Lecture 15

Lecturer: Madhu Sudan *Scribe: Yael Tauman*

1 Introduction

In this lecture, we present a local decoding algorithm and a local list-decoding algorithm for Reed-Muller codes. These algorithms are very interesting, since they played a significant role in many developments in complexity theory over the past two decades. For example, these algorithms were used in the following results.

- "The permanent of a matrix is hard to compute on random matrices", due to Lipton [11]. (This result makes use of a simple unambiguous decoding algorithm for Reed-Muller codes implicit, in Beaver and Feigenbaum [6]).
- "IP = PSPACE [12, 14]", "MIP=PSPACE [4]" and the "PCP Theorem [2, 1]".
- The sequence of results showing progressively weaker complexity-theoretic conditions that would suffice to show "BPP = P" [13, 5, 10].

The algorithms in today's lecture are very simple, and do not exploit any serious algebraic property. The only algebraic elements that the algorithms use are decoding algorithms for Reed-Solomon codes, and we'll just adopt them as a black box from a previous lecture.

We'll describe the algorithms in three steps - first we give an algorithm that decodes from a very low error rate. Then we show how, using a Reed-Solomon decoder, one can recover from a slightly larger error rate. Finally, we jump to a list-decoder, which uses a Reed-Solomon list-decoder, and can recover from a much larger error rate.

Let us first recall the definition of Reed-Muller codes (defined in lecture 4). 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 $d \leq q$ (follows from the "Schwartz Lemma").

For today's lecture, it is convenient to think of the decoding task as a "function reconstruction task" rather than as a 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 codeword (or a list of all nearby codewords), where the codewords are also functions $p : \mathbb{F}_q^m \to \mathbb{F}_q$. For any two functions f and g, let

$$\delta(f, g) = \Pr_{\mathbf{x} \in \mathbb{F}_q^m} [f(x) \neq g(x)].$$

Note that, if f and g are interpreted as strings rather than functions, this is just the standard Hamming distance normalized so as to be between 0 and 1.

> **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 $\delta(f, p) \leq \delta$.

We will present randomized algorithms that, given any vector $\mathbf{a} \in \mathbb{F}_q^m$, compute $p(\mathbf{a})$ in time $\mathrm{poly}(m, d, q)$. Note that the length of the codewords is $\binom{m+d}{d}$, which could be exponentially larger than $\mathrm{poly}(m, d, q)$, and hence the our algorithms are much more efficient than explicit decoding algorithms.

2 Decoding from very low error

We start with describing an extremely simple randomized algorithm that recovers from a very low error rate δ. The amount of error will certainly be small enough to put us in the case of an 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 $\mathbb{F}_q^m$.

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:*

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

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

Let us point out two nice properties of lines.

1. **Randomness Property:** The first nice property of a line is its randomness property.
   - (Pairwise Independence): 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 in $\mathbb{F}_q^m$.
   - (1-wise Independence): 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 $\mathbb{F}_q^m$.

2. **Algebraic Property**: This property alludes to the restriction 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 $f|_\ell(t) = f(\ell(t))$.
   - 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.

2.2 The algorithm

Our first algorithm for decoding Reed-Muller codes is an elementary algorithm that uses the above two properties of lines in $\mathbb{F}_q^m$. This algorithm, given oracle access to a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$ that is very close to a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$, and given an element $\mathbf{a} \in \mathbb{F}_q^m$, outputs $p(\mathbf{a})$ 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 $p|_\ell$. From the algebraic property, $p|_\ell$ is a univariate polynomial of degree d. ¿From the randomness property, $p|_\ell$ and $f|_\ell$ have good agreement. So $p|_\ell$ can hopefully be recovered efficiently. 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 idea. In this form, the algorithm only needs $q \geq d+2$ to work. However, the amount of error it will correct is quite small.

Simple RM decoder:

Given: Oracle access to $f : \mathbb{F}_q^m \to \mathbb{F}_q$, a point $\mathbf{a} \in \mathbb{F}_q^m$ and a degree 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 in $\mathbb{F}_q - \{0\}$. 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 $q \geq d+2$. We will show below that this requirement suffices for correctness, provided the error is small enough.

Lemma 3 *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 $1 - (d+1)\delta$. Note that the Lemma follows once this 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 $\mathbb{F}_q$. Furthermore, the algebraic property implies that $p|_\ell$ is a polynomial of degree at most d. Thus, the polynomial h found in Step 3 *is* the function $p|_\ell$, and hence the value output in Step 4 is $p|_\ell(0) = p(\ell(0)) = p(\mathbf{a})$. So it suffices to bound the probability of the bad events.

Note that B_i happens if and only if $f(\ell(\alpha_i)) \neq p(\ell(\alpha_i))$. The randomness property implies that $\ell(\alpha_i)$ is a random point in $\mathbb{F}_q^m$ and hence 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 least $1-(d+1)\delta$. 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.

Note that with the following choice of parameters, we get a Reed-Muller code with polynomial rate and constant distance, that can correct a poly-logarithmic fraction of errors.

- $d = (\lg k)^c$
- $m = \frac{\lg k}{(c-1)\lg\lg k}$

  (Verify that $\binom{d+m}{m} \approx k$).
- $q = 2(d+2)$
- $n = q^m \approx (2d)^m \approx k^{1+\frac{1}{c-1}}$

3 Improving the error-correction capability

We would like to improve the error-correction capability of the simple algorithm described in the previous section. Increasing the error-correction capability comes at a price: The requirement on the field size goes up, and the algorithm gets slightly more complicated.

To motivate the basic idea behind the improvement, let us look back at 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 depends on the rate of error we wish to correct, on the running time we desire, and on the field size. We will set this number, somewhat arbitrarily, to $5(d+1)$ and demonstrate the effect.

Improved RM decoder:

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

Promise: There exists a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that $\delta = \delta(p, f) < \frac{1}{5}$.

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 in $\mathbb{F}_q - \{0\}$. 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 an unambiguous Reed-Solomon decoding algorithm (such as the Welch-Berlekamp algorithm described in a previous lecture). We thus get the following proposition.

Proposition 4 **Improved RM decoder** *runs in* $\mathrm{poly}(d, m)$ *time.*

Lemma 5 *Under the assumption* $\delta = \delta(f, p) < \frac{1}{5}$*, the algorithm* **Simple RM decoder** *outputs* $p(\mathbf{a})$*, with probability greater than* $\frac{1}{2}$*.*

Proof As in the proof of Lemma 3, 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 $5\delta(d+1)/2(d+1) = 5\delta/2$. 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, and thus, in Step 4 we output $p|_\ell(0) = p(\mathbf{a})$. Hence, to achieve error probability less than $\frac{1}{2}$, we need $5\delta/2 < 1/2$, which happens for $\delta < 1/5$. ■

The ideas in the algorithm above can be pushed further to get a decoding algorithm for $\delta < 1/4$. It is known that we cannot achieve a decoding algorithm for $\delta \geq 1/4$, using Markov's inequality.

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 3). If we did, we would be able to find polynomials with much smaller agreement with f on any given line ℓ. However, 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, 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.

We will solve this problem by focussing on one of the specific polynomials p that is close to f. This will be done by giving a small amount of additional information, called *advice*, that suffices to specify it uniquely. We will then give a reconstruction procedure that 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 that with high probability, 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, with high probability, the value of a polynomial at a random point specifies it uniquely, given a nearby function f.

Proposition 6 *Let $p_1, \ldots, p_n$ be a list of all m-variate degree d polynomials over $\mathbb{F}_q$ satisfying $\tau(f, p_i) \geq \tau$. If $\tau \geq \sqrt{2d/q}$ then $n \leq 2/\tau$, and with probability at least $1 - \binom{n}{2}(\frac{d}{q}) \geq 1 - \frac{2d}{\tau^2 q}$ 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 $\frac{d}{q}$ (by the "Schwartz Lemma"). ■

This motivates the idea 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 section 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 $\mathbf{z}$! The algorithm is described for a fixed choice of $\mathbf{z}$ and advice γ.

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

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

Step 1: Let $\mathbf{b} = \mathbf{z} - \mathbf{a}$ and let $\ell = \ell_{\mathbf{a},\mathbf{b}}$. (Note that $\ell(0) = \mathbf{a}$ and $\ell(1) = \mathbf{z}$).

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

Step 3: 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 require a polynomial running time, provided the parameters are favorable. In particular, Step 2 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 7 *If $\tau > \sqrt{4d/q}$ then the subroutine $A_{\mathbf{z},\gamma}$ runs in time* $\mathrm{poly}(q, m)$.

Next we analyze the correctness of the algorithm. Note that $A_{\mathbf{z},\gamma}$ is a deterministic algorithm. We will analyze it only for a random $\mathbf{a}$. That is, We show that for a random pair $(\mathbf{z}, \mathbf{a})$, if $\tau(p, f) \geq \tau$, then 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 8 *Let p be a polynomial which agrees with f on a τ fraction of the inputs. Then, for any $\epsilon > 0$, and for random $\mathbf{z}, \mathbf{a} \in \mathbb{F}_q^m$, $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$ with probability at least $1 - \epsilon$, assuming $q \geq \frac{16(d+1)}{\tau^2 \epsilon}$.*

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 is the event that "p and f have less than $\tau/2$ agreement on ℓ". Let $h_1, \ldots, h_n$ be all univariate polynomials that have $\tau/2$ agreement with $f|_\ell$. The second bad event, C, is 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 9 *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 2 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 3 and hence Step 3 results in the output $h_i(0) = p(\mathbf{a})$. ■

Claim 10 *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 Chebychev inequality. Since ℓ is a random line (for random $\mathbf{z}, \mathbf{a} \in \mathbb{F}_q^m$), the randomness property implies that the points in the line ℓ are distributed uniformly over $\mathbb{F}_q^m$ and are pairwise independent. On any one point, the probability that f agrees with p is τ. Thus, the expected number of agreements between f and p on all q points in ℓ is τq. Since these q points are pairwise independent, using the Chebychev inequality, we derive that the probability that this number deviates from its expectation by half the expectation is bounded by $\frac{4}{\tau q}$. ■

Claim 11 *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 6, 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, and then choose $\mathbf{a}$ and $\mathbf{z}$ at random from ℓ. Let $h_1, \ldots, h_n$ be all the univariate polynomials that have $\frac{\eta}{2}$ agreement with $f|_\ell$. Thus, the probability, when we pick $\mathbf{z}$ at random from 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 6 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 $p(\mathbf{a})$. ■

5 Bibliographic Notes

The simple algorithm for decoding Reed-Muller codes from Section 2 is based on an algorithm of Beaver and Feigenbaum [6], whose ability to decode all polynomials was pointed out by Lipton [11]. The improvement in Section 3 is due to Gemmell et al. [8]. Further improvements to this algorithm, decoding arbitrarily close to half the minimum distance for $q \gg d$ were given by Gemmell and Sudan [9]. The list-decoding algorithm in Section 4 is due to Sudan, Trevisan, and Vadhan [15]. This algorithm simplifies a previous list-decoding algorithm of Arora and Sudan [3].

References

[1] 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.

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

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

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

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

[15] 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.