^m}[P(\mathbf{a}) = 0] \leq \frac{d}{|S|}.$$

[*]Department of Mathematics, Rutgers University. jonykim@math.rutgers.edu Research supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1433187.

[†]Department of Mathematics & Department of Computer Science, Rutgers University. swastik@math.rutgers.edu Research supported in part by a Sloan Fellowship and NSF grant CCF-1253886.

This lemma implies that for any two polynomials P, Q of degree at most d, $\Delta(P, Q) \geq (1 - \frac{d}{|S|})|S|^m$. In other words the minimum distance of $\mathcal{C}$ is at least $(1 - \frac{d}{|S|})|S|^m$. It turns out that the minimum distance of $\mathcal{C}$ is in fact exactly $(1 - \frac{d}{|S|})|S|^m$, and we let $\Delta_{\mathcal{C}}$ denote this quantity.

For error-correcting purposes, if we are given a "received word" $r : S^m \to \mathbb{F}$ such that there exists a polynomial P of degree at most d with $\Delta(r, P) \leq \Delta_{\mathcal{C}}/2$, then we know that there is a unique such P. The problem that we consider in this paper, "decoding $\mathcal{C}$ upto half its minimum distance", is the algorithmic task of finding this P.

1.1 Our Results

There is a rich history with several deep algebraic ideas surrounding the problem of decoding multivariate polynomial codes. We first state our main results, and then discuss their relationship to various other known results.

Theorem 1.2 (Efficient decoding of multivariate polynomial codes upto half their minimum distance)**.** *Let $\mathbb{F}$ be a finite field, let S, d, m be as above. Let $N = n^m$ be the length of the corresponding Reed-Muller code, and let $\Delta_{\mathcal{C}} = (1 - \frac{d}{|S|})N$ be its minimum distance.*

There is an algorithm, which when given as input a function $r : S^m \to \mathbb{F}$, runs in time $\mathsf{poly}(N, \log|\mathbb{F}|)$ and finds the polynomial $P(X_1, \ldots, X_m) \in \mathbb{F}[X_1, \ldots, X_m]$ of degree at most d (if any) such that:

$$\Delta(r, P) < \Delta_{\mathcal{C}}/2.$$

As we will discuss below, previously known efficient decoding algorithms for these codes only worked for either (1) very algebraically special sets S, or (2) very low degrees d, or (3) decoded from a much smaller number of errors ($\approx \frac{1}{m+1}\Delta_{\mathcal{C}}$ instead of $\frac{1}{2}\Delta_C$).

Using several further ideas, we also show how to implement the above algorithm in near-linear time to decode upto almost half the minimum distance, provided d is not $(1 - o(1))|S|$.

Theorem 1.3 (Near-linear time decoding)**.** *Let $\mathbb{F}$ be a finite field, let S, d, m be as above. Let $N = n^m$ be the length of the corresponding Reed-Muller code, and let $\Delta_{\mathcal{C}} = (1 - \frac{d}{|S|})N$ be its minimum distance. Let $\delta_{\mathcal{C}} = (1 - \frac{d}{|S|})$. Assume $\delta_C > 0$, and m are constants.*

There is a randomized algorithm, which when given as input a function $r : S^m \to \mathbb{F}$, runs in time $N \cdot \mathsf{poly}(\log(N), \log(|\mathbb{F}|))$ and finds the polynomial $P(X_1, \ldots, X_m) \in \mathbb{F}[X_1, \ldots, X_m]$ of degree at most d (if any) with:

$$\Delta(r, P) < (1 - \Theta(\frac{1}{\sqrt{n}})) \cdot \Delta_{\mathcal{C}}/2.$$

Over the rational numbers, we get a version of Theorem 1.2 where the running time is $\mathsf{poly}(|S|^m, t)$, where t is the maximum bit-complexity of any point in S or in the image of r. This enables us to decode multivariate polynomial codes upto half the minimum distance in the natural special case where the evaluation set S equals $\{1, 2, \ldots, n\}$.

We also mention that decoding Reed-Muller codes over an arbitrary product set S^m appears as a subroutine in the local decoding algorithm for multiplicity codes [19] (see Section 4 on "Solving the noisy system"). Our results allow the local decoding algorithms there to run efficiently over all fields ([19] could only do this over fields of small characteristic, where algebraically special sets S are available).

1.2 Related work

There have been many works studying the decoding of multivariate polynomial codes, which prove (and improve) various special cases of our main theorem.

Reed-Solomon codes ($m = 1$): When $m = 1$, our problem is also known as the problem of decoding Reed-Solomon codes upto half their minimum distance. That this problem can be solved efficiently (and further, in near-linear time) is very classical, and a number of algorithms are known for this (Mattson-Solomon [7], Berlekamp-Massey [6], Berlekamp-Welch [14]). The underlying algorithmic ideas have subsequently had a tremendous impact on algebraic algorithms.

For Reed-Solomon codes, it is in fact known how to list-decode beyond half the minimum distance, upto the "Johnson radius" $\left(1 - \sqrt{1 - \frac{\Delta_C}{N}}\right) N$, by the important Guruswami-Sudan [8] algorithm. This has had numerous further applications in coding theory, complexity theory and pseudorandomness.

Special sets S: For very special sets S, it turns out that there are some algebraic ways to reduce the decoding of multivariate polynomial codes over S^m to the decoding of univariate polynomial codes. This kind of reduction is possible when $\mathbb{F}$ is a finite field $\mathbb{F}_q$ and S equals the whole field $\mathbb{F}_q$, or more generally when S equals an affine subspace over the prime subfield of $\mathbb{F}_q$.

When $S = \mathbb{F}_q$, then $S^m = \mathbb{F}_q^m$ and S^m can then be identified with the large field $\mathbb{F}_{q^m}$ in a natural $\mathbb{F}_q$-linear way (this understanding of Reed-Muller codes was discovered by [10]). This converts the multivariate setting into univariate setting, identifies the multivariate polynomial code as a subcode of the univariate polynomial code, and (somewhat miraculously), the minimum distance of the univariate polynomial code equals the minimum distance of the multivariate polynomial code. Thus the classical Reed-Solomon decoding algorithms can then be used, and this leads to an algorithm for the multivariate setting decoding upto half the minimum distance. In fact, Pellikaan-Wu [9] observed that this connection allows one to decode multivariate polynomial codes beyond half the minimum distance too, provided S is special in the above sense.

Another approach which works in the case of $S = \mathbb{F}_q$ is based on local decoding [5]. Here we use the fact that $S^m = \mathbb{F}_q^m$ contains many lines (not just the axis-parallel ones), and then use the univariate decoding algorithms to decode on those lines from $\frac{1-\frac{d}{q}}{2}N$ errors. This approach manages to decode multivariate polynomial codes with $S = \mathbb{F}_q$ from $(\frac{1}{2} - o(1))$ of the minimum distance. Again, this approach does not work for general S, since a general S^m usually contains only axis-parallel lines (while $\mathbb{F}_q^m$ has many more lines).

Low degree d: When the degree d of the multivariate polynomial code is significantly smaller than $|S|$, then a number of other list-decoding based methods come into play.

The powerful Reed-Muller list-decoding algorithm of Sudan [11] and its multiplicity-based generalization, based on $(m+1)$-variate interpolation and root-finding, can decode from $\left(1 - (\frac{d}{|S|})^{\frac{1}{m+1}}\right) N$ errors. With small degree $d = o(|S|)$ and $m = O(1)$, this decoding radius equals $(1 - o(1))N$! However when d is much larger (say $0.9 \cdot |S|$), then the number of errors decodable by this algorithm is around $\frac{1}{m+1} \cdot (1 - \frac{d}{|S|})N = \frac{1}{m+1} \cdot \Delta_C$.

Another approach comes from the list-decoding of tensor codes [12]. While the multivariate polynomial codes we are interested in are not tensor codes, they are subcodes of the code of polynomials with *individual degree* at most d. Using the algorithm of [12] for decoding tensor codes, we get an algorithm that can decode from a $(1-o(1))N$ errors when $d = o(|S|)$, but fails to approach a constant fraction of the minimum distance when d approaches $|S|$.

In light of all the above, to the best of our knowledge, for multivariate polynomial codes with $d > 0.9 \cdot |S|$ (i.e., with $\Delta_C < (0.1)N$), and S generic, the largest number of errors which could be corrected efficiently was about $\frac{1}{m+1}\Delta_C$. In particular, the correctable number of errors is a vanishing fraction of the minimum distance, as the number of variables m grows.

We thus believe it is worthwhile to investigate this problem, not only because of its basic nature, but also because of the many different powerful algebraic ideas that only give partial results towards it.

1.3 Overview of the decoding algorithm

We now give a brief overview of our decoding algorithms. Let us first discuss the bivariate ($m = 2$) case. Here we are given a received word $r : S^2 \to \mathbb{F}$ such that there exists a codeword $P(X, Y) \in \mathbb{F}[X, Y]$ of degree at most $d = (1 - \delta_C)|S|$ with $\Delta(P, r) < \frac{\delta_C}{2} \cdot |S|^2$. Our goal is to find $P(X, Y)$.

First some high-level strategy. An important role in our algorithm is played by the following observation: the restriction of a degree $\leq d$ bivariate polynomial $P(X, Y)$ to a vertical line (fixing $X = \alpha$) or a horizontal line (fixing $Y = \beta$) gives a degree $\leq d$ univariate polynomial. Perhaps an even more important role is played by the following disclaimer: *the previous observation does not characterize bivariate polynomials of degree d!* The set of functions $f : S^2 \to \mathbb{F}$ for which the horizontal restrictions and vertical restrictions are polynomials of degree $\leq d$ is the code of polynomials with *individual degree* at most d (this is the tensor Reed-Solomon code, with much smaller distance than the Reed-Muller code). For such a function f to be in the Reed-Muller code, the different univariate polynomials that appear as horizontal and vertical restrictions must be related in some way. The crux of our algorithm is to exploit these relations.

It will also help to recap the standard algorithm to decode tensor Reed-Solomon codes upto half their minimum distance (this scheme, known as Forney's Generalized Minimum Distance (GMD) decoding algorithm [15], actually works for general tensor codes). Suppose we are given a received word $r : S^2 \to \mathbb{F}$, and we want to find a polynomial $P(X, Y)$ with individual degrees at most d which is close to r. One first decodes, for every $\alpha \in S$, the row $r(\alpha, \cdot)$ to the nearest univariate polynomial of degree $\leq d$. One then takes the columns of this new received word (after having corrected the rows) and decodes each of them to the nearest degree $\leq d$ polynomial. As stated, this strategy only allows one to decode from $1/4$ the minimum distance. To go upto $1/2$ the minimum distance, the key point is to pass some "soft information" from the row decodings to the column decodings; the rows which were decoded from more errors are treated with lower confidence. This decodes the tensor Reed-Solomon code from $1/2$ the minimum distance fraction errors. Several ingredients from this algorithm will appear in our Reed-Muller decoding algorithm.

Now we return to the problem of decoding Reed-Muller codes. Let us write $P(X, Y)$ as a single variable polynomial in Y with coefficients in $\mathbb{F}[X]$: $P(X, Y) = \sum_{i=0}^{d} P_i(X)Y^{d-i}$, where $\deg(P_i) \leq i$. For each $\alpha \in S$, consider the restricted univariate polynomial $P(\alpha, Y)$. Since $\deg(P_0) = 0$, $P_0(\alpha)$ must be the same for each α. Thus all the polynomials $\langle P(\alpha, Y)\rangle_{\alpha \in S}$ have the same coefficient for Y^d. Similarly, the coefficients of Y^{d-i} in the polynomials $\langle P(\alpha, Y)\rangle_{\alpha \in S}$ fit a degree i polynomial.

As in the tensor Reed-Solomon case, our algorithm begins by decoding each row $r(\alpha, \cdot)$ to the nearest degree $\leq d$ univariate polynomial. Now, instead of trying to use these decoded row polynomials to recover $P(X, Y)$ in one shot, we aim lower and just try to recover $P_0(X)$. The advantage is that $P_0(X)$ is only a degree 0 polynomial, and is thus resilient to many more errors than a degree d polynomial. Armed with $P_0(X)$, we then proceed to find $P_1(X)$. The knowledge of $P_0(X)$ allows us to decode the rows $r(\alpha, \cdot)$ to a slightly larger radius; in turn this improved radius allows us to recover the degree 1 polynomial $P_1(X)$. At the i^{th} stage, we have already recovered $P_0(X)$, $P_1(X)$, $\ldots$, $P_{i-1}(X)$. Consider, for each $\alpha \in S$, the function $f_\alpha(Y) = r(\alpha, Y) - \sum_{j=0}^{i-1} P_j(\alpha)Y^{d-j}$. Our algorithm decodes $f_\alpha(Y)$ to the nearest degree $d - i$ polynomial: note that as i increases, we are decoding to a lower degree polynomial, and hence we are able to handle a larger fraction of errors. Define $h(\alpha)$ to be the coefficient of Y^{d-i} in the polynomial so obtained; this "should" equal the evaluation of the degree i polynomial $P_i(\alpha)$. So we next decode $h(\alpha)$ to the nearest degree i polynomial (using the appropriate soft information), and it turns out that this decoded polynomial must equal $P_i(X)$. By the time i reaches d, we would have recovered $P_0(X), P_1(X), \ldots, P_d(X)$, and hence all of $P(X, Y)$. Summarizing, the algorithm repeatedly decodes the rows $r(\alpha, \cdot)$, and at each stage it uses the relationship between the different univariate polynomial $P(\alpha, Y)$ to: (1) learn a little bit more about the polynomial $P(X, Y)$, and (2) increase the radius to which we can decode $r(\alpha, \cdot)$ in the next stage. This completes the description of the algorithm in the $m = 2$ case.

The case of general m is very similar, with only a small augmentation needed. Decoding m-variate polynomials turns out to reduce to decoding $m-1$-variate polynomials with soft information; thus in order to make a sustainable recursive algorithm, we aim a little higher and instead solve the more general problem of decoding multivariate polynomial codes with uncertainties (where each coordinate of the received word has an associated "confidence" level).

To implement the above algorithms in near-linear time, we use some tools from list-decoding. The main bottleneck in the running time is the requirement of having to decode the same row $r(\alpha, \cdot)$ multiple times to larger and larger radii (to lower and lower degree polynomials). To save on these decodings, we instead list-decode $r(\alpha, \cdot)$ to a large radius using a near-linear time list-decoder for Reed-Solomon codes; this reduces the number of required decodings of the same row from d to $O(1)$ (provided $d < (1 - \Omega(1))|S|$). For the $m = 2$ case this works fine, but for $m > 2$ case this faces a serious obstacle; in general it is impossible to efficiently list-decode Reed-Solomon codes *with uncertainties* beyond half the minimum distance of the code (the list size can be superpolynomial). We get around this using some technical ideas, based on speeding-up the decoding of Reed-Muller codes with uncertainties when the fraction of errors is significantly smaller than half the minimum distance. For details, see Section 6.

1.4 Organization of this paper

In Section 2, we cover the notion of weighted distance, which will be used in handling Reed-Solomon and Reed-Muller decoding with soft information on the reliability of the symbols in the encoding. In Section 3, we give a polynomial time algorithm for decoding bivariate Reed-Muller codes to half the minimum distance. We then generalize the algorithm to decode multivariate Reed-Muller codes in Section 4. In Section 5 and Section 6, we show that decoding Reed-Muller codes to almost half the minimum distance can be done in near-linear time by improving on the algorithms in Section 3 and Section 4. We conclude with some open problems.

2 Preliminaries

2.1 Weighted functions

At various stages of the decoding algorithm, we will need to deal with symbols and received words in which we have varying amounts of confidence. We now introduce some language to deal with such notions.

Let Σ denote an alphabet. A *weighted symbol* of Σ is simply an element of $\Sigma \times [0,1]$. In the weighted symbol (σ, u), we will be thinking of $u \in [0,1]$ as our uncertainty that σ is the symbol we should be talking about. Thus $u = 0$ represents absolute certainty about σ, while $u = 1$ represents having no confidence at all in the symbol σ.

For a weighted symbol (σ, u) and a symbol σ', we define their distance $\Delta((\sigma, u), \sigma')$ by:

$$\Delta((\sigma,u),\sigma') = \begin{cases} 1 - u/2 & \sigma \neq \sigma' \\ u/2 & \sigma = \sigma' \end{cases}$$

For a weighted function $w : T \to \Sigma \times [0,1]$, and a (conventional) function $f : T \to \Sigma$, we define their Hamming distance by

$$\Delta(w, f) = \sum_{t \in T} \Delta(w(t), f(t)).$$

The key inequality here is the triangle inequality.

Lemma 2.1 (Triangle inequality for weighted functions)**.** *Let $f, g : T \to \Sigma$ be functions, and let $w : T \to \Sigma \times [0,1]$ be a weighted function. Then:*

$$\Delta(w, f) + \Delta(w, g) \geq \Delta(f, g).$$

Proof. We will show that if $t \in T$ is such that $f(t) \neq g(t)$, then $\Delta(w(t), f(t)) + \Delta(w(t), g(t)) \geq 1$. This will clearly suffice to prove the lemma.

Let $w(t) = (\sigma, u)$. Suppose $f(t) = \sigma_1$ and $g(t) = \sigma_2$. Then either $\sigma \neq \sigma_1$ or $\sigma \neq \sigma_2$, or both. Thus either we have $\Delta(w(t), f(t)) + \Delta(w(t), g(t)) = (1 - u/2) + u/2$ or we have $\Delta(w(t), f(t)) + \Delta(w(t), g(t)) = u/2 + (1 - u/2)$, or we have $\Delta(w(t), f(t)) + \Delta(w(t), g(t)) = (1 - u/2) + (1 - u/2)$. In all cases, we have $\Delta(w(t), f(t)) + \Delta(w(t), g(t)) \geq 1$, as desired. □

The crucial property that this implies is the unique decodability up to half the minimum distance of a code for *weighted* received words.

Lemma 2.2. *Let $\mathcal{C} \subseteq \Sigma^T$ be a code with minimum distance Δ. Let $w : T \to \Sigma \times [0,1]$ be a weighted function. Then there is at most one $f \in \mathcal{C}$ satisfying*

$$\Delta(w, f) < \Delta/2.$$

2.2 Algorithms for decoding Reed-Solomon codes

We will need to use several known algorithms for decoding Reed-Solomon codes. We list these below:

- Fast Reed-Solomon decoding $\mathsf{FastRSDecoder}$: Given a received word $r : S \to \mathbb{F}_q$ and an integer $d < |S|$, the algorithm $\mathsf{FastRSDecoder}$ runs in time $O(|S|\mathsf{poly}(\log |S|, \log q))$ and finds the unique polynomial $P(X)$ (if any) of degree at most d such that $\Delta(P, r) < (|S| - d)/2$.

  There are many classical algorithms known for this, see [7].

- Weighted Reed-Solomon decoding $\mathsf{WeightedRSDecoder}$: Given a weighted received word $w : S \to \mathbb{F}_q \times [0, 1]$ and an integer $d < |S|$, the algorithm $\mathsf{WeightedRSDecoder}$ runs in time $O(|S|^2\mathsf{poly}(\log |S|, \log q))$ and finds the unique polynomial $P(X)$ (if any) of degree at most d such that $\Delta(P, w) < (|S| - d)/2$.

  This algorithm, due to Forney [15], is via a reduction to standard Reed-Solomon decoding, and is an important part of Forney's Generalized Minimum Distance decoding algorithm. For completeness, we describe it in the appendix.

- Fast weighted Reed-Solomon decoding $\mathsf{FastWeightedRSDecoder}$: Given a weighted received word $w : S \to \mathbb{F}_q \times [0, 1]$ and an integer $d < |S|$, the randomized algorithm $\mathsf{FastWeightedRSDecoder}$ runs in time $O(|S|\mathsf{poly}(\log |S|, \log q, \log \frac{1}{\gamma}))$ and finds (with probability at least $1 - \gamma$) the unique polynomial $P(X)$ (if any) of degree at most d such that

$$\Delta(P, w) < (|S| - d)/2 - \frac{10}{\sqrt{|S|}}.$$

  This algorithm is a randomized variant of Forney's algorithm $\mathsf{WeightedRSDecoder}$. The above performance guarantee is folklore, and follows from a simple second moment analysis. For completeness, we describe it in the appendix.

- Fast Reed-Solomon list-decoding $\mathsf{FastRSListDecoder}$: Given a received word $r : S \to \mathbb{F}_q$, an integer $d < |S|$, and a distance parameter Λ satisfying:

$$\Lambda < |S| - \sqrt{d|S|} + \epsilon|S|,$$

  the algorithm $\mathsf{FastRSListDecoder}$ runs in time $O(|S|\mathsf{poly}(\log |S|, \log q, \frac{1}{\epsilon}))$ and finds *all* polynomials $P(X)$ of degree at most d such that $\Delta(P, r) < \Lambda$. The Johnson bound further guarantees that there are at most $\mathsf{poly}(\frac{1}{\epsilon})$ such $P(X)$.

  This algorithm, which is a fast implementation of the fundamental Guruswami-Sudan algorithm [11, 8] for list-decoding Reed-Solomon codes, is due to Alekhnovich [17] (building on Roth and Ruckenstein [4]).

For the first three algorithms mentioned above, we assume that if no such polynomial exists, then the algorithm returns a special error symbol $\perp$.

3 Bivariate Reed-Muller Decoding

In this section, we give an algorithm for decoding bivariate Reed-Muller codes to half their minimum distance. Let $\mathbb{F}$ be a finite field, $S \subseteq \mathbb{F}$ with $|S| = n$, and let $d < n$. We are given a received word $r : S^2 \to \mathbb{F}$. Suppose that there is a codeword $C \in \mathbb{F}[X, Y]$ with $\deg(C) \leq d$, whose distance $\Delta(r, C)$ from the received word is at most half the minimum distance $n(n - d)/2$. The following result says that there is a polynomial time algorithm to find C.

Theorem 3.1. *Let $\mathbb{F}$ be a finite field and let $S \subseteq \mathbb{F}$ be a nonempty subset of size $|S| = n$.*

There is a $O(n^3 \operatorname{polylog}(n, |\mathbb{F}|))$ time algorithm, which given a received word $r : S^2 \to \mathbb{F}$, finds the unique polynomial (if any) $C \in \mathbb{F}[X, Y]$ with $\deg(C) \leq d$ such that

$$\Delta(r, C) < \frac{1}{2}\left(1 - \frac{d}{n}\right) n^2.$$

3.1 Outline of Algorithm

The general idea of the algorithm is to write $C(X,Y) = \sum_{i=0}^{d} P_i(X)Y^{d-i} \in \mathbb{F}[X][Y]$ as a polynomial in Y with coefficients as polynomials in $\mathbb{F}[X]$, and attempt to uncover the coefficients $P_i(X)$ one at a time.

We outline the first iteration of the algorithm, which uncovers the coefficient $P_0(X)$ of degree 0. View the encoded message as a matrix on $S \times S$, where the rows are indexed by $x \in S$ and the columns by $y \in S$. We first Reed-Solomon decode the rows $r(x,Y), x \in S$ to half the minimum distance $(n-d)/2$ and get the polynomial $G_x(Y)$. We then extract the coefficient, $\mathrm{Coeff}_{Y^d}(G_x)$, of Y^d from the polynomial $G_x(Y)$. This gives us guesses for what $P_0(x)$ is for $x \in S$. However, this isn't quite enough to determine $P_0(X)$. So we will also include some soft information which tells us how uncertain we are that the coefficient is correct. The uncertainty is a number in $[0,1]$ that is based on how far the decoded codeword $G_x(Y)$ is from the received word $r(x,Y)$. The farther apart, the higher the uncertainty. A natural choice for the uncertainty is simply the ratio of the distance $\Delta(G_x(Y), r(x,Y))$ to half the minimum distance $(n-d)/2$. Let $f : S \to F \times [0,1]$ be the function of guesses for $P_0(x)$ and their uncertainties. We then use a Reed-Solomon decoder with uncertainties to find the degree 0 polynomial that is closest to $f(X)$. This will give us $P_0(X)$. Finally, subtract $P_0(X)Y^d$ from $r(X,Y)$ and repeat to get the subsequent coefficients.

Algorithm 1 Decoding bivariate Reed-Muller codes

```
1: Input: $r : S^2 \to \mathbb{F}$.
2: for $i = 0, 1, \ldots, d$ do
3:     Define $r_i : S \times S \to \mathbb{F}$ by
           $$r_i(X,Y) = r(X,Y) - \sum_{j=0}^{i-1} Q_j(X)Y^{d-j}.$$
4:     for $x \in S$ do
5:         Define $r_{i,x} : S \to \mathbb{F}$ by
               $$r_{i,x}(Y) = r_i(x,Y).$$
6:         Define $G_x(Y) \in \mathbb{F}[Y]$ by
               $$G_x(Y) = \mathsf{FastRSDecoder}(r_{i,x}(Y), d-i).$$
7:         if $G_x \neq \perp$ then
8:             $\sigma_x = \mathrm{Coeff}_{Y^{d-i}}(G_x)$.
9:             $\delta_x = \Delta(r_{i,x}, G_x)$.
10:        else
11:            $\sigma_x = 0$
12:            $\delta_x = n$
13:        end if
14:    end for
15:    Define the weighted function $f_i : S \to \mathbb{F} \times [0,1]$ by
           $$f_i(x) = \left(\sigma_x, \min\{1, \frac{\delta_x}{(n-d+i)/2}\}\right).$$
16:    Define $Q_i : S \to \mathbb{F}$ by
           $$Q_i(X) = \mathsf{WeightedRSDecoder}(f_i(X), i).$$
17: end for
18: Output: $\sum_{i=0}^{d} Q_i(X)Y^{d-i}$.
```

3.2 Proof of Theorem 3.1

Proof. **Correctness of Algorithm** It suffices to show that $Q_i(X) = P_i(X)$ for $i = 0, 1, \ldots, d$, which we prove by induction. For this proof, the base case and inductive step can be handled by a single proof. We assume the inductive hypothesis that we have $Q_j(X) = P_j(X)$ for $j < i$. Note that the base case is $i = 0$ and in this case, we assume nothing.

It is enough to show $\Delta(f_i(X), P_i(X)) < \frac{n}{2}\left(1 - \frac{i}{n}\right)$. Then $P_i(x)$ is the unique polynomial within weighted distance $\frac{n}{2}\left(1 - \frac{i}{n}\right)$ of $f_i(X)$. So $\mathsf{WeightedRSDecoder}(f_i(X), i)$ will output $Q_i(X) = P_i(X)$.

We first show that $r_i(X,Y)$ is close to $C_i(X,Y) = \sum_{j=i}^{d} P_j(X)Y^{d-j}$. Observe that:

$$\begin{aligned}
&r_i(X,Y) - C_i(X,Y)\\
&= (r_i(X,Y) + \sum_{j=0}^{i-1} P_j(X)Y^{d-j}) - (C_i(X,Y) + \sum_{j=1}^{i-1} P_j(X)Y^{d-j}))\\
&= (r_i(X,Y) + \sum_{j=0}^{i-1} Q_j(X)Y^{d-j}) - C(X,Y)\\
&= r(X,Y) - C(X,Y).
\end{aligned}$$

Hence,

$$\Delta(r_i(X,Y), C_i(X,Y)) = \Delta(r(X,Y), C(X,Y)) < \frac{n^2}{2}\left(1 - \frac{d}{n}\right).$$

For each $x \in S$, define $C_{i,x}(Y) = C_i(x,Y)$. Define $\Delta_x = \Delta(r_{i,x}(Y), C_{i,x}(Y))$. Let $A = \{x \in S | G_x(Y) = C_{i,x}(Y)\}$ be the set of choices of x such that $G_x(Y) = \mathsf{FastRSDecoder}(r_{i,x}(Y), d-i)$ produces $C_{i,x}(Y)$.

Then, for $x \in A$, we have

$$\delta_x = \Delta(r_{i,x}(Y), G_x(Y)) = \Delta(r_{i,x}(Y), C_{i,x}(Y)) = \Delta_x.$$

And for $x \notin A$, we have that the degree $d-i$ polynomials $G_x \neq C_{i,x}$, and so

$$\delta_x = \Delta(r_{i,x}(Y), G_x(Y)) \geq n - d + i - \Delta(r_{i,x}(Y), C_{i,x}(Y)) = n - d + i - \Delta_x.$$

Note that this inequality holds even when $G_x = \perp$, because the algorithm explicitly sets $\delta_x = n$.

We now upper bound $\Delta(f_i(X), P_i(X))$:

$$\begin{aligned}
\Delta(f_i(X), P_i(X)) &\leq \sum_{x\in A} \frac{1}{2}\frac{\delta_x}{(n-d+i)/2} + \sum_{x\notin A}\left(1 - \frac{1}{2}\frac{\delta_x}{(n-d+i)/2}\right) && (1)\\
&\leq \sum_{x\in A} \frac{\Delta_x}{n-d+i} + \sum_{x\notin A}\left(1 - \frac{n-d+i-\Delta_x}{n-d+i}\right) && (2)\\
&= \sum_{x\in A} \frac{\Delta_x}{n-d+i} + \sum_{x\notin A} \frac{\Delta_x}{n-d+i} && (3)\\
&= \sum_{x\in S} \frac{\Delta_x}{n-d+i} && (4)\\
&= \frac{\Delta(r_i(X,Y), C_i(X,Y))}{n-d+i} && (5)\\
&< \frac{n^2}{2}\left(1 - \frac{d}{n}\right)\frac{1}{n-d+i} && (6)\\
&= \frac{n}{2} \cdot \frac{n-d}{n-d+i} && (7)
\end{aligned}$$

$$\leq \frac{n}{2} \cdot \frac{n-i}{n} \quad \text{since } i \leq d \tag{8}$$

$$= \frac{n}{2}\left(1 - \frac{i}{n}\right). \tag{9}$$

Runtime of Algorithm

We claim that the runtime of our algorithm is $O(n^3 \mathsf{poly}(\log(n), \log(|\mathbb{F}|))$. The algorithm has $d+1$ iterations. In each iteration, we update r_i, apply FastRSDecoder to n rows and apply WeightedRSDecoder a single time to get the leading coefficient. Since field operations take polylog($|\mathbb{F}|$)) time, at the cost of a polylog($|\mathbb{F}|$) factor, we can count field operations as running in unit time. As updating takes $O(n^2)$ time, FastRSDecoder takes $O(n \operatorname{polylog}(n))$ time, and WeightedRSDecodertakes $O(n^2 \operatorname{polylog}(n))$ time, we get $O(n^2 \operatorname{polylog}(n))$ for each iteration. $d+1$ iterations gives a total runtime of $O(dn^2 \operatorname{polylog}(n)) < O(n^3 \operatorname{polylog}(n))$.

□

4 Reed-Muller Decoding for General m

We now generalize the algorithm for decoding bivariate Reed-Muller codes to handle Reed-Muller codes of any number of variables. As before, we write the codeword as a polynomial in one of the variables and attempt to uncover its coefficients one at a time. Interestingly, this leads us to a Reed-Muller decoding on one fewer variable, but with uncertainties. This lends itself nicely to an inductive approach on the number of variables. However, the generalization requires us to be able to decode Reed-Muller codes with uncertainties. This leads us to our main theorem:

Theorem 4.1. *Let $\mathbb{F}$ be a finite field and let $S \subseteq \mathbb{F}$ be a nonempty subset of size $|S| = n$.*

There is a $O(n^{m+2} \operatorname{polylog}(n, |\mathbb{F}|))$ time algorithm, which given a weighted received word $w : S^m \to \mathbb{F} \times [0,1]$, finds the unique polynomial (if any) $C \in \mathbb{F}[X_1, \ldots, X_m]$ with $\deg(C) \leq d$ such that

$$\Delta(w, C) < \frac{1}{2}\left(1 - \frac{d}{n}\right) n^m.$$

To decode a pure received word (without uncertainties), we may just set all the initial uncertainties to 0.

Note that in the bivariate case, the algorithm given by the above theorem is slower than the one given by Theorem 3.1. This is because the above theorem can also handle uncertanties.

Proof. The proof is by induction on the number of variables, and closely mirrors the proof of the bivariate case.

Base Case

We are given a received word with uncertainties $w : S \to \mathbb{F} \times [0,1]$ and asked to find the unique polynomial $C \in \mathbb{F}[X]$ with $\deg(C) \leq d$ within weighted distance $\frac{n-d}{2}$ of w. This is just Reed-Solomon decoding with uncertainty, which can be done in time $O(n^2 \operatorname{polylog}(n))$ via algorithm WeightedRSDecoder.

Inductive Step

Assume that the result holds for $(m-1)$-variable Reed-Muller codes. That is, assume we have access to an algorithm WeightedRMDecoder(w, d) which takes as input a received word with uncertainties $w : S^{m-1} \to \mathbb{F} \times [0,1]$, and outputs the unique polynomial of degree at most d (if any) within weighted distance $\frac{n^{m-1}}{2}\left(1 - \frac{d}{n}\right)$ from w. We want to produce an algorithm for m variables.

We are given $w : S^m \to \mathbb{F} \times [0,1]$. View w as a map from $S^{m-1} \times S \to \mathbb{F} \times [0,1]$, and write

$$w(\boldsymbol{X}, Y) = (r(\boldsymbol{X}, Y), u(\boldsymbol{X}, Y)),$$

where $X_1, \ldots, X_{m-1}, Y$ are the m-variables, and $\boldsymbol{X} = (X_1, \ldots, X_{m-1})$. Suppose that there exists a polynomial $C \in \mathbb{F}[\boldsymbol{X}, Y]$ with $\deg(C) \leq d$ such that

$$\Delta(w, C) < \frac{n^m}{2}\left(1 - \frac{d}{n}\right).$$

View C as a polynomial in Y with coefficients in $\mathbb{F}[\boldsymbol{X}]$, $C(\boldsymbol{X},Y) = \sum_{i=0}^{d} P_i(\boldsymbol{X})Y^{d-i}$.

The general strategy of the algorithm is to determine the P_i's inductively by performing $d+1$ iterations from $i = 0$ to $i = d$, and recovering $P_i(\boldsymbol{X})$ at the i-th iteration. We give an informal overview next, and then give the algorithm itself.

For the i-th iteration, consider the word $w_i : S^m \to \mathbb{F} \times [0,1]$ given by:

$$w_i(\boldsymbol{X},Y) = \left(r(\boldsymbol{X},Y) - \sum_{j=0}^{i-1} P_j(\boldsymbol{X})Y^{d-j}, u(\boldsymbol{X},Y) \right).$$

Since w is close to $\sum_{j=0}^{d} P_j(\boldsymbol{X})Y^{d-j}$, w_i will be close to $C_i = \sum_{j=i}^{d} P_j(\boldsymbol{X})Y^{d-j}$. Our goal is to find $P_i(\boldsymbol{X})$, the coefficient of Y^{d-i} when C_i when viewed as a polynomial in Y. For each $\boldsymbol{x} \in S^{m-1}$, we decode the Reed-Solomon code with uncertainties given by $w_i(\boldsymbol{x},Y)$ and extract the coefficient of Y^{d-i} along with how uncertain we are about the correctness of this coefficient. This gives us a guess for the value $P_i(\boldsymbol{x})$ and our uncertainty for this guess. We construct the function $f_i : S^{m-1} \to F \times [0,1]$ of guesses for P_i with their uncertainties. We then apply the algorithm given by induction hypothesis to f_i, and this gives us P_i.

Algorithm 2 WeightedRMDecoder: Decoding general Reed-Muller codes with uncertainties

```
1: Input: $w : S^m \to \mathbb{F} \times [0,1]$.
2: Define $r : S^m \to \mathbb{F}$ and $u : S^m \to [0,1]$ by:
       $$w(\boldsymbol{X}, Y) = (r(\boldsymbol{X}, Y), u(\boldsymbol{X}, Y)).$$
   .
3: for $i = 0, 1, \ldots, d$ do
4:     Define $w_i : S^m \times S \to \mathbb{F} \times [0,1]$ by
           $$w_i(\boldsymbol{X}, Y) = \left( r(\boldsymbol{X}, Y) - \sum_{j=0}^{i-1} Q_j(\boldsymbol{X}) Y^{d-j}, u(\boldsymbol{X}, Y) \right).$$
5:     for $\boldsymbol{x} \in S^{m-1}$ do
6:         Define $w_{i,\boldsymbol{x}} : S \to \mathbb{F} \times [0,1]$ by
               $$w_{i,\boldsymbol{x}}(Y) = w_i(\boldsymbol{x}, Y).$$
7:         Define $G_{\boldsymbol{x}}(Y) \in \mathbb{F}[Y]$ by
               $$G_{\boldsymbol{x}}(Y) = \mathsf{WeightedRSDecoder}(w_{i,\boldsymbol{x}}(Y), d - i).$$
8:         if $G_{\boldsymbol{x}} \neq \perp$ then
9:             $\sigma_{\boldsymbol{x}} = \mathrm{Coeff}_{Y^{d-i}}(G_{\boldsymbol{x}})$.
10:            $\delta_{\boldsymbol{x}} = \Delta(w_{i,\boldsymbol{x}}, G_{\boldsymbol{x}})$.
11:        else
12:            $\sigma_{\boldsymbol{x}} = 0$.
13:            $\delta_{\boldsymbol{x}} = n$.
14:        end if
15:    end for
16:    Define the weighted function $f_i : S^m \to \mathbb{F} \times [0,1]$ by
           $$f_i(\boldsymbol{x}) = \left( \sigma_{\boldsymbol{x}}, \min\{1, \frac{\delta_{\boldsymbol{x}}}{(n-d+i)/2}\} \right).$$
17:    Construct the polynomial $Q_i(\boldsymbol{X}) \in \mathbb{F}[X_1, \ldots, X_{m-1}]$ by
           $$Q_i(\boldsymbol{X}) = \mathsf{WeightedRMDecoder}(f_i(\boldsymbol{X}), i).$$
18: end for
19: Output: $\sum_{i=0}^{d} Q_i(\boldsymbol{X}) Y^{d-i}$.
```

Correctness of Algorithm

Suppose there is a polynomial $C(\boldsymbol{X}, Y) = \sum_{i=0}^{d} P_i(\boldsymbol{X}) Y^{d-i}$ such that

$$\Delta(w, C) < \frac{n^m}{2} \left(1 - \frac{d}{n} \right).$$

We will show by induction that the i-th iteration of the algorithm produces $Q_i(\boldsymbol{X}) = P_i(\boldsymbol{X})$. For this proof, the base case and inductive step can be handled by a single proof. We assume the inductive hypothesis that we have $Q_j(\boldsymbol{X}) = P_j(\boldsymbol{X})$ for $j < i$. Note that the base case is $i = 0$ and in this case, we assume nothing.

It is enough to show $\Delta(f_i(\boldsymbol{X}), P_i(\boldsymbol{X})) < \frac{n^{m-1}}{2} \left(1 - \frac{i}{n} \right)$. Then $P_i(\boldsymbol{X})$ is the unique polynomial within weighted distance $\frac{n^{m-1}}{2}\left(1-\frac{i}{n}\right)$ of $f_i(\boldsymbol{X})$. So by the induction hypothesis for $(m-1)$-variate Reed-Muller codes, $\mathsf{WeightedRMDecoder}(f_i(\boldsymbol{X}), m, i)$ will output $Q_i(\boldsymbol{X}) = P_i(\boldsymbol{X})$.

We first show that w_i is close to $C_i(\boldsymbol{X}, Y) = \sum_{j=i}^{d} P_j(\boldsymbol{X})Y^{d-j}$. Define $r_i : S^m \to \mathbb{F}$ and $u_i : S^m \to [0,1]$ by:

$$w_i(\boldsymbol{x}, y) = (r_i(\boldsymbol{x}, y), u_i(\boldsymbol{x}, y)).$$

Observe that:

$$\begin{aligned}
&r_i(\boldsymbol{X}, Y) - C_i(\boldsymbol{X}, Y)\\
&= (r_i(\boldsymbol{X}, Y) + \sum_{j=0}^{i-1} P_j(\boldsymbol{X})Y^{d-j}) - (C_i(\boldsymbol{X}, Y) + \sum_{j=0}^{i-1} P_j(\boldsymbol{X})Y^{d-j}))\\
&= (r_i(\boldsymbol{X}, Y) + \sum_{j=0}^{i-1} Q_j(\boldsymbol{X})Y^{d-j}) - C(\boldsymbol{X}, Y)\\
&= r(\boldsymbol{X}, Y) - C(\boldsymbol{X}, Y).
\end{aligned}$$

Also observe that $u_i(\boldsymbol{x}, y) = u(\boldsymbol{x}, y)$. Hence,

$$\Delta(w_i, C_i) = \Delta(w, C) < \frac{n^m}{2}\left(1 - \frac{d}{n}\right).$$

For each $\boldsymbol{x} \in S^{m-1}$, define $C_{i,\boldsymbol{x}}(Y) = C_i(\boldsymbol{x}, Y)$. Define $\Delta_{\boldsymbol{x}} = \Delta(w_{i,\boldsymbol{x}}(Y), C_{i,\boldsymbol{x}}(Y))$. Let $A = \{\boldsymbol{x} \in S^{m-1} | G_{\boldsymbol{x}}(Y) = C_{i,\boldsymbol{x}}(Y)\}$ be the set of choices of $\boldsymbol{x}$ such that $G_{\boldsymbol{x}}(Y) = \mathsf{FastRSDecoder}(w_{i,\boldsymbol{x}}(Y), d-i)$ produces $C_{i,\boldsymbol{x}}(Y)$.

Then, for $\boldsymbol{x} \in A$, we have

$$\delta_{\boldsymbol{x}} = \Delta(w_{i,\boldsymbol{x}}(Y), G_{\boldsymbol{x}}(Y)) = \Delta(w_{i,\boldsymbol{x}}(Y), C_{i,\boldsymbol{x}}(Y)) = \Delta_{\boldsymbol{x}}.$$

And for $\boldsymbol{x} \notin A$, we have $G_{\boldsymbol{x}} \neq C_{i,\boldsymbol{x}}$ and both are low-degree polynomials, so $\Delta(G_{\boldsymbol{x}}, C_{i,\boldsymbol{x}}) \geq n - d + i$. By the triangle inequality, we have:

$$\delta_{\boldsymbol{x}} = \Delta(w_{i,\boldsymbol{x}}(Y), G_{\boldsymbol{x}}(Y)) \geq n - d + i - \Delta(w_{i,\boldsymbol{x}}(Y), C_{i,\boldsymbol{x}}(Y)) = n - d + i - \Delta_{\boldsymbol{x}}.$$

Note that this inequality holds even when $G_{\boldsymbol{x}} = \perp$, because the algorithm explicitly sets $\delta_{\boldsymbol{x}} = n$.

We now upper bound $\Delta(f_i(\boldsymbol{X}), P_i(\boldsymbol{X}))$:

$$\begin{aligned}
\Delta(f_i(\boldsymbol{X}), P_i(\boldsymbol{X})) &\leq \sum_{\boldsymbol{x}\in A} \frac{1}{2}\frac{\delta_{\boldsymbol{x}}}{(n-d+i)/2} + \sum_{\boldsymbol{x}\notin A}\left(1 - \frac{1}{2}\frac{\delta_{\boldsymbol{x}}}{(n-d+i)/2}\right)\\
&\leq \sum_{\boldsymbol{x}\in A} \frac{\Delta_{\boldsymbol{x}}}{n-d+i} + \sum_{\boldsymbol{x}\notin A}\left(1 - \frac{n-d+i-\Delta_{\boldsymbol{x}}}{n-d+i}\right)\\
&= \sum_{\boldsymbol{x}\in A} \frac{\Delta_{\boldsymbol{x}}}{n-d+i} + \sum_{\boldsymbol{x}\notin A} \frac{\Delta_{\boldsymbol{x}}}{n-d+i}\\
&= \sum_{\boldsymbol{x}\in S^{m-1}} \frac{\Delta_{\boldsymbol{x}}}{n-d+i}\\
&= \frac{\Delta(w_i(\boldsymbol{X}, Y), C_i(\boldsymbol{X}, Y))}{n-d+i}\\
&< \frac{n^m}{2}\left(1 - \frac{d}{n}\right)\frac{1}{n-d+i}\\
&= \frac{n^{m-1}}{2} \cdot \frac{n-d}{n-d+i}
\end{aligned}$$

$$\begin{aligned} &\leq \frac{n^{m-1}}{2} \cdot \frac{n-i}{n} \\ &= \frac{n^{m-1}}{2} \left(1 - \frac{i}{n}\right). \end{aligned}$$

This implies that $P_i = Q_i$, as desired.

Runtime of Algorithm

We now show that the runtime of our m-variate Reed-Muller decoder is $O(n^{m+2} \operatorname{polylog}(n) \operatorname{polylog}(|\mathbb{F}|))$. We again proceed by induction on m. In the base case of $m = 1$, we simply run the Reed-Solomon decoder with uncertainties, which runs in $O(n^2 \operatorname{polylog}(n))$ time (which is even a factor n better than what we need to show). Now suppose the $(m-1)$-variate Reed-Muller decoder runs in time $O(n^{m+1} \operatorname{polylog}(n))$. We need to show that the m-variate Reed-Muller decoder runs in time $O(n^{m+2} \operatorname{polylog}(n))$.

The algorithm makes $d+1$ iterations. In each iteration, we first (1) perform n^{m-1} Reed-Solomon decodings with uncertainties and extract the leading coefficient along with its uncertainty for each one, and then (2) do an $(m-1)$-variate Reed-Muller decoding with uncertainties . Each Reed-Solomon decoding takes $O(n^2 \operatorname{polylog}(n))$ time, while computing an uncertainty of a leading coefficient takes $O(n \operatorname{polylog}(n))$. So in this step, we have cumulative runtime $O(n^{m-1} \cdot n^2 \cdot \operatorname{polylog}(n)) = O(n^{m+1} \operatorname{polylog}(n))$. The $(m-1)$-variate Reed-Muller decoding with uncertainties takes $O(n^{m+1} \operatorname{polylog}(n))$ time by our induction hypothesis.

This makes the total runtime for all iterations $O(dn^{m+1} \operatorname{polylog}(n)) \leq O(n^{m+2} \operatorname{polylog}(n))$, as desired. □

5 Near-Linear Time Decoding in the Bivariate Case

In this section, we present our near-linear time, randomized decoding algorithm for bivariate Reed-Muller codes.

Theorem 5.1. *Let $\alpha \in (0,1)$ be a constant. Let $\mathbb{F}$ be a finite field and let $S \subseteq \mathbb{F}$ be a nonempty subset of size $|S| = n$. Let $d = \alpha n$.*

There is a $O(n^2 \operatorname{polylog}(n, |\mathbb{F}|))$ time randomized algorithm which, given a received word $r : S^2 \to \mathbb{F}$, finds the unique polynomial (if any) $C \in \mathbb{F}[X,Y]$ with $\deg(C) \leq d$ such that

$$\Delta(r, C) < \frac{1}{2}\left(1 - \alpha - \frac{10}{\sqrt{n}}\right) n^2.$$

5.1 Outline of Improved Algorithm

Recall that the decoding algorithm we presented in the previous sections make $d+1$ iterations, revealing a single coefficient of the nearest codeword during one iteration. In a given iteration, the bivariate algorithm decodes each row of $r_i(X,Y)$ to the nearest polynomial of degree $d-i$, extracting the coefficient of Y^{d-i} and its uncertainty. Then the algorithm Reed-Solomon decodes (with uncertainties) to get the leading coefficient of $C(X,Y)$, when viewed as a polynomial in Y.

The runtime of this algorithm is $O(n^3 \operatorname{polylog}(n))$. Each iteration has n Reed-Solomon decodings and a single Reed-Solomon decoding with uncertainties. As Reed-Solomon decoding takes $O(n \operatorname{polylog}(n))$ time and Reed-Solomon decoding with uncertainties takes $O(n^2 \operatorname{polylog}(n))$ time, we get a runtime of $O(n^3 \operatorname{polylog}(n))$ with $d+1$ iterations. To achieve near-linear time, we need to shave off a factor of n on both the number of Reed-Solomon decodings and the runtime of Reed-Solomon decoding with uncertainties.

To save on the number of Reed-Solomon decodings, we will instead list decode beyond half the minimum distance (using the near-linear time Reed-Solomon list-decoder FastRSListDecoder), and show that the list we get is both small and essentially contains all of the decoded polynomials we require for $\Omega(n)$ iterations of i (thus we can reduce the number of decodings by a factor $\Omega(n)$). So we will do $O(n)$ Reed-Solomon list-decodings total instead of $O(n^2)$ Reed-Solomon unique decodings to half the minimum distance.

To save on the runtime of Reed-Solomon decoding with uncertainties, we will use the faster probabilistic variant FastWeightedRSDecoder of Forney's generalized minimum distance decoding algorithm, which runs in near-linear time, but reduces the decoding radius from $1/2$ the minimum distance to $1/2 - o(1)$ of the minimum distance.

5.2 Ingredients of the algorithm

Reducing the Number of Decodings

To reduce the number of decodings, we will list decode past half the minimum distance. Let $r_{i,x} : S \to \mathbb{F}$ be a received word for a Reed-Solomon code $\mathcal{C}_i$ of degree at most $d_i = d - i$. Let t be the radius to which we list decode, and let $L_{i,x} = \{C \in \mathcal{C}_i | \Delta(C, r_{i,x}) < t\}$ be the list of codewords within distance t of $r_{i,x}$. The radius to which we can decode while maintaining a polynomial-size list is given by the Johnson bound:

$$n(1 - \sqrt{1 - \delta_i}),$$

where $\delta_i = 1 - \frac{d-i}{n} > 1 - \frac{d}{n} = 1 - \alpha$ is the relative distance of the code. By Taylor approximating the square root, we see that the Johnson bound exceeds half the minimum distance by $\Omega(n)$:

$$\begin{aligned} n(1 - \sqrt{1 - \delta_i}) &> n(1 - (1 - (\delta_i/2 + \delta_i^2/8 + 3\delta_i^3/16))) \\ &= n(\delta_i/2 + (1 - \alpha)^2/8 + 3(1 - \alpha)^3/16) \\ &= (n - d + i)/2 + ((1 - \alpha)^2/8)n + \epsilon n, \end{aligned}$$

where $\epsilon = 3(1-\alpha)^3/16$ is a positive constant. By the quantitative version of the Johnson bound (such as the one by Cassuto and Bruck [16]), we see that if we set the list decoding radius $t = (n-d+i)/2 + ((1-\alpha)^2/8)n$, then the size of the list $|L_{i,x}| < \frac{1}{\epsilon}$ is constant. So the list decoding radius exceeds half the minimum distance by $\Omega(n)$, and the list size is constant. The algorithm $\mathsf{FastRSListDecoder}$ (due to Alekhnovich [17]) can find this list $L_{i,x}$ in time $(1/\alpha)^{O(1)} n \log^2 n \log\log(n) = O(n \,\mathrm{polylog}(n))$. More precisely, we will let $\mathsf{FastRSListDecoder}(r, d, \Lambda)$ denote this algorithm, that outputs a list of all polynomials P of degree at most d within distance Λ of the received word r.

Faster Reed-Solomon Decoding with Uncertainties

We will use a faster randomized algorithm for decoding Reed-Solomon codes with uncertainties. We will refer to this algorithm as $\mathsf{FastWeightedRSDecoder}(f, d)$, where $f : S \to \mathbb{F} \times [0, 1]$ is a received word with uncertainties, and d is the degree of the code. More precisely, $\mathsf{FastWeightedRSDecoder}(f, d, \gamma)$ will run in time $n\mathsf{poly}(\log(n), \log \frac{1}{\gamma})$ and output the codeword within distance $(n-d-10\sqrt{n})/2$ (if any) with probability at least $1 - \gamma$. In our Reed-Muller decoding algorithms, we will invoke $\mathsf{FastWeightedRSDecoder}$ with γ set to n^{-c} for some large c. This will allow us to say, by a union bound, that the probability that any of the invocations return a wrong answer is at most $o(1)$. We may thus assume (for the sake of analysis of the correctness of the algorithms) that $\mathsf{FastWeightedRSDecoder}$ always correctly decodes Reed-Solomon codes (with uncertainties) up to radius $(n - d - 10\sqrt{n})/2$.

Algorithm 3 Fast decoding of bivariate Reed-Muller codes

```
1: Input: $r : S^2 \to \mathbb{F}$.
2: Let $c = ((1-\alpha)^2/8)$.
3: Let $e = \lfloor 2cn \rfloor$.
4: for $i = 0, 1, \ldots, d$ do
5:     Define $r_i : S \times S \to \mathbb{F}$ by
           $$r_i(X,Y) = r(X,Y) - \sum_{\ell=0}^{i-1} Q_\ell(X) Y^{d-\ell}.$$
6:     for $x \in S$ do
7:         Define $r_{i,x} : S \to \mathbb{F}$ by
               $$r_{i,x}(Y) = r_i(x,Y).$$
8:         if $(e+1) \mid i$ then
9:             Let $j = \frac{i}{e+1}$
10:            Let $t_j = \frac{n-d+i}{2} + cn$.
11:            Define $L_{i,x} = \mathsf{FastRSListDecoder}(r_{i,x}, d-i, t_j)$.
12:        else
13:            Define $L_{i,x} = \{R(Y) - Q_{i-1}(x)Y^{d-i+1} \mid R(Y) \in L_{i-1,x}, \mathrm{Coeff}_{Y^{d-i+1}}(R) = Q_{i-1}(x)\}$.
14:        end if
15:        Define $G_x(Y)$ to be the unique element (if any) of the list $L_{i,x}$ with degree at most $d-i$ and:
               $$\Delta(G_x, r_{i,x}) < \frac{n-d+i}{2}$$
16:        if $G_x \neq \perp$: then
17:            $\sigma_x = \mathrm{Coeff}_{Y^{d-i}}(G_x)$.
18:            $\delta_x = \Delta(G_x, r_{i,x})$.
19:        else
20:            $\sigma_x = 0$
21:            $\delta_x = n$
22:        end if
23:    end for
24:    Define the weighted function $f_i : S \to \mathbb{F} \times [0,1]$ by
           $$f_i(x) = \left(\sigma_x, \min\{1, \frac{\delta_x}{(n-d+i)/2}\}\right).$$
25:    Define $Q_i : S \to \mathbb{F}$ by
           $$Q_i(X) = \mathsf{FastWeightedRSDecoder}(f_i(X), i, n^{-5}).$$
26: end for
27: Output: $\sum_{i=0}^{d} Q_i(\boldsymbol{X}) Y^{d-i}$.
```

Proof of Theorem 5.1. We claim that Algorithm 3 has the required properties. We first analyze correctness, and then we analyze the running time.

Correctness of Algorithm

Let $r : S \times S \to \mathbb{F}$ be a received word, and let $C(X,Y)$ be a polynomial of degree at most d such that

$$\Delta(r, C) < \frac{1}{2}\left(1 - \alpha - \frac{10}{\sqrt{n}}\right) n^2.$$

Let $C(X,Y) = \sum_{i=0}^{d} P_i(X)Y^{d-i}$. We will show that the Q_i produced by the algorithm satisfy $Q_i(X) = P_i(X)$ for $i = 0, \dots, d$ (with high probability over the randomness of FastWeightedRSDecoder).

As commented earlier, we will assume that every invocation of FastWeightedRSDecoder is successful: this is allowed since we have tuned the parameter γ to be a suitably small polynomial function of n.

By induction on i, we will show that all the following statements hold for each $i \le d$.

1. $r_{i,x}$ in Algorithm 1 and Algorithm 3 are the same for all $x \in S$,
2. For $j = \lfloor \frac{i}{e+1} \rfloor$, $L_{i,x}$ contains all polynomials $R'(Y)$ of degree at most $d-i$ such that
$$\Delta(R', r_{i,x}) \le t_j.$$
3. $Q_i(X) = P_i(X)$.

The analysis will follow the analysis of Algorithm 1 very closely.

For $i = 0$, Item 1 is trivial (since in both Algorithm 1 and Algorithm 3, $r_{0,x}(y) = r(x,y)$). For other i, Item 1 follows immediately from the definition of $r_{i,x}$ and from Item 3 of the induction hypothesis for $i-1$.

For every i with $(e+1) \mid i$ (and in particular for $i = 0$), Item 2 follows from Step 11. For other i, $L_{i,x}$ is obtained from $L_{i-1,x}$ in Step 13 of the algorithm. Suppose $R'(Y)$ is a polynomial of degree at most $d-i$ such that $\Delta(R', r_{i,x}) \le t_j$. Consider the polynomial $R(Y) = R'(Y) - Q_{i-1}(x)Y^{d-i+1}$ which is of degree at most $d-i+1$. We will show that $R(Y) \in L_{i-1,x}$, and then Step 13 of the algorithm will put $R'(Y)$ into $L_{i,x}$. By construction,

$$\begin{aligned}
\Delta(R, r_{i-1,x}) &= \Delta(R'(Y) - Q_{i-1}(x)Y^{d-i+1}, r_{i-1,x}) \\
&= \Delta(R'(Y), r_{i-1,x} + Q_{i-1}(x)Y^{d-i+1}) \\
&= \Delta(R', r_{i,x}) \\
&\le t_j.
\end{aligned}$$

This, along with the induction hypothesis for $i-1$, implies that $R(Y) \in L_{i-1,x})$, as desired.

Now we discuss Item 3. Let $j = \lfloor \frac{i}{e+1} \rfloor$. If Item 2 holds for some i, then using the fact that

$$t_j = \frac{n-d+j\cdot(e+1)}{2} + cn = \frac{n-d+j\cdot(e+1)+2cn}{2} > \frac{n-d+j\cdot(e+1)+e}{2} \ge \frac{n-d+i}{2},$$

we conclude that $G_x(Y)$ is the unique polynomial in $\mathbb{F}[Y]$ (if any) of degree at most $d-i$ with $\Delta(G_x, r_{x,i}) < \frac{n-d+i}{2}$.

Thus the G_x produced in Step 15 of Algorithm 3 is the same as the corresponding G_x in Algorithm 1. This implies that the function f_i in Algorithm 1 and Algorithm 3 are the same.

We will now show that

$$\Delta(f_i, P_i) < \frac{n-i-10\sqrt{n}}{2}. \tag{10}$$

By induction, we may assume that we have already shown that $P_\ell = Q_\ell$ for each $\ell \in \{0, 1, \dots, i-1\}$. Letting:

$$C_i(X,Y) = \sum_{j=i}^{d} P_j(X)Y^{d-j},$$

we get that $\Delta(r_i, C_i) = \Delta(r, C)$. Following the same calculation as in (5), we have that:

$$\begin{aligned}
\Delta(f_i, P_i) &< \frac{\Delta(r_i, C_i)}{n-d+i} \\
&= \frac{\Delta(r, C)}{n-d+i} \\
&\le \frac{\frac{1}{2}(1-\alpha-\frac{10}{\sqrt{n}})n^2}{n(1-\alpha)+i}
\end{aligned}$$

$$
\begin{aligned}
&\leq \frac{1}{2}\frac{1-\alpha-\frac{10}{\sqrt{n}}}{1-\alpha+\frac{i}{n}}\cdot n \\
&\leq \frac{1}{2}\left(1-\frac{i}{n}-\frac{10}{\sqrt{n}}\right)\cdot n \\
&= \frac{1}{2}(n-i-10\sqrt{n}),
\end{aligned}
$$

as required[1].

Now, Inequality (10) implies that the invocation of FastWeightedRSDecoder in Step 25 will return P_i, and thus $Q_i = P_i$. This completes the proof of the inductive step.

Now that we have shown that for all i we have $Q_i = P_i$, we can conclude that the output of the algorithm is indeed $C(X, Y)$, as desired.

Analysis of Runtime of Bivariate Reed-Muller Decoder

Algorithm 3 invokes FastRSListDecoder$\frac{d}{e+1}n \leq \frac{\alpha}{2c}n \leq \frac{4\alpha}{(1-\alpha)^2}n$ times, and invokes FastWeightedRSDecoder$d = \alpha n$ times. As both of these algorithms run in $O(n\,\mathrm{polylog}(n))$ time, the total runtime of Algorithm 3 algorithm is $O(n^2\,\mathrm{polylog}(n, |\mathbb{F}|))$ (after accounting for the remaining operations), as desired.

□

6 Near-Linear Time Decoding in the General Case

In this section, we give a more involved variation of the previous algorithm to get a near-linear time, randomized algorithm for decoding Reed-Muller codes up to half the minimum distance over any number of variables. It is worth noting that we do not know how to decode Reed-Muller codes *with uncertainties* up to half the minimum distance in near-linear time. Nevertheless, our algorithm for decoding Reed-Muller codes in near-linear time is based on a subroutine which decodes Reed-Muller codes with uncertainties from $\left(\frac{1}{2}-\epsilon\right)$ of the minimum distance in near-linear time.

Below we state the main theorem of this section.

Theorem 6.1. *Let $\alpha \in (0,1)$ be a constant. Let $\mathbb{F}$ be a finite field and let $S \subseteq \mathbb{F}$ be a nonempty subset of size $|S| = n$. Let $d = \alpha n$.*

There is a $O\left(n^m \cdot \mathrm{polylog}(n, |\mathbb{F}|)\right)$ time randomized algorithm FastRMDecoder*, which given a received word $r : S^m \to \mathbb{F}$, finds with probability $2/3$, the unique (if any) polynomial $C \in \mathbb{F}[X_1, \ldots, X_m]$ with $\deg(C) \leq d$ such that*

$$\Delta(r, C) < \frac{1}{2}\left(1-\alpha-\frac{10(m-1)\sqrt{n}}{n}\right)n^m.$$

As part of this algorithm for near linear time Reed-Muller decoding to a radius nearly half the minimum distance, we will need to quickly decode Reed-Muller codes with uncertainties to various radii significantly less than half the minimum distance. Such an algorithm, which comes with a tunable slack parameter e, is given by the following theorem. The larger e is, the faster the algorithm runs, at the cost of reducing the number of errors that can be corrected.

Theorem 6.2. *Let $\mathbb{F}$ be a finite field and let $S \subseteq \mathbb{F}$ be a nonempty subset of size $|S| = n$. Let e, d be nonnegative integers, with $e \leq d < n$.*

There is a $O\left(\frac{1}{e+1}\cdot n^m \cdot (d+1)\cdot \mathrm{polylog}(n, |\mathbb{F}|, \frac{1}{\gamma})\right)$ time randomized algorithm FastWeightedRMDecoder$_e$*, which given a received word with uncertainties $w : S^m \to \mathbb{F}\times[0,1]$, finds, with probability $1-\gamma$, the unique (if any) polynomial $C \in \mathbb{F}[X_1, \ldots, X_m]$ with $\deg(C) \leq d$ such that*

$$\Delta(w, C) < \frac{1}{2}\left(1-\frac{d}{n}-\frac{10m\sqrt{n}+e}{n}\right)n^m.$$

[1] In the penultimate step, we used the inequality:

$$\frac{u}{u+\gamma} \leq 1-\gamma,$$

which holds for nonnegative reals u, γ with $u+\gamma \leq 1$. For our application, we set $u = 1-\alpha-\frac{10}{\sqrt{n}}$ and $\gamma = \frac{i}{n}+\frac{10}{\sqrt{n}}$.

Remark 6.3. *By setting* $e = \frac{d}{\mathrm{polylog}(n)}$ *in Theorem 6.2, the resulting algorithm is almost as good as the algorithm from Theorem 6.1. The algorithm from Theorem 6.2 is still near-linear time, but decodes from a number of errors which is a* $(\frac{1}{2} - \frac{1}{\mathrm{polylog}(n)})$*-fraction of the minimum distance, instead of* $(\frac{1}{2} - \frac{1}{\sqrt{n}})$*-fraction of the minimum distance.*

6.1 The main subroutine: decoding Reed-Muller codes with uncertainties

Proof of Theorem 6.2. The proof is by induction on the number of variables m. The proof of the base case of $m = 1$ follows by simply noting that the algorithm FastWeightedRSDecoder mentioned earlier satisfies the requirement.

Now we will show how to decode m-variate Reed-Muller codes, assuming that we have already shown how to decode $m-1$-variate Reed-Muller codes. Thus we will assume access to the algorithm $\mathsf{FastWeightedRMDecoder}_e(f, d, \gamma)$ denote the $O\left(\frac{n^{m-1}(d+1)}{e+1} \cdot \mathrm{polylog}(n, |\mathbb{F}|, \frac{1}{\gamma})\right)$ time algorithm that finds, with probability $1 - \gamma$, the unique polynomial (if any) of degree at most d within distance Λ from f, where $f : S^{m-1} \to \mathbb{F} \times [0, 1]$ and

$$\Lambda = \frac{n^{m-1}}{2}\left(1 - \frac{d}{n} - \frac{10(m-1)\sqrt{n} + e}{n}\right).$$

First some convenient notation. Instead of working with the m variables $X_1, X_2, \ldots, X_m$, we work with the m variables $X_1, X_2, \ldots, X_{m-1}$ and Y. We will use $\boldsymbol{X}$ to denote $(X_1, \ldots, X_{m-1})$.

We begin by observing that it suffices to give a randomized $O\left(\frac{n^m(d+1)}{e+1} \cdot \mathrm{polylog}(n, |\mathbb{F}|)\right)$ time algorithm that succeeds with probability only 2/3: the success probability can be amplified to $1 - \gamma$ by repeating $O(\log \frac{1}{\gamma})$ times.

Below we give a formal description of the algorithm with success probability 2/3. We then give a high-level description of what it does.

Algorithm 4 FastWeightedRMDecoder$_e$: (Quite) Fast decoding of general Reed-Muller codes with Uncertainties

```
1: Input: $w : S^m \to \mathbb{F} \times [0,1]$.
2: Define $r : S^m \to \mathbb{F}$ and $u : S^m \to [0,1]$ by:
       $$w(\boldsymbol{X}, Y) = (r(\boldsymbol{X}, Y), u(\boldsymbol{X}, Y)).$$
   .
3: for $i = 0, 1, \ldots, d$ do
4:     Define $r_i : S^{m-1} \times S \to \mathbb{F}$ by
           $$r_i(\boldsymbol{X}, Y) = r(\boldsymbol{X}, Y) - \sum_{\ell=0}^{i-1} Q_\ell(\boldsymbol{X}) Y^{d-\ell}.$$
5:     for $\boldsymbol{x} \in S^{m-1}$ do
6:         if $(e+1) \mid i$ then
7:             Define $w_{i,\boldsymbol{x}} : S \to \mathbb{F} \times [0,1]$ by
                   $$w_{i,\boldsymbol{x}}(Y) = (r_i(\boldsymbol{x}, Y), u(\boldsymbol{x}, Y)).$$
8:             Define $G_{i,\boldsymbol{x}}(Y) = \mathsf{FastWeightedRSDecoder}(w_{i,\boldsymbol{x}}(Y), d-i, n^{-2m})$.
9:         else
10:            Define $G_{i,\boldsymbol{x}}(Y) = G_{i-1,\boldsymbol{x}}(Y) - Q_{i-1}(\boldsymbol{x}) Y^{d-i+1}$.
11:        end if
12:        if $\deg(G_{i,\boldsymbol{x}}(Y)) \le d - i$ then
13:            $\sigma_{\boldsymbol{x}} = \mathrm{Coeff}_{Y^{d-i}}(G_{i,\boldsymbol{x}}(Y))$.
14:            $\delta_{\boldsymbol{x}} = \Delta(G_{i,\boldsymbol{x}}, w_{i,\boldsymbol{x}})$.
15:        else
16:            $\sigma_{\boldsymbol{x}} = 0$.
17:            $\delta_{\boldsymbol{x}} = n$.
18:        end if
19:    end for
20:    Define the weighted function $f_i : S^{m-1} \to \mathbb{F} \times [0,1]$ by
           $$f_i(\boldsymbol{x}) = \left(\sigma_{\boldsymbol{x}}, \min\left\{1, \frac{\delta_{\boldsymbol{x}}}{(n-d+i-10\sqrt{n}-e)/2}\right\}\right).$$
21:    Define $e_i$ by:
           $$e_i = \lfloor \frac{(i + 10(m-1)\sqrt{n})(d-i)}{n-(d-i)} \rfloor.$$
22:    Define $Q_i : S^{m-1} \to \mathbb{F}$ by
           $$Q_i(\boldsymbol{X}) = \mathsf{FastWeightedRMDecoder}_{e_i}\left(f_i, i, n^{-2m}\right)$$
23: end for
24: Output: $\sum_{i=0}^{d} Q_i(\boldsymbol{X}) Y^{d-i}$.
```

At the high level, the key difference between Algorithm 4 and Algorithm 2 is that we reduce the number of Reed-Solomon decodings, at the cost of reducing the radius to which our overall Reed-Muller decoder works. We write the nearby polynomial $C(\boldsymbol{X}, Y)$ as $\sum_{i=0}^{d} P_i(\boldsymbol{X}) Y^{d-i}$, and find the P_i iteratively. In the i-th iteration, we would like to decode row $w_{i,\boldsymbol{x}}$, $\boldsymbol{x} \in S^{m-1}$ to a nearby degree $d-i$ univariate polynomial. To reduce the number of times we decode, we only do the decoding for one in every $e+1$ iterations of i, and use this decoding for the subsequent e iterations. This reduces the number of decodings by a factor $e+1$, at the cost of reducing the radius to which we decode a univariate polynomial by e (at most). These univariate decodings allow us to construct the weighted function $f_i : S^{m-1} \to \mathbb{F}$, where for each $\boldsymbol{x} \in S^{m-1}$, $f_i(\boldsymbol{x})$ is our guess for $P_i(\boldsymbol{x})$. The algorithm then recovers P_i by recursively calling the $(m-1)$-variable Reed-Muller decoder to decode f_i, with the parameter e set to a carefully chosen value (denoted e_i above). Analyzing this recursion gives the claimed running time.

Overall, this translates into a faster Reed-Muller decoder, albeit with a slightly smaller decoding radius.

Proof of Correctness

We will show that for each $i \in [0,d]$, $Q_i(\boldsymbol{X}) = P_i(\boldsymbol{X})$.

Formally, we will show by induction on i, the following two statements:

1. If $j = \lfloor \frac{i}{e+1} \rfloor$, then for all $\boldsymbol{x} \in S^{m-1}$, $G_{i,\boldsymbol{x}}$ is the unique (if any) polynomial of degree at most $d-i$ such that:
$$\Delta(G_{i,\boldsymbol{x}}, w_{i,\boldsymbol{x}}) \leq \frac{n-d+j\cdot(e+1))}{2} - \frac{10\sqrt{n}}{2}.$$
2. $Q_i = P_i$

Item 1 implies, in particular, that $G_{i,\boldsymbol{x}}$ is the unique polynomial (if any) of degree at most $d-i$ within distance $\frac{n-d+i}{2} - \frac{10\sqrt{n}+e}{2}$ from $w_{i,\boldsymbol{x}}$.

We first prove Item 1. If $(e+1) \mid i$, then Step 8 of the algorithm immediately implies Item 1. For other i, $G_{i,\boldsymbol{x}}$ is obtained from $G_{i-1,\boldsymbol{x}}$ in Step 10 of the algorithm. Suppose $R'(Y) \in \mathbb{F}[Y]$ is a polynomial of degree at most $d-i$ such that
$$\Delta(R', w_{i,\boldsymbol{x}}) \leq \frac{n-d+j\cdot(e+1))}{2} - \frac{10\sqrt{n}}{2}.$$
Consider the polynomial $R(Y) = R'(Y) + Q_{i-1}Y^{d-i+1}$, which is a polynomial of degree at most $d-i+1$. By Item 2 of the induction hypothesis, $R(Y) = R'(Y) + P_{i-1}Y^{d-i+1}$. Thus:[2]
$$\Delta(R, w_{i-1,\boldsymbol{x}}) = \Delta(R', w_{i-1,\boldsymbol{x}} - P_{i-1}Y^{d-i+1}) = \Delta(R', w_{i,x}) \leq \frac{n-d+j\cdot(e+1))}{2} - \frac{10\sqrt{n}}{2}.$$
By Item 1 of the induction hypothesis, we have that R must equal $G_{i-1,\boldsymbol{x}}$, and thus Step 10 of the algorithm sets $G_{i,\boldsymbol{x}}(Y) = R'(Y)$.

Now we prove Item 2. Since Q_i is found by decoding f_i to a nearby polynomial (in Step 22), it will be enough for us to show that P_i is sufficiently close f_i. We begin by computing the radius Λ_i to which the error-correction algorithm in Step 22 decodes. Recall that $e_i = \lfloor \frac{(i+10(m-1)\sqrt{n})(d-i)}{n-(d-i)} \rfloor$. Thus:
$$\begin{aligned}
\Lambda_i &= \frac{1}{2}n^{m-1}\left(1 - \frac{i}{n} - \frac{10(m-1)\sqrt{n}+e_i}{n}\right) \\
&\geq \frac{1}{2}n^{m-1}\left(1 - \frac{i}{n} - \frac{10(m-1)\sqrt{n}}{n} - \frac{(i+10(m-1)\sqrt{n})(d-i)}{(n-(d-i))n}\right) \\
&= \frac{1}{2}n^{m-1}\left(1 - \frac{i(n-(d-i)) + 10(m-1)(n-(d-i))\sqrt{n} + (i+10(m-1)\sqrt{n})(d-i)}{n(n-(d-i))}\right) \\
&= \frac{1}{2}n^{m-1}\left(1 - \frac{in + 10(m-1)n\sqrt{n}}{n(n-(d-i))}\right)
\end{aligned}$$

[2]Here we use a simple fact. For a weighted word $w = (\sigma, u) : S \to \mathbb{F} \times [0,1]$ and a pure word $r : S \to \mathbb{F}$, define $w + r$ by:
$$(w+r)(y) = (\sigma(y) + r(y), u(y)).$$
Then for a pure word $q : S \to \mathbb{F}$, we have:
$$\Delta(q-r, w) = \Delta(q, w+r).$$

$$= \frac{1}{2} n^{m-1} \left(1 - \frac{i + 10(m-1)\sqrt{n}}{n - (d-i)}\right)$$

Thus, if we show that:

$$\Delta(f_i, P_i) < \frac{1}{2}\left(1 - \frac{i + 10(m-1)\sqrt{n}}{n - d + i}\right), \tag{11}$$

then it will mean that P_i is the unique polynomial of degree i within distance Λ_i of f_i, and thus Step 22 will set $Q_i = P_i$.

We now show that Inequality (11) holds.

Letting

$$C_i(\boldsymbol{X}, Y) = \sum_{j=i}^{d} P_j(\boldsymbol{X}) Y^{d-j},$$

we get that $\Delta(w_i, C_i) = \Delta(w, C)$. For $\boldsymbol{x} \in S^{m-1}$, let $C_{i,\boldsymbol{x}}(Y) = C_i(\boldsymbol{x}, Y)$.

By Item 1, we have that $G_{i,\boldsymbol{x}}$ is the result of decoding $w_{i,\boldsymbol{x}}$ to radius $\frac{1}{2}(n - d + i - 10\sqrt{n} - e)$. There are four possibilities:

1. The decoding is unsuccessful (i.e., $G_{i,\boldsymbol{x}} = \perp$). In this case, $\delta_x = n$, and so the the uncertainty in $f_i(\boldsymbol{x})$ equals 1. The contribution of $\boldsymbol{x}$ to $\Delta(f_i, P_i)$ is $\Delta(f_i(\boldsymbol{x}), P_i(\boldsymbol{x})) = 1/2$, which is bounded above by

$$\frac{1}{2} \frac{\Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{(n - d + i - 10\sqrt{n} - e)/2} = \frac{\Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{n - d + i - 10\sqrt{n} - e}.$$

2. The decoding succeeds and is correct. In this case, $G_{i,\boldsymbol{x}} = C_{i,\boldsymbol{x}}$, so:

$$\begin{aligned}\Delta(f_i(\boldsymbol{x}), P_i(\boldsymbol{x})) &= \frac{1}{2} \frac{\Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{(n - d + i - 10\sqrt{n} - e)/2} \\ &= \frac{\Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{n - d + i - 10\sqrt{n} - e}\end{aligned}$$

3. The decoding succeeds, but is the wrong codeword, whose leading coefficient disagrees with that of the correct codeword. In this case, the degree $d - i$ polynomials $G_{i,\boldsymbol{x}} \neq C_{i,\boldsymbol{x}}$, and so:

$$\begin{aligned}\Delta(f_i(\boldsymbol{x}), P_i(\boldsymbol{x})) &= 1 - \frac{1}{2} \frac{\Delta(w_{i,\boldsymbol{x}}, G_{i,\boldsymbol{x}})}{(n - d + i - 10\sqrt{n} - e)/2} \\ &\leq 1 - \frac{(n - d + i) - \Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{n - d + i - 10\sqrt{n} - e} \\ &\leq 1 - \frac{(n - d + i - 10\sqrt{n} - e) - \Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{n - d + i - 10\sqrt{n} - e} \\ &\leq \frac{\Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{n - d + i - 10\sqrt{n} - e)}.\end{aligned}$$

4. The decoding succeeds, but is the wrong codeword, whose leading coefficient matches that of the correct codeword. As in the previous case, $G_{i,\boldsymbol{x}} \neq C_{i,\boldsymbol{x}}$, and we have:

$$\begin{aligned}\Delta(f_i(\boldsymbol{x}), P_i(\boldsymbol{x})) &= \frac{1}{2} \frac{\Delta(w_{i,\boldsymbol{x}}, G_{i,\boldsymbol{x}})}{(n - d + i - 10\sqrt{n} - e)/2} \\ &\leq 1 - \frac{1}{2} \frac{\Delta(w_{i,\boldsymbol{x}}, G_{i,\boldsymbol{x}})}{(n - d + i - 10\sqrt{n} - e)/2} \quad \text{Since } \tfrac{1}{2}\epsilon \leq 1 - \tfrac{1}{2}\epsilon \text{ if } \epsilon \leq 1 \\ &\leq \frac{\Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{n - d + i - 10\sqrt{n} - e},\end{aligned}$$

where the deduction of the last step follows from exactly the same argument as in the previous case.

Putting it all together, we have:

$$\begin{aligned}
\Delta(f_i, P_i) &\leq \sum_{\boldsymbol{x} \in S^{m-1}} \frac{\Delta(w_{i,\boldsymbol{x}}, C_{i,\boldsymbol{x}})}{n - d + i - 10\sqrt{n} - e} \\
&= \frac{\Delta(w_i, C_i)}{n - d + i - 10\sqrt{n} - e} \\
&= \frac{\Delta(w, C)}{n - d + i - 10\sqrt{n} - e} \\
&\leq \frac{\frac{n^m}{2}\left(1 - \frac{d + 10m\sqrt{n} + e}{n}\right)}{n - d + i - 10\sqrt{n} - e} \\
&= \frac{n^{m-1}}{2} \frac{n - d - 10m\sqrt{n} - e}{n - d + i - 10\sqrt{n} - e} \\
&\leq \frac{n^{m-1}}{2} \frac{n - d - 10(m-1)\sqrt{n}}{n - d + i} \\
&= \frac{n^{m-1}}{2}\left(1 - \frac{i + 10(m-1)\sqrt{n}}{n - d + i}\right).
\end{aligned}$$

Analysis of Runtime

The algorithm can be divided into two parts:

1. Constructing the f_i, $i = 0, \ldots, d$.
2. Decoding the f_i to get the P_i, $i = 0, \ldots, d$.

The dominant contribution to the runtime when constructing f_i comes from all the Reed-Solomon decodings with uncertainties we have to do to get the $G_{i,\boldsymbol{x}}(Y)$. For every $e+1$ iterations, we have to decode each row $\boldsymbol{x} \in S^{m-1}$ again. The total number of such decodings is given by $\frac{d+1}{e+1} \cdot n^{m-1} = \frac{n^{m-1}(d+1)}{e+1}$. Since each Reed-Solomon decoding with uncertainty can be done in $O(n \operatorname{polylog}(n))$ time via the FastWeightedRSDecoder, we have that the runtime of this part of the algorithm is $O\left(\frac{n^m(d+1)}{e+1} \operatorname{polylog}(n)\right)$.

To understand the runtime of the second part of the algorithm, we will compute the runtime of decoding f_i for each i. By induction, we know that the running time for this decoding during iteration i is at most $O(\frac{n^{m-1} \cdot (i+1)}{e_i + 1} \cdot \operatorname{polylog}(n))$. Summing this up over all i, we get that the total running time for decoding all the f_is is at most:

$$O(\sum_{i=0}^{d} \frac{n^{m-1} \cdot (i+1)}{e_i + 1} \cdot \operatorname{polylog}(n)) \leq O(n^{m-1} \operatorname{polylog}(n)) \cdot \left(\sum_{i=0}^{d} \frac{i+1}{e_i + 1}\right).$$

It remains to bound $\sum_{i=0}^{d} \frac{i+1}{e_i + 1}$ from above:

$$\begin{aligned}
\sum_{i=0}^{d} \frac{i+1}{e_i + 1} &\leq \sum_{i=0}^{d-1} \frac{i+1}{e_i + 1} + (d+1) \\
&\leq \sum_{i=0}^{d-1} \frac{i+1}{\frac{(i + 10(m-1)\sqrt{n})(d-i)}{n-d+i}} + (d+1) \\
&\leq \sum_{i=0}^{d-1} \frac{(i+1)(n-d+i)}{(i + 10(m-1)\sqrt{n})(d-i)} + (d+1)
\end{aligned}$$

$$\begin{aligned}
&\le \sum_{i=0}^{d-1} \frac{(i+1)(n-d+i)}{(i+10(m-1)\sqrt{n})(d-i)} + (d+1) \\
&\le \sum_{i=0}^{d-1} \frac{n}{d-i} + (d+1) \\
&\le O(n \log d).
\end{aligned}$$

So the runtime for decoding the f_i for all $d+1$ iterations is:

$$O\left(n^{m-1}(n \log d) \operatorname{polylog}(n)\right) = O(n^m \operatorname{polylog}(n)).$$

This means the runtime for both parts of the algorithm is just $O\left(\frac{n^m(d+1)}{e+1} \operatorname{polylog}(n)\right)$, as desired.

□

It is worth noting that the bottleneck for the running time is the the first part; that is, for the Reed-Solomon decodings. This will be speeded up in the subsequent algorithm.

6.2 Decoding Reed-Muller codes in near-linear time

We now prove Theorem 6.1. This algorithm combines aspects of the fast bivariate Reed-Muller decoding algorithm, Algorithm 3, and the previous algorithm, Algorithm 4.

Proof of Theorem 6.1. We begin by giving the decoding algorithm.

Algorithm 5 FastRMDecoder: Fast decoding of general Reed-Muller codes

```
1: Input: $r : S^m \to \mathbb{F}$.
2: Let $c = ((1-\alpha)^2/8)$.
3: Let $e = \lfloor 2cn \rfloor$.
4: for $i = 0, 1, \ldots, d$ do
5:     Define $r_i : S^{m-1} \times S \to \mathbb{F}$ by
           $$r_i(\boldsymbol{X}, Y) = r(\boldsymbol{X}, Y) - \sum_{\ell=0}^{i-1} Q_\ell(\boldsymbol{X}) Y^{d-\ell}.$$
6:     for $\boldsymbol{x} \in S^{m-1}$ do
7:         Define $r_{i,\boldsymbol{x}} : S \to \mathbb{F}$ by
               $$r_{i,\boldsymbol{x}}(Y) = r_i(\boldsymbol{x}, Y).$$
8:         if $(e+1) \mid i$ then
9:             Let $j = \frac{i}{e+1}$.
10:            Let $t_j = \frac{n-d+i}{2} + cn$.
11:            Define $L_{i,\boldsymbol{x}} = \mathsf{FastRSListDecoder}(r_{i,\boldsymbol{x}}, d-i, t_j)$.
12:        else
13:            Define $L_{i,\boldsymbol{x}} = \{R(Y) - Q_{i-1}(\boldsymbol{x}) Y^{d-i+1} \mid R(Y) \in L_{i-1,\boldsymbol{x}}, \mathrm{Coeff}_{Y^{d-i+1}}(R) = Q_{i-1}(\boldsymbol{x})\}$.
14:        end if
15:        Define $G_{\boldsymbol{x}}(Y) \in \mathbb{F}[Y]$ to be the unique element (if any) of the list $L_{i,\boldsymbol{x}}$ with degree at most $d-i$
     and:
               $$\Delta(G_{\boldsymbol{x}}, r_{i,\boldsymbol{x}}) < \frac{n-d+i}{2}$$
16:        if $G_{\boldsymbol{x}} \neq \perp$: then
17:            $\sigma_{\boldsymbol{x}} = \mathrm{Coeff}_{Y^{d-i}}(G_{\boldsymbol{x}})$.
18:            $\delta_{\boldsymbol{x}} = \Delta(G_{\boldsymbol{x}}, r_{i,\boldsymbol{x}})$.
19:        else
20:            $\sigma_{\boldsymbol{x}} = 0$.
21:            $\delta_{\boldsymbol{x}} = n$.
22:        end if
23:    end for
24:    Define the weighted function $f_i : S^{m-1} \to \mathbb{F} \times [0,1]$ by
           $$f_i(\boldsymbol{x}) = \left(\sigma_{\boldsymbol{x}}, \min\{1, \frac{\delta_{\boldsymbol{x}}}{(n-d+i)/2}\}\right).$$
25:    Define
           $$e_i = \lfloor \frac{(i + 10(m-1)\sqrt{n})(d-i)}{n-(d-i)} \rfloor.$$
26:    Obtain the polynomial $Q_i(\boldsymbol{X}) \in \mathbb{F}[X_1, \ldots, X_m]$ by
           $$Q_i(\boldsymbol{X}) = \mathsf{FastWeightedRMDecoder}_{e_i}(f_i(\boldsymbol{X}), i, n^{-2m}).$$
27: end for
28: Output: $\sum_{i=0}^{d} Q_i(\boldsymbol{X}) Y^{d-i}$.
```

Proof of Correctness

As commented earlier, since we have set the error probability argument γ suitably small, we may assume for the analysis that all invocations of the randomized decoder FastWeightedRMDecoder give the correct answer.

Suppose $C(\boldsymbol{X}, Y) = \sum_{i=0}^{d} P_i(\boldsymbol{X})Y^{d-i}$ is a codeword such that

$$\Delta(r, C) < \frac{n^m}{2}\left(1 - \frac{d}{n} - \frac{10(m-1)\sqrt{n}}{n}\right).$$

We want to show that for each i, $Q_i(\boldsymbol{X}) = P_i(\boldsymbol{X})$. We will do this by induction on i.

By induction, on i, we will show that:

1. $r_{i,\boldsymbol{x}}$ in Algorithm 2 and Algorithm 5 are the same for all $\boldsymbol{x} in S^{m-1}$.

2. For $j = \lfloor \frac{i}{e+1} \rfloor$, $L_{i,\boldsymbol{x}}$ contains all polynomials $R'(Y)$ of degree at most $d - i$ such that

$$\Delta(R', r_{i,\boldsymbol{x}}) \leq t_j.$$

3. $Q_i(X) = P_i(X)$.

The analysis will follow the analyses of Algorithm 2, Algorithm 3 and Algorithm 4 very closely.

For $i = 0$, Item 1 is trivial (since in both Algorithm 2 and Algorithm 5, $r_{0,\boldsymbol{x}}(y) = r(\boldsymbol{x}, y)$. For other i, Item 1 follows immediately from the definition of $r_{i,\boldsymbol{x}}$ and from Item 3 of the induction hypothesis for $i - 1$.

For every i with $(e+1) \mid i$ (and in particular for $i = 0$), Item 2 follows from Step 11 of the algorithm. For other i, $L_{i,\boldsymbol{x}}$ is obtained from $L_{i-1,\boldsymbol{x}}$ in Step 13 of the algorithm. Suppose $R'(Y)$ is a polynomial of degree at most $d - i$ such that $\Delta(R', r_{i,\boldsymbol{x}}) \leq t_j$. Consider the polynomial $R(Y) = R'(Y) - Q_{i-1}(\boldsymbol{x})Y^{d-i+1}$ which is of degree at most $d - i + 1$. We will show that $R(Y) \in L_{i-1,\boldsymbol{x}}$, and then Step 13 of the algorithm will put $R'(Y)$ into $L_{i,\boldsymbol{x}}$. By construction,

$$\begin{aligned}
\Delta(R, r_{i-1,\boldsymbol{x}}) &= \Delta(R'(Y) - Q_{i-1}(\boldsymbol{x})Y^{d-i+1}, r_{i-1,\boldsymbol{x}}) \\
&= \Delta(R'(Y), r_{i-1,\boldsymbol{x}} + Q_{i-1}(\boldsymbol{x})Y^{d-i+1}) \\
&= \Delta(R', r_{i,\boldsymbol{x}}) \\
&\leq t_j.
\end{aligned}$$

This, along with the induction hypothesis for $i - 1$, implies that $R(Y) \in L_{i-1,\boldsymbol{x}}$), as desired.

Now we discuss Item 3. Let $j = \lfloor \frac{i}{e+1} \rfloor$. If Item 2 holds for some i, then using the fact that

$$t_j = \frac{n - d + j \cdot (e+1)}{2} + cn = \frac{n - d + j \cdot (e+1) + 2cn}{2} > \frac{n - d + j \cdot (e+1) + e}{2} \geq \frac{n - d + i}{2},$$

we conclude that $G_{\boldsymbol{x}}(Y)$ is the unique polynomial in $\mathbb{F}[Y]$ (if any) of degree at most $d - i$ with $\Delta(G_{\boldsymbol{x}}, r_{\boldsymbol{x},i}) < \frac{n-d+i}{2}$.

Thus the $G_{\boldsymbol{x}}$ produced in Step 15 of Algorithm 5 is the same as the corresponding $G_{\boldsymbol{x}}$ in Algorithm 2. This implies that the function f_i in Algorithm 2 and Algorithm 5 are the same.

We will now show that

$$\Delta(f_i, P_i) < \frac{1}{2}\left(1 - \frac{i + 10(m-1)\sqrt{n}}{n - d + i}\right) \cdot n^{m-1}. \tag{12}$$

By induction, we may assume that we have already shown that $P_\ell = Q_\ell$ for each $\ell \in \{0, 1, \ldots, i-1\}$. Letting:

$$C_i(\boldsymbol{X}, Y) = \sum_{j=i}^{d} P_j(\boldsymbol{X})Y^{d-j},$$

we get that $\Delta(r_i, C_i) = \Delta(r, C)$. Following the same calculation as in (10), we have that:

$$\Delta(f_i, P_i) < \frac{\Delta(r_i, C_i)}{n - d + i}$$

$$\begin{aligned}
&= \frac{\Delta(r, C)}{n-d+i} \\
&\leq \frac{\frac{1}{2}(1-\alpha-\frac{10(m-1)}{\sqrt{n}})n^m}{n-d+i} \\
&\leq \frac{1}{2}\frac{n-d-10(m-1)\sqrt{n}}{n-d+i} \cdot n^{m-1} \\
&= \frac{1}{2}\left(1-\frac{i+10(m-1)\sqrt{n}}{n-d+i}\right) \cdot n^{m-1}.
\end{aligned}$$

This completes the proof of Inequality (12).

Now we study what happens in the invocation of FastWeightedRMDecoder in Step 26 of the algorithm. To do this, we need to compute the radius Λ_i to which the decoder $\mathsf{FastWeightedRMDecoder}_{e_i}$ in Step 26 will decode. Recall that $e_i = \lfloor \frac{(i+10(m-1)\sqrt{n})(d-i)}{n-(d-i)} \rfloor$. Thus:

$$\begin{aligned}
\Lambda_i &= \frac{1}{2}n^{m-1}\left(1-\frac{i}{n}-\frac{10(m-1)\sqrt{n}+e_i}{n}\right) \\
&\geq \frac{1}{2}n^{m-1}\left(1-\frac{i}{n}-\frac{10(m-1)\sqrt{n}}{n}-\frac{i(d-i)}{(n-(d-i))n}\right) \\
&= \frac{1}{2}n^{m-1}\left(1-\frac{i(n-(d-i))+10(m-1)(n-(d-i))\sqrt{n}+(i+10(m-1)\sqrt{n})(d-i)}{n(n-(d-i))}\right) \\
&= \frac{1}{2}n^{m-1}\left(1-\frac{in+10(m-1)n\sqrt{n}}{n(n-(d-i))}\right) \\
&= \frac{1}{2}n^{m-1}\left(1-\frac{i+10(m-1)\sqrt{n}}{n-(d-i)}\right).
\end{aligned}$$

Combining this with Inequality (12), we see that $\Lambda_i > \Delta(f_i, P_i)$. This implies that the invocation of FastWeightedRMDecoder in Step 25 will return P_i, and thus $Q_i = P_i$. This completes the proof of the inductive step.

Now that we have shown that for all i we have $Q_i = P_i$, we can conclude that the output of the algorithm is indeed $C(\boldsymbol{X}, Y)$, as desired.

Analysis of Runtime

All the invocations of FastWeightedRMDecoder to decode the f_i over the $d+1$ values of i take time $O(n^m \operatorname{polylog}(n))$ in total, following the exact same runtime analysis from Theorem 6.2. Now we analyze the time required to construct the f_i. This involves a total of $n^{m-1} \cdot \frac{d}{e+1} = O(n^{m-1})$ invocations of FastRSListDecoder, along with some other simpler operations. The total running time for this part is thus also $O(n^m \operatorname{polylog}(n))$. □

7 Open Problems

We conclude with some open problems.

1. The problem of list-decoding Reed-Muller codes over general product sets S^m up to the Johnson radius is a very interesting open problem left open by our work. Again, when S is algebraically special, it is known how to solve this problem [9]. Generalizing our approach seems to require progress on another very interesting open problem, that of list-decoding Reed-Solomon concatenated codes. See [18] for the state of the art on this problem.

2. It would be interesting to understand the relationship between our algorithms and the $m+1$-variate interpolation-based list-decoding algorithm of Sudan [11]. Their decoding radii are incomparable, and perhaps there is some insight into the polynomial method, which is known to face some difficulties in > 2 dimensions, that can be gained here.

3. It would be interesting to see if one can decode multiplicity codes [19] on arbitrary product sets S^m up to half their minimum distance. Here too, we know algorithms that decode up to half the minimum distance only in the case when S is very algebraically special (from [20]), or if the degree d is very small compared to $|S|$ (via an $m+1$-variate interpolation algorithm, similar to [11]).

Acknowledgments

We are grateful to Madhu Sudan for introducing this problem to us many years ago. Many thanks to the anonymous reviewers for very detailed comments that greatly improved the presentation.

References

[1] D. E. Muller, *Application of boolean algebra to switching circuit design and to error detection*, Electronic Computers, Transactions of the I.R.E. Professional Group on EC-3, 3:6-12, 1954.

[2] I. Reed, *A class of multiple-error-correcting codes and the decoding scheme*, Information Theory, Transactions of the I.R.E. Professional Group on 4, 4:38-49, 1954.

[3] I. S. Reed and G. Solomon, *Polynomial Codes over Certain Finite Fields*, Journal of the Society for Industrial and Applied Mathematics (SIAM) 8:300-304, 1960.

[4] Ronny Roth and Gitit Ruckenstein, *Efficient decoding of Reed-Solomon codes beyond half the minimum distance*, IEEE Transactions on Information Theory 46:246-257, 2000.

[5] Laszlo Babai, Lance Fortnow, Leonid Levin and Mario Szegedy, *Checking computations in polylogarithmic time*, STOC 1991.

[6] J. L. Massey, *Shift-register synthesis and BCH decoding*, IEEE Transactions on Information Theory, 15:122-127, 1969.

[7] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*, Elsevier, 1977.

[8] V. Guruswami and M. Sudan, *Improved decoding of Reed-Solomon and Algebraic-geometric codes* IEEE Transactions on Information Theory, 45:1757-1767, 1999.

[9] R. Pellikaan and X. Wu, *List decoding of q-ary Reed-Muller codes* IEEE Transactions on Information Theory, 50:679-682, 2004.

[10] T. Kasami, S. Lin, and W. Peterson, *Polynomial Codes*, IEEE Transactions on Information Theory, 14:807-814, 2006.

[11] M. Sudan, *Decoding of Reed Solomon Codes beyond the Error-Correction Bound*, J. Complexity, 13:180-193, 1997.

[12] P. Gopalan, V. Guruswami, and P. Raghavendra, *List Decoding Tensor Products and Interleaved Codes*, SIAM J. Comput., 40:1432-1462, 2011.

[13] E. Berlekamp, *Bounded distance +1 soft-decision Reed-Solomon decoding*, IEEE Transactions on Information Theory, 42:704-720, 1996.

[14] L. R. Welch and E. R. Berlekamp, *Error correction of algebraic block codes*, US Patent Number 4633470, 1986.

[15] G. D. Forney, *Generalized Minimum Distance decoding*, IEEE Transactions on Information Theory, 12:125-131, 1966.

[16] Y. Cassuto and J. Bruck, *A Combinatorial Bound on the List Size*, Paradise Laboratory Technical report, California Institute of Technology, 2004.

[17] M. Alekhnovich, *Linear Diophantine Equations Over Polynomials and Soft Decoding of Reed-Solomon Codes*, IEEE Transactions on Information Theory, 51:2257-2265, 2005.

[18] V. Guruswami and M. Sudan, *Decoding concatenated codes using soft information*, Proceedings of the 17th IEEE Annual Conference on Computational Complexity, 122-131, 2002.

[19] S. Kopparty, S. Saraf, and S. Yekhanin, *High-rate Codes with Sublinear-time Decoding*, Journal of the ACM, 61, 28:1-20, 2014.

[20] S. Kopparty, *List decoding multiplicity codes*, Theory of Computing, 11:149-182, 2015.

Appendix A: Decoding of Reed-Solomon Codes with Uncertainties

In this section, we present algorithms for decoding Reed-Solomon codes with uncertainties. We first present Forney's near-quadratic time deterministic algorithm to decode Reed-Solomon codes with uncertainties up to half the minimum distance. We then present a randomized near-linear time version of that algorithm that decodes Reed-Solomon codes with uncertainties up to almost half the minimum distance.

Lemma A.1. *Let $\mathbb{F}$ be a finite field and let $S \subseteq \mathbb{F}$ be a nonempty subset of size $|S| = n$.*

There is a deterministic algorithm WeightedRSDecoder *which, given a weighted received word $w : S \to \mathbb{F} \times [0,1]$, finds the unique polynomial (if any) $C \in \mathbb{F}[X]$ satisfying $\deg(C) \leq d$ and $\Delta(w, C) < \frac{n-d}{2}$ in time $O(n^2 \mathsf{poly}(\log(n), \log(1/\gamma)))$.*

Proof. We first present a randomized algorithm, and then we show how to derandomize it.

Let $r : S \to \mathbb{F}$ and $u : S \to [0,1]$ be the components of w.

The randomized algorithm is based on randomly rounding the weighted received word w to an "errors and erasures" received word (i.e., a word $r' : S \to \mathbb{F} \cup \{?\}$, where ? denotes erasure).

Concretely, for each $x \in S$, the algorithm decides to make $r'(x) = r(x)$ with probability $1-u$, and decides to make $r'(x) =?$ with probability u.

If $C(X)$ is a polynomial of degree at most d with $\Delta(w, C) = \delta$, then a simple analysis shows that:

$$\mathbb{E}[2E + F] = 2\delta,$$

where E is the number of errors (= the number of x with $r'(x) \neq C(x)$ and $r'(x) \neq ?$). and F is the number of erasures (= the number of x with $r'(x) =?$).

Furthermore, this holds even if the random choices in the construction of r' are made in the following correlated way: we first pick $p \in [0,1]$ uniformly at random, and then we set $r'(x) =?$ if and only if $p \leq u(x)$, and we set $r'(x) = r(x)$ otherwise.

Thus there is a choice of $p \in [0,1]$ (and in fact $p \in \{u(x) : x \in S\} \cup \{0,1\}$) such that the resulting function r' has $2E + F \leq 2\delta$.

Finally, a standard fast Reed-Solomon decoding algorithm, such as FastRSDecoder, can decode Reed-Solomon codes of degree d when

$$2 \cdot (\text{num errors}) + (\text{num erasures}) < n - d.$$

This uses the fact that erasures can be thought of as changing the domain S to a subset of S.

To make this algorithm deterministic, we simply cycle over all choices of $p \in \{u(x) : x \in S\} \cup \{0,1\}$ and consider the corresponding r'. There are at most $O(n)$ such choices of p, and this gives us the desired algorithm and running time. For details, see Forney [15]. □

The faster randomized variant simply augments Forney's randomized decoding algorithm with a variance analysis to show that it succeeds with high probability (provided $\Delta(w, C)$ is somewhat smaller than half the minimum distance of the Reed-Solomon code).

Lemma A.2. *Let $\mathbb{F}$ be a finite field and let $S \subseteq \mathbb{F}$ be a nonempty subset of size $|S| = n$.*

There is a randomized algorithm $\mathsf{FastWeightedRSDecoder}$ *which, given a weighted received word $w : S \to \mathbb{F} \times [0,1]$, finds the unique polynomial (if any) $C \in \mathbb{F}[X]$ satisfying $\deg(C) \le d$ and $\Delta(w, C) < \frac{n-d-10\sqrt{n}}{2}$ with probability $1 - \gamma$ in time $O(n\mathsf{poly}(\log(n), \log(1/\gamma)))$.*

Proof. For a given weighted received word $w : S \to \mathbb{F} \times [0,1]$, with $w = (r, u)$, suppose there is a polynomial C of degree at most d such that $\Delta(w, C) < \frac{n-d-\sqrt{n}}{2}$. We will show how to design such an algorithm with success probability $2/3$. This can be amplified to $1 - \gamma$ by repeating $O(\log \frac{1}{\gamma})$ times.

Algorithm 6 Fast Reed-Solomon Decoding with Uncertainties

```
1: Input: $w : S \to \mathbb{F} \times [0,1]$.
2: Let $r : S \to \mathbb{F}$, $u : S \to [0,1]$ be the components of $w$.
3: for $x \in S$ do
4:     Pick $p_x \in [0,1]$ uniformly at random.
5:     Define $r' : S \to \mathbb{F} \cup \{?\}$ by
           $r'(x) = \begin{cases} r(x) & \text{if } p_x \le u(x) \\ ? & \text{if } p_x > u(x) \end{cases}.$
6: end for
7: $g = \mathsf{FastRSDecoder}(r', d)$.
8: Output: $g$.
```

As before, we have $\mathbb{E}[2E + F] < n - d$.

We will use Chebyshev's inequality to show that $2E + F < n - d$ with probability at least $\frac{3}{4}$. To help us compute the expectation and variance of $2E + F$, we write E and F as a sum of indicator random variables. Let $A = \{x \in S | C(x) = r(x)\}$ be the set of agreeing indices, and let $D = \{x \in S | r(x) \ne C(x)\}$ be the set of disagreeing indices. Let $T = \{x \in X | r'(x) = ?\}$ be the set of erasure indices.

Then we can write

$$\begin{aligned} E &= \sum_{x \in D} 1_{x \notin T} \\ F &= \sum_{x \in S} 1_{x \in T}. \end{aligned}$$

We then can show $\mathbb{E}[2E + F]$ is less than $n - d$ by a significant amount $\sqrt{n}$:

$$\begin{aligned} \mathbb{E}[2E + F] &= 2\sum_{x \in D} (1 - u(x)) + \sum_{s \in X} u(x) \\ &= 2\sum_{x \in D} (1 - u(x)) + \sum_{x \in D} u(x) + \sum_{x \in A} u(x) \\ &= 2\left(\sum_{x \in D} \left(1 - \frac{u(x)}{2}\right) + \sum_{x \in A} \frac{u(x)}{2}\right) \\ &= 2\Delta(w, C) \\ &< n - d - 10\sqrt{n}. \end{aligned}$$

Finally, we show that $\mathrm{Var}(2E + F)$ is small:

$$\begin{aligned} & \mathrm{Var}(2E + F) \\ = \; & 4\mathrm{Var}(E) + 4\mathrm{Cov}(E, F) + \mathrm{Var}(F) \\ = \; & 4\sum_{x \in D} u(x)(1 - u(x)) + 4\left(\mathbb{E}\left[\sum_{x \in D}\sum_{y \in S} 1_{x \notin T \wedge y \in T}\right] - \sum_{x \in D}(1 - u(x))\sum_{y \in S} u(y)\right) + \sum_{y \in S} u(y)(1 - u(y)) \end{aligned}$$

$$
\begin{aligned}
&= 4\sum_{x\in D} u(x)(1-u(x)) + 4\left(\mathbb{E}\left[\sum_{x\in D}\sum_{y\neq x}(1-u(x))u(y)\right] - \sum_{x\in D}\sum_{y\in S}(1-u(x))u(y)\right) + \sum_{y\in S} u(y)(1-u(y)) \\
&= 4\sum_{x\in D} u(x)(1-u(x)) - 4\sum_{x\in D} u(x)(1-u(x)) + \sum_{y\in S} u(y)(1-u(y)) \\
&= \sum_{y\in S} u(y)(1-u(y)) \\
&\leq \frac{n}{4}.
\end{aligned}
$$

By Chebyshev's inequality, $\Pr(2E+F \geq n-d) \leq \frac{1}{4}$. Hence we have $\Pr(2E+F < n-d) \geq \frac{3}{4}$. That is, with probability at least $\frac{3}{4}$, the algorithm outputs C.

□