6.897 Algorithmic Introduction to Coding Theory	September 5, 2001
Lecture 15	
Lecturer: Madhu Sudan	*Scribe: Constantine Caramanis*

1 Introduction

This lecture covers the decoding of Reed-Muller Codes. Over the years, these codes have been repeated targets of decoding algorithms, with many successful approaches. In fact, the one of the two papers that led to the current name of these codes really only gives an algorithm for decoding (and doesn't construct codes, per se). Much of this work is carried out in the coding theory literature. It culminates in a decoding algorithm that can decode Reed-Muller codes up to half the minimum distance, for *every* choice of parameters. An elegant description of this algorithm is given in The Handbook of Coding Theory [19] in the chapter on Algebraic Geometry Codes by Høholdt, van Lint, and Pellikaan [13]. We won't cover their algorithm today, but if Madhu gets the energy, he might add a writeup on their algorithm to these lecture notes sometime soon.

Instead today's lecture will focus on list-decoding algorithms for Reed-Muller codes. Unlike the algorithms mentioned above, these do not work for *every* choice of parameters. However if the alphabet size q is sufficiently large, compared to the degree of the polynomials in the message space, these algorithms work very well and decode much more than half the errors.

In addition to the ability to correct many errors, these algorithms are significant in that they played a significant role in many developments in complexity theory over the past two decades. E.g., Lipton [16] showed that the "permanent of a matrix was hard to compute on random matrices". This result was a consequence of a simple unambiguous decoding algorithm for Reed-Muller codes implicit in Beaver and Feigenbaum [7]. The decoding algorithm for Reed-Muller codes also played a role in results showing "IP = PSPACE [17, 20]", "MIP=PSPACE [5]" and the "PCP Theorem [3, 2]". There is also a sequence of results showing progressively weaker complexity-theoretic conditions that would suffice to show "BPP = P" [18, 6, 14]. These results can also be simplified and optimized by using list-decoders for Reed-Muller codes [22].

The algorithm in today's lecture will be derived as a simple algorithm that exploits some randomness properties of "lines" in $\mathbb{F}_q^m$, rather than any serious algebraic properties. The only algebraic elements will be decoding algorithms for Reed-Solomon codes and we'll just adopt them as a black box from previous lecture. We'll describe the algorithm in two steps - first we give an algorithm that decodes from a small number of errors (much less than half the minimum distance). Then we jump to a list-decoder recovering from large number of errors (under the above-mentioned caveat of $q \gg \ell$).

Let us first recall the definition of Reed-Muller codes (first covered in lecture 4 on September 19th, 2001). Recall that these are the generalization of Reed-Solomon codes to multivariate polynomials.

Definition 1 (Reed-Muller Codes) *A Reed-Muller code,* $\mathrm{RM}_{m,d,q}$*, is the code whose codewords are evaluations of* m*-variate polynomials of total degree at most* d*, over all elements in* $\mathbb{F}_q$*.*

The $\mathrm{RM}_{m,d,q}$ code is a linear code with $n = q^m$, $k = \binom{m+d}{d}$, and with relative distance $\left(1 - \frac{d}{q}\right)$, when $d \leq q$. The goal of today's lecture to give list-decoding algorithms that work provided $d \ll q$.

For today's lecture it is convenient to think of this task as a "function reconstruction task" rather than that of the task of reconstructing a vector or string representing the codeword. We will thus think of the received word being given by some function $f : \mathbb{F}_q^m \to \mathbb{F}_q$, and our goal is to output a list of all nearby codewords, where the codewords are also given by functions $p : \mathbb{F}_q^m \to \mathbb{F}_q$. We will also extend the relative Hamming distance to apply to functions. Thus we have the distance between functions f and g to be $\delta(f,g) = \Pr_{\mathbf{x} \in \mathbb{F}_q^m}[f(x) \neq g(x)]$. Note that this is just the standard Hamming distance normalized so as to be between 0 and 1, if f and g are interpreted as strings rather than functions.

> **Reed-Muller Decoding:**
> GIVEN: Oracle access to a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$, a degree parameter d and a disagreement parameter δ.
> TASK: A (list of all) degree d polynomials $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that $\delta(f,p) \leq \delta$.

Note that we haven't specified representation of the output polynomial p precisely. The natural representation may be the coefficients of p. However, given that interpolation is a simple task, it would be equally useful to have p represented as a table of values. The latter representation certainly allows one to compute the former representation in time $\mathrm{poly}(q^m)$. However our algorithms will be more natural in the latter representation; and also significantly more efficient. In fact we will produce randomized algorithms that compute $p(\mathbf{a})$ for any vector $\mathbf{a} \in \mathbb{F}_q^m$ in time $\mathrm{poly}(m,d,q)$. When $d \approx m$ this is vastly more efficient that $\mathrm{poly}(q^m)$. The model needs careful fleshing out; but we won't do so right away. Instead we will let it evolve and the spend some time reviewing the final model at the end of these notes.

2 Decoding from very low error

We start with describing an extremely simple randomized algorithm that recovers from very little error δ. The amount of error will certainly be small enough to put us in the case of the unambiguous decoding problem. So the polynomial p that is δ-close to f is unique. Our algorithm will try to guess the value $p(\mathbf{a})$, by looking at f on a small random sample of values. The trick is in picking the right "random sample". We will look at f on a "line" in $\mathbb{F}_q^m$. We define the notion of a line next.

2.1 Lines in $\mathbb{F}_q^m$

Definition 2 *The line $\mathbb{F}_q^m$ through a point $\mathbf{a}$ with slope $\mathbf{b}$ is the set of points:*

$$\ell_{\mathbf{a},\mathbf{b}} := \{\mathbf{a} + t\mathbf{b} \mid t \in \mathbb{F}_q\}.$$

Note that if $\mathbf{b} = \mathbf{0}$, then the line $\ell_{\mathbf{a},\mathbf{b}}$ consists of a single point, else it consists of exactly q distinct points of $\mathbb{F}_q^m$. While it might be tempting to simply rule out all lines with $\mathbf{b} = \mathbf{0}$ as uninteresting, we won't do so, for reasons to be clarified later. However we will refer to them as *degenerate* lines, just to put them in their place.

In the definition above, we thought of the line as an unordered set of points. However it is also useful to think of them as a "parameterized" set of points. More precisely, a line is a function $\ell_{\mathbf{a},\mathbf{b}} : \mathbb{F}_q \to \mathbb{F}_q^m$, given by $\ell_{\mathbf{a},\mathbf{b}}(t) = \mathbf{a} + t\mathbf{b}$.

The two representations have their advantages. Switching between them is even further beneficial, as we will see towards the end of this lecture. While switching, one should keep in mind that the parameterization specifies the set uniquely; however the set does not specify the parameterization uniquely. In particular the lines $\ell_{\mathbf{a},\mathbf{b}}$ and $\ell_{\mathbf{a}+t_1\mathbf{b},t_2\mathbf{b}}$ are the same for $t_1, t_2 \in \mathbb{F}_q$ if $t_2 \neq 0$. This will be useful to us later.

We start by describing the nice properties of lines. The first nice property of a line is its randomness property. We first define the notion of a random line and a random line through a point $\mathbf{a} \in \mathbb{F}_q^m$.

Definition 3 *For a point $\mathbf{a} \in \mathbb{F}_q^m$, a random line through $\mathbf{a}$ is the random variable $\ell_{\mathbf{a},\mathbf{b}}$ where $\mathbf{b}$ is picked uniformly from $\mathbb{F}_q^m$. A random line in $\mathbb{F}_q^m$ is the random variable $\ell_{\mathbf{a},\mathbf{b}}$ where both $\mathbf{a}$ and $\mathbf{b}$ are chosen independently and uniformly at random from $\mathbb{F}_q^m$.*

The following proposition explains the niceties of random lines.

Proposition 4 *A random line through $\mathbb{F}_q^m$ is a collection of pairwise independent points in $\mathbb{F}_q^m$. I.e., for $t_1 \neq t_2 \in \mathbb{F}_q$, the points $\ell_{\mathbf{a},\mathbf{b}}(t_1)$ and $\ell_{\mathbf{a},\mathbf{b}}(t_2)$ are distributed independently and uniformly from $\mathbb{F}_q^m$. Furthermore, for every $\mathbf{a}$ every non-zero point on a random line through $\mathbb{F}_q^m$ is a random point. I.e., for every $t \in \mathbb{F}_q - \{0\}$, the point $\ell_{\mathbf{a},\mathbf{b}}(t)$ is distributed uniformly in $\mathbb{F}_q^m$, when $\mathbf{b}$ is distributed uniformly in $\mathbb{F}_q^m$.*

Remark: Note that the proposition above relies on the fact that we allow lines to be degenerate, else it would not be true.

Proof Proof to be added. ■

Next we point out the nice algebraic properties of the line. This notion alludes to the restrictions of functions to lines. Note that when the line is viewed as a function $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$, then it composes naturally with a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$ to give a "univariate function" $f|_\ell : \mathbb{F}_q \to \mathbb{F}_q$, given by $f|_\ell(t) = f(\ell(t))$. The following proposition asserts that restrictions of algebraically nice functions give algebraically nice functions.

Proposition 5 *For every degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ and every line $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$, the function $p|_\ell$ is a univariate polynomial of degree at most d.*

Given a line $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$ we say that a point $\mathbf{z} \in \mathbb{F}_q^m$ *lies* on the line ℓ if there exists $t \in \mathbb{F}_q$ such that $\mathbf{z} = \ell(t)$. If the line is not degenerate, then such a t is unique if it exists. If the line is degenerate, then every t works, and we will default to $t = 0$ in case of ambiguities. Specifically, if $\mathbf{z}$ lies on ℓ, then we let $\ell^{-1}(\mathbf{z}) = t$ such that $\mathbf{z} = \ell(t)$, if ℓ is not degenerate, and $\ell^{-1}(\mathbf{z}) = 0$ otherwise.

Univariate functions defined on lines arise naturally in this lecture. Given a univariate polynomial h and a line $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$, we will let $h_{\ell^{-1}}$ denote the partial function from ℓ (viewed as a subset of $\mathbb{F}_q^m$) to $\mathbb{F}_q$ given as follows: $h_{\ell^{-1}}\{\mathbf{z}\} = h(\ell^{-1}(\mathbf{z}))$. If the line ℓ is clear from context we will omit it, and use the notation $h\{\mathbf{z}\}$ to denote $h_{\ell^{-1}}\{\mathbf{z}\}$.

2.2 The algorithm

Our first algorithm for decoding Reed-Muller codes is an elementary consequence of the two properties of random lines in $\mathbb{F}_q^m$ mentioned above. (The latter property is actually a property of all lines and hence also a property of random lines.)

As noted earlier it suffices to construct an algorithm that computes $p(\mathbf{a})$, given oracle access to a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$ that is very close to a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$.

In fact it even suffices to give a randomized algorithm that computes this value with probability greater than, say, 2/3, over its internal coin tosses. Repetition followed by a plurality vote suffices to reduce this error probability. Once the error probability reduces to say less than $\frac{1}{3}q^{-m}$, then repeating this trial for every choice of $\mathbf{a}$ produces the correct codeword with probability at least $\frac{2}{3}$.

The basic idea to compute $p(\mathbf{a})$ is to focus on just a random line ℓ through $\mathbf{a}$ and to reconstruct the function $p|_\ell$. By Proposition 5 this is a univariate polynomial of degree d; and based on Proposition 4 we know that $p|_\ell$ and $f|_\ell$ have good agreement. So $p|_\ell$ can, hopefully, be recovered efficiently and once this is done, all we need to do is output $p_\ell\{\mathbf{a}\}$. Below we describe and analyze the most elementary form of this algorithm. In this form the algorithm only needs $q \geq d+2$ to work. However the amount of error it will correct is quite small. We will worry about that later though.

Simple RM decoder:

Given: Oracle access to $f : \mathbb{F}_q^m \to \mathbb{F}_q$. Point $\mathbf{a} \in \mathbb{F}_q^m$ and parameter d.

Promise: There exists a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that $\delta = \delta(p, f) \leq \frac{1}{3(d+1)}$.

Goal: Output $p(\mathbf{a})$.

Step 1: Pick $\mathbf{b} \leftarrow \mathbb{F}_q^m$ at random and let $\ell = \ell_{\mathbf{a},\mathbf{b}}$.

Step 2: Let $\alpha_1, \ldots, \alpha_{d+1}$ be distinct elements of $\mathbb{F}_q - \{0\}$. For $i \in [d+1]$, let $\beta_i = f(\mathbf{a} + \alpha_i \mathbf{b})$.

Step 3; Interpolate to find a degree d univariate polynomial h such that $h(\alpha_i) = \beta_i$ for every $i \in [d+1]$.

Step 4: Output $h(0)$.

Note that Step 2 above requires $q > d+1$. We will show below that this requirement suffices for correctness, provided the error is small enough.

Lemma 6 *Under the assumption $\delta = \delta(f, p) \leq \frac{1}{3(d+1)}$ the algorithm* **Simple RM decoder** *outputs $p(\mathbf{a})$ with probability at least $\frac{2}{3}$.*

Proof We define $d+1$ "bad" events B_i over the random choice of $\mathbf{b}$. We show that if none of these events occurs, then the algorithm correctly outputs $p(\mathbf{a})$. Then we show that the probability that none of these events occurs is at least $1-(d+1)\delta$. The lemma follows once the above is shown.

We define the event B_i to be the case "$\beta_i \neq p|_\ell(\alpha_i)$". Note that if none of these events occur, then we know the value of the function $p|_\ell(\cdot)$ at $d+1$ distinct values in $\mathbb{F}_q$. Further, by Proposition 5, $p|_\ell$ is a polynomial of degree at most d. Thus, the polynomial h found in Step 3 *is* the function $p|_\ell$.

Thus the value output in Step 4 is $p|_\ell(0) = p(\ell(0)) = p(\mathbf{a})$. It thus suffices to bound the probability of the bad events.

Note that B_i happens if and only if $f(\ell(\alpha_i)) \neq p(\ell(\alpha_i))$. By Proposition 4, we have that $\ell(\alpha_i)$ is a random point of $\mathbb{F}_q^m$ and thus the probability that f does not agree with p at this point is exactly $\delta(f,p)$. Thus we have that the probability of B_i is δ. By the union bound, the probability that at least one of the bad events occurs is at most $(d+1)\delta$. The probability that none of them occurs is at most $1-(d+1)\delta$. This concludes the proof. ■

Note that the algorithm is quite efficient — it runs in time $\mathrm{poly}(m,d)$, while the codeword has length $\binom{m+d}{d}$ which could be exponentially larger. Of course, it does not recover all the codeword at once — this is simply impossible in this much time; but it can recover any coordinate of the codeword in such time. When we formally describe the model in which the algorithm works, we will focus on these aspects and then formally describe the result obtained so far. For now, we will satisfy ourselves with just an informal understanding of the result.

3 Improving the error-correction capability

We now describe ideas that can be applied to improving the error-correction capability of the simple algorithm described in the previous section. The error-correction capability comes at a price: The requirement on the field size now goes up, and the algorithms get slightly more complicated.

To motivate the basic idea behind the improvement, let us look back to the reason why the error correction capability was so low ($\Theta(1/d)$) in the algorithm of the last section. The reason we lost so much was that we required that $d+1$ queries on a random line through $\mathbf{a}$ should *all* be error-free. To improve the performance, we will make a few more queries, but then allow for the possibility that a few answers are incorrect. The polynomial $p|_\ell$ will then be the polynomial that "usually" agrees with the queried values — this polynomial can be found by a Reed-Solomon decoding step. The number of queries that the algorithm makes can be varied depending on the rate of error we wish to correct, the run time we desire, and the field size. We will set this number, somewhat arbitrarily, to $5(d+1)$ and demonstrate the effect. We thus get the algorithm below:

Improved RM decoder:

Given: Oracle access to $f : \mathbb{F}_q^m \to \mathbb{F}_q$. Point $\mathbf{a} \in \mathbb{F}_q^m$ and parameter d.

Promise: There exists a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that $\delta = \delta(p,f) \leq \frac{2}{15}$.

Goal: Output $p(\mathbf{a})$.

Step 1: Pick $\mathbf{b} \leftarrow \mathbb{F}_q^m$ at random and let $\ell = \ell_{\mathbf{a},\mathbf{b}}$.

Step 2: Let $\alpha_1, \ldots, \alpha_{5(d+1)}$ be distinct elements of $\mathbb{F}_q - \{0\}$. For $i \in [5(d+1)]$, let $\beta_i = f(\mathbf{a} + \alpha_i \mathbf{b})$.

Step 3; Find a degree d univariate polynomial h such that $h(\alpha_i) = \beta_i$ for at least $3(d+1)$ choices of $i \in [5(d+1)]$.

Step 4: Output $h(0)$.

All steps above are efficient. In particular, Step 3 can be executed in $\mathrm{poly}(d)$ time by using any unambiguous decoding algorithm for the problem such as the Welch-Berlekamp algorithm described in a previous lecture. We thus get the following proposition.

Proposition 7 **Improved RM decoder** *runs in* $\text{poly}(d, m)$ *time.*

Lemma 8 *Under the assumption* $\delta = \delta(f, p) \leq \frac{2}{15}$ *the algorithm* **Simple RM decoder** *outputs* $p(\mathbf{a})$ *with probability at least* $\frac{2}{3}$.

Proof As in the proof of Lemma 6 we define, for every $i \in [5(d+1)]$, B_i to be the event "$\beta_i \neq p|_\ell(\alpha_i)$". Note that for every i the probability of B_i is exactly δ. Now let B be the event that B_i is true for more than $2(d+1)$ choices of i. The probability that B occurs can be upper bounded, using Markov's inequality, by $\delta * (5(d+1))/(2(d+1)) \leq 1/3$. Thus with probability at least 2/3rd B does not occur. Now we note that if B does not occur, then $p|_\ell$ agrees with the points $\{(\alpha_i, \beta_i) \mid i \in [(5(d+1)]\}$ on $3(d+1)$ points and hence is the unique solution in Step 3. Thus in Step 4 we output $p|_\ell(0) = p(\mathbf{a})$ with probability at least $2/3$. ■

The ideas in the algorithm above can be pushed further to get an algorithm correcting almost $\frac{1}{2}$ of error though this requires some work (and further restrictions on the ratio of q to d). However, we won't describe the details. Instead we move on to the task of list-decoding.

4 A List Decoding Algorithm

Part of the reason why the algorithms from the previous section could not correct too many errors is that they never exploited the ability to list-decode the Reed-Solomon codes (in Step 4). If we did, we would be able to find polynomials with much smaller agreement with f on any given line ℓ. However this doesn't suffice to solve the list-decoding problem for Reed-Muller codes. What should we do with a list of univariate polynomials that agree with f on ℓ? How do we find out which one is the polynomial "p"? In fact, come to think of it, what is p? In previous sections p was uniquely specified as the nearest polynomial (codeword) to the function f (the received vector). Now that we are hoping to perform list-decoding, there is a list of polynomials that could have the desired agreement. It seems that we can no longer use the "reconstruction problem" (compute the value of the nearest polynomial at some fixed point $\mathbf{a} \in \mathbb{F}_q^m$) to solve the decoding problem.

Turns out one can salvage this path after all. The way we'll do this is by focussing on one of the specific polynomials p that is close to f, and by giving a small amount of additional information, called *advice*, that suffices to specify it uniquely. We will then give a reconstruction procedure to computes $p(\mathbf{a})$ given $\mathbf{a}$. By varying the advice, we'll then be able to compute all the polynomials close to f.

So what is the advice that we'll use to specify p? It turns out the value of p at one, randomly chosen point $\mathbf{b} \in \mathbb{F}_q^m$, specifies it uniquely, provided q is large enough relative to δ and d. As usual, when it comes to list-decoding it is more informative to focus on the amount of agreement rather than the amount of disagreement between f and the polynomial p. Define $\tau(f, g) = \Pr_{\mathbf{x} \leftarrow \mathbb{F}_q^m}[f(x) = g(x)]$ to be the *agreement* between f and g. We start with the following simple proposition that proves that the value of a polynomial at a random point specifies it uniquely, given a nearby function f.

Proposition 9 *Let* $p_1, \ldots, p_n$ *be a list of all* m*-variate degree* d *polynomials over* $\mathbb{F}_q$ *satisfying* $\tau(f, g) \geq \tau_0$. *If* $\tau \geq \sqrt{2d/q}$, *then* $n \leq 2/\tau$ *and with probability at least* $1 - \binom{n}{2}\left(\frac{d}{q}\right) \geq 1 - \frac{2d}{\tau^2 q}$ *over the choice of* $\mathbf{z} \in \mathbb{F}_q^m$ *it is the case that the sequence of elements* $\langle p_1(\mathbf{z}), \ldots, p_n(\mathbf{z})\rangle$ *are all distinct.*

Proof The first part of the proposition, claiming $n \leq 2/\tau$ is just recalling the Johnson bound from a previous lecture. The second part follows from an application of the union bound to the $\binom{n}{2}$ possible "bad" events B_{ij}, $1 \leq i < j \leq n$, where B_{ij} is the event "$p_i(\mathbf{z}) = p_j(\mathbf{z})$". Note that B_{ij} occurs with probability at most d/q (by the "Schwartz Lemma"). ■

So this motivates the plan of the next algorithm. We will assume that we are given one useful piece of information — namely the value of p at one (randomly chosen) point $\mathbf{z} \in \mathbb{F}_q^m$. Say $p(\mathbf{z}) = \gamma$. Now suppose we wish to reconstruct the value of p at $\mathbf{a} \in \mathbb{F}_q^m$. We will execute the algorithm of the previous sections and pick a line ℓ through $\mathbf{a}$. Suppose we are fortunate enough that $\mathbf{z}$ lies on ℓ. In this case, given a list of polynomials $h_1, \ldots, h_n$ that have non-trivial agreement with f on ℓ, we can determine which one is $p|_\ell$ by considering the values $h_i\{\mathbf{z}\}$. Hopefully they are all different and then the polynomial h_i for which $h_i\{\mathbf{z}\} = \gamma$ is $p|_\ell$. $h_i\{\mathbf{a}\}$ is then the value we are seeking. The only catch is that a random line will no longer work - it is very unlikely to pass through $\mathbf{z}$. We will fix this by deterministically picking the line that passes through $\mathbf{z}$! This plan is implemented below, with some of the steps being clarified further. The algorithm is described for a fixed choice of $\mathbf{z}$ and advice γ. For reasons to be clarified later, we will simply call this a subroutine.

List-decoding subroutine $A_{\mathbf{z},\gamma}$

Given: Oracle access to f and point $\mathbf{a} \in \mathbb{F}_q^m$.

Step 1: Let $\mathbf{b} = \mathbf{z} - \mathbf{a}$ and let $\ell = \ell_{\mathbf{a},\mathbf{b}}$.

Step 2: For $\alpha \in \mathbb{F}_q$, let $\beta_\alpha = f(\mathbf{a} + \alpha\mathbf{b})$.

Step 3; Find all degree d univariate polynomials $h_1, \ldots, h_n$ such that $h_i(\alpha) = \beta_\alpha$ for at least $\frac{\tau}{2}q$ choices of $\alpha \in \mathbb{F}_q$.

Step 4: If there exists a unique index $i \in [n]$ such that $h_i\{\mathbf{z}\} = \gamma$, output $h_i(0)$, else output error.

We start by noticing that all steps above run in polynomial time provided the parameters are favorable. In particular, Step 3 can be executed in polynomial time assuming $\frac{\tau}{2} > \sqrt{d/q}$ using the improved List-decoding algorithm for Reed-Solomon codes from a previous lecture.

Proposition 10 *If* $\tau > \sqrt{4d/q}$ *then the subroutine* $A_{\mathbf{z},\gamma}$ *runs in time* $\text{poly}(q, m)$.

Exercise: Suggest variations of the algorithm above so it runs in time $\text{poly}(d, \frac{1}{\tau}, \log q)$.

Next we analyze the correctness of the algorithm. We show first that for a random pair $(\mathbf{z}, \mathbf{a})$, the algorithm $A_{\mathbf{z},p(\mathbf{z})}$ is very likely to output $p(\mathbf{a})$. We conclude that there exists a vector $\mathbf{z}$ (in fact most choices would work) such that $A_{\mathbf{z},p(\mathbf{z})}$ computes a function very close to p. This is not what we want — we want an algorithm that always computes p. However the algorithms of the previous section can now be applied to $A_{\mathbf{z},p(\mathbf{z})}$ to get a randomized algorithm that computes p correctly everywhere with high probability. Thus the following lemma will be quite sufficient for our purposes.

Lemma 11 *For any* $\epsilon > 0$, *suppose* $q \geq \frac{16(d+1)}{\tau^2\epsilon}$ *and* p *is a polynomial with agreement* τ *with* f. *Then, for a random pair* $\mathbf{z}, \mathbf{a} \in \mathbb{F}_q^m$, $A_{\mathbf{z},p(\mathbf{z})}$ *outputs* $p(\mathbf{a})$ *with probability at least* $1 - \epsilon$.

Proof As in previous proofs, we describe some bad events and then claim that if none of the bad events occur, then the algorithm $A_{z,p(\mathbf{z})}$ outputs $p(\mathbf{a})$. Our first bad event B corresponds the bad events of previous proofs, i.e., to poor agreement between p and f on ℓ. Specifically B is the event that "p and f have less than $\tau/2$ agreement on ℓ". We now describe the second bad event: Let $h_1, \ldots, h_n$ be all univariate polynomials that have $\tau/2$ agreement with $f|_\ell$. Let C be the event that there exists a pair $1 \leq i < j \leq n$ such that $h_i\{\mathbf{z}\} = h_j\{\mathbf{z}\}$. We show below that if neither B nor C occurs, then the algorithm $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$. Later we give upper bounds on the probabilities of B and C.

Claim 12 *If neither of the events B or C occurs, then $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$ on input $\mathbf{a}$.*

Proof This is relatively straightforward. Since the event B does not occur, we have that the polynomial $p|_\ell$ has at least $\tau/2$ agreement with f on the line ℓ. Thus one of the polynomials $h_1, \ldots, h_n$ computed by $A_{\mathbf{z},p(\mathbf{z})}$ in Step 3 is $p|_\ell$. Say it is the polynomial h_i. Then $h_i\{\mathbf{z}\} = p(\mathbf{z})$. But since the event C did not occur, we know that $h_j\{\mathbf{z}\} \neq h_i\{\mathbf{z}\}$ for any other index j. Thus h_i is the unique polynomial satisfying the condition of Step 4 and thus Step 4 results in the output $h_i(0) = p(\mathbf{a})$. ■

Claim 13 *The probability of event B, taken over the choices of $\mathbf{z}$ and $\mathbf{a}$, is at most $\frac{4}{\tau q}$.*

Proof This is a simple application of the Chernoff bound (see first lecture). By Proposition 4, the points of the line ℓ are distributed uniformly over $\mathbb{F}_q^m$ and pairwise independent. On any one point, the probability that f agrees with p is τ. The expected number of agreements between f and p on q pairwise independent points is thus τq. The probability that this number deviates from its expectation by half the expectation (a requirement for the event B) is bounded by $\frac{4}{\tau q}$. ■

Claim 14 *The probability of event C, taken over the choice of $\mathbf{z}$ and $\mathbf{a}$, is at most $\frac{8d}{\tau^2 q}$, provided $\tau > 2\sqrt{d/q}$.*

Proof The claim would be obvious, following immediately from Proposition 10, if $\mathbf{z}$ was chosen to be a random point on line ℓ after the line is fixed. But this is not the case! Or is it?

In the way the algorithm is described, $\mathbf{a}$ and $\mathbf{z}$ are chosen first and then ℓ is defined based on them. However we could pick ℓ at random first, as a *set* and then $\mathbf{a}$ and $\mathbf{z}$ to be two random (distinct, assuming ℓ turns out to be non-degenerate) points on it later. Note that the polynomials $h_1, \ldots, h_n$ are already well-defined once the line is chosen as a *set*, without fixing the parameterization. (They are not fixed in terms of their coefficients; but they are fixed as functions from ℓ to $\mathbb{F}_q$.) Thus the probability, when we pick $\mathbf{z}$ at random on ℓ that $h_i\{\mathbf{z}\} = h_j\{\mathbf{z}\}$ for some distinct pair i, j is at most $\frac{8d}{\tau^2 q}$ (applying Proposition 10 in the univariate case with agreement set to $\tau/2$). The claim follows. ■

We are now ready to glue together the proof of the lemma. We will pick q large enough so that the two events above happen with probability at most $\epsilon/2$ each. Thus we get the condition $q \geq \max\{\frac{8}{\tau\epsilon}, \frac{16d}{\tau^2\epsilon}\}$. To make this simpler we set $q \geq \frac{16(d+1)}{\tau^2\epsilon}$. Once we have this condition we find that the probability that B or C occurs is at most ϵ and with the remaining probability $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$. ■

Now we describe the actual list-decoding algorithm. The algorithm simply picks a collection of random points $\mathbf{z}_1, \ldots, \mathbf{z}_t$'s and enumerates all possible choices of $\gamma \in \mathbb{F}_q$ for $p(\mathbf{z}_i)$. It then applies the algorithm **Improved RM decoder** to the functions $A_{\mathbf{z}_i,\gamma}$. For every degree d polynomial p that has agreement τ with f, one of the algorithms above computes p. Again we summarize this formally below:

RM List-decoder

Given: Oracle access to $f : \mathbb{F}_q^m \to \mathbb{F}_q$. Point $\mathbf{a} \in \mathbb{F}_q^m$, agreement parameter τ and degree parameter d.

Goal: To produce a list of randomized algorithms computing functions $g_1, \ldots, g_n : \mathbb{F}_q^m \to \mathbb{F}_q$ that includes every degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ with τ agreement with f.

Step 1: Compute parameters $\epsilon = \frac{1}{15}$, $t = \log \frac{4}{\tau}$ and $n = tq$.

Step 2: Pick $\mathbf{z}_1, \ldots, \mathbf{z}_t$ independently and uniformly at random from $\mathbb{F}_q^m$.

Step 3: For every $i \in [t]$ and $\gamma \in \mathbb{F}_q$ output the algorithm **Improved RM Decoder** accessing the oracle $A_{\mathbf{z}_i,\gamma}$.

The following lemma analyzes the correctness of the algorithm.

Lemma 15 *With probability at least $\frac{1}{2}$, the algorithms constructed by* **RM List-Decoder** *include one for every polynomial p that has agreement τ with f, provided $q \geq \frac{240(d+1)}{\tau^2}$.*

Proof Let $p_1, \ldots, p_m$ be all polynomials of degree at most d that have agreement at least τ with f. Note, by the usual Johnson bounding argument, that $m \leq \frac{2}{\tau}$. Fix $i \in [m]$ and let $p = p_i$. We prove that the event that none of the algorithms constructed by **RM List-decoder** turns out to be a randomized algorithm computing p occurs with probability at most $2^{-t} \leq \frac{\tau}{4}$ (this inequality is the one that leads to the setting of t in the algorithm). By the union bound it follows that all the required polynomials are in the output list with probability at least $\frac{1}{2}$.

Fix $j \in [t]$. Note that the choice of q ensures that Lemma 11 can be applied for $\epsilon = \frac{1}{15}$. By Lemma 11, $A_{\mathbf{z}_j,p(\mathbf{z}_j}$ computes $p(\mathbf{a}')$ with probability at least $1 - \epsilon$ for random input $\mathbf{a}'$. Thus, with probability at least $\frac{1}{2}$ it must be the case that $\mathbf{z}_j$ is such that $A_{\mathbf{z}_j,p(\mathbf{z}_j}$ has agreement at least $1 - 2\epsilon$ with p. Since $2\epsilon \leq 2/15$, we have that **Improved RM Decoder** computes $p(\mathbf{a})$ correctly on every input $\mathbf{a}$ with probability at least $2/3$. Thus with probability at most $\frac{1}{2}$, the algorithm **Improved RM Decoder** with oracle access to $A_{\mathbf{z}_j,p(\mathbf{z}_j)}$ does *not* compute p. The probability that this happens for every $j \in [t]$ is thus at most 2^{-t}. ■

5 Formal Model and Theorems

To encapsulate the results of this lecture in the strongest possible way, we ought to formalize the model that is being developed to represent words and decoding algorithms. The main feature we wish to focus on is the *extreme efficiency* of the algorithms in some *implicit* model of the inputs, outputs, decoding and list-decoding. We describe these in order.

In implicit decoding problems, both inputs and outputs will be represented implicitly. In particular, the received vector will be given by an oracle. A vector $\mathbf{v} = \langle v_1, \ldots, v_n \rangle \in \mathbb{F}_q^n$ is specified *implicitly*

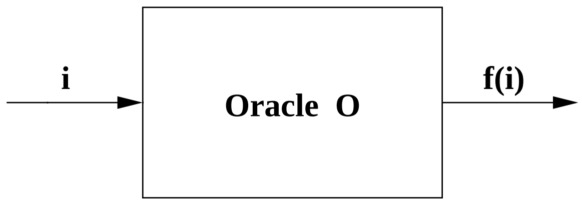

Figure 1: Implicit representation of input

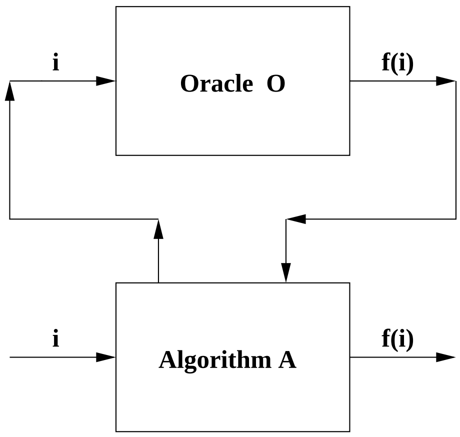

Figure 2: Implicit representation of output

by an oracle O for a function $f : [n] \to \mathbb{F}_q$. Given an index $i \in [n]$ the oracle responds with the value $f(i) = v_i$. Since the vector $\mathbf{v}$, the oracle O, and the function f represent the same information (though they carry different meanings as computational entities) we will save ourselves some symbols and use the function to represent all three objects. Figure 1 gives a pictorial view of implicit inputs.

The implicit representation of outputs requires some care. A first attempt may be to suggest exactly the same definition as used for implicit inputs, i.e., by oracles. But that does not appear to be constructive! What does it mean to say we will simply output a box — we need to be more explicit. How does the box compute its functions. So really we should be representing outputs by algorithms to compute them. But then if the output contains k bits of information, then it is impossible to produce a small algorithm (of length smaller than k) that can give out these k bits of information. Yet, we have seen above that it is possible to decode very efficiently, much more efficiently than the time it would take to write down the output. How did we acheive that? Close examination reveals that our outputs were actually being specified relative to the input. Specifically, we produced an algorithm to compute the output vector that made oracle calls to the input oracle. (See Figure 2.) Strictly speaking all we describe is the algorithm A (pictured in Figure 3) that, when given oracle access to the input f, computes the output word. These kind of objects are referred to in the computer science literature as probabilistic oracle machines and play a significant role in some of the major developments there.

We now formally describe the implicit decoding problem:

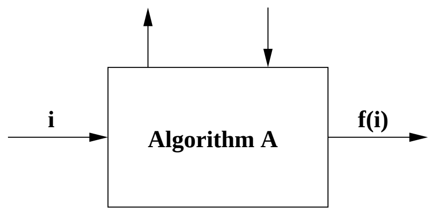

Figure 3: The actual implicit output

Definition 16 (Implicit unambiguous decoding) *For a fixed family of codes $\mathcal{C}$, the implicit decoding problem is:*
GIVEN: *Parameters n, q (and any other parameters that are relevant) of the code $C \in \mathcal{C}$, error parameter $e < \frac{\Delta(C)}{2}$, and implicit representation of received vector $\mathbf{f} \in \mathbb{F}_q^n$.*
GOAL: *Output an implicit representation A of codeword $\mathbf{c} \in C$ such that $\Delta(\mathbf{c}, \mathbf{f}) \leq e$, if such a codeword exists. Specifically, A takes as input $i \in [n]$ and outputs c_i, the ith coordinate of $\mathbf{c}$.*

Remarks:

1. The running time of the decoder is defined to be the sum of the the time taken to compute a description of A and the worst-case running time of A.
2. A is allowed to be randomized, in which case it must produce the correct answer (for every coordinate) with probability at least $\frac{2}{3}$ (as is standard for randomized algorithms).
3. If the algorithm runs in time $t(n)$ and uses $r(n)$ random bits on codes of length n, then it can be converted into a randomized algorithm producing the entire codeword in time $O(nt(n)\log n)$ by running it $O(\log n)$ times on each coordinate and taking majority votes and outputting the string so obtained. It can also be converted into a deterministic decoding algorithm running in time $O(n2^{r(n)}t(n))$ by simulating all random strings of length $r(n)$.

The following theorem sums up the unambiguous decoding results for Reed-Muller codes that we obtained in this lecture.

Theorem 17 *For a Reed-Muller code $\mathrm{RM}_{m,d,q}$ with $q > 5(d+1)$, the algorithm* **Improved RM decoder** *is a randomized algorithm that corrects upto a fraction $\frac{2}{15}$ error in time $\mathrm{poly}(m, d, \log q)$.*

The list-decoding version can be defined analogously. The list-decoding algorithm should output implicit representations of the ℓ-nearby codewords. We will be slightly more relaxed and allow the algorithm to output $\ell' \geq \ell$ algorithms, with the condition that every nearby codeword be represented (as also other algorithms that don't correspond to codewords). Thus we obtain the following problem:

Definition 18 (Implicit list decoding) *For a fixed family of codes $\mathcal{C}$, the implicit decoding problem is:*
GIVEN: *Parameters n, q (and any other parameters that are relevant) of the code $C \in \mathcal{C}$, error parameter $e < \frac{\Delta(C)}{2}$, and implicit representation of received vector $\mathbf{f} \in \mathbb{F}_q^n$.*
GOAL: *Output ℓ' algorithms $A_1, \ldots, A_{\ell'}$ such that for every codeword $\mathbf{c} \in C$ such that $\Delta(\mathbf{c}, \mathbf{f}) \leq e$ there exists an index $i \in [\ell']$ such that A_i with oracle access to $\mathbf{f}$ is an algorithm to compute $\mathbf{c}$.*

In terms of this definition, we can summarize our list-decoding algorithm as follows:

Theorem 19 *For a Reed-Muller code* $\mathrm{RM}_{m,d,q}$ *with* $q > \frac{240(d+1)}{\gamma^2}$*, for some* $\gamma > 0$*, the algorithm* **RM List-decoder** *is a randomized algorithm that corrects upto a fraction* $1-\gamma$ *error in time* $\mathrm{poly}(m,d,q)$.

6 Bibliographic notes

We start with the implicit models discussed in the last section. It is hard to pin down the exact point where these models were first introduced. Often they were implicit in some technical results and then made explicit only by later works. They also have multiple origins within computer science — several works in the late eighties seem to converge on these models independently. One of the first works that used such models is that of Kaltofen and Trager [15], who applied it in the context of algebraic algorithms. This work explicitly focusses on the implicit models (ahem!) and thereby captured the efficiency of several algorithms very elegantly. Around the same time, the implicit input model was introduced into coding theory by Goldreich and Levin [12]. In the language of coding theory, they developed a list-decoding algorithm (the first non-trivial list-decoder!) for the Hadamard code in the implicit input model. They also showed a powerful application of list-decoding to the foundations of cryptography. (We might talk about this result in a later lecture.) Also, at the same time, such models were exploited to check or improve correctness of programs by Blum and Kannan [8] and Blum, Luby, and Rubinfeld [9]. As it turned out, such results could be interpreted as error-correction algorithms for some codes [21, Section 1.3]. Finally, the full-fledged model (implicit input and output, with list-decoding capabilities) were considered in Ar et al. [1] and Arora and Sudan [4], and made explicit in Sudan, Trevisan, and Vadhan [22].

The algorithms have a somewhat simpler history. (As always, notions are harder to trace than algorithms!) The simple algorithm for decoding Reed-Muller codes from Section 2 is based on an algorithm of Beaver and Feigenbaum [7], whose ability to decode all polynomials was pointed out by Lipton [16]. The improvement in Section 3 is due to Gemmell et al. [10]. Further improvements to this algorithm, decoding arbitrarily close to half the minimum distance for $q \gg d$ were given by Gemmell and Sudan [11]. The list-decoding algorithm in Section 4 is due to Sudan, Trevisan, and Vadhan [22]. This algorithm simplifies a previous list-decoding algorithm of Arora and Sudan [4].

References

[1] Sigal Ar, Richard Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, 28(2):488–511, 1999.

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

[3] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

[4] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, 4-6 May 1997.

[5] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[6] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[7] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In C. Choffrut and T. Lengauer, editors, *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, pages 37–48, Rouen, France, 22–24 February 1990. Springer.

[8] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, January 1995.

[9] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[10] Peter Gemmell, Richard Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 32–42, New Orleans, Louisiana, 6-8 May 1991.

[11] Peter Gemmell and Madhu Sudan. Highly resilient correctors for multivariate polynomials. *Information Processing Letters*, 43(4):169–174, September 1992.

[12] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15-17 May 1989.

[13] Tom Høholdt, J. H. van Lint, and Ruud Pellikaan. Algebraic geometry codes. *Handbook of Coding Theory, Chapter 10*, 1998.

[14] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR Lemma. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 220–229, May 1997.

[15] Erich Kaltofen and Barry Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.

[16] Richard Lipton. New directions in testing. In *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. AMS, 1991.

[17] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

[18] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[19] Vera S. Pless and W. Cary Huffman (Eds.). *Handbook of Coding Theory (2 Volumes)*. Elsevier, 1998.

[20] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.

[21] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*. PhD thesis, University of California at Berkeley, October 1992. Also appears as *Lecture Notes in Computer Science*, vol. 1001, Springer, 1996.

[22] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.