Introduction to Coding Theory **CMU: Spring 2010**

Notes 6: Reed-Solomon, BCH, Reed-Muller, and concatenated codes

February 2010

Lecturer: Venkatesan Guruswami *Scribe: Eric Blais & Venkat Guruswami*

In this lecture, we begin the algorithmic component of the course by introducing some explicit families of good algebraic codes. We begin by looking at Reed-Solomon codes.

1 Reed-Solomon codes

Reed-Solomon codes are a family of codes defined over large fields as follows.

Definition 1 (Reed-Solomon codes) *For integers $1 \leq k < n$, a field $\mathbb{F}$ of size $|\mathbb{F}| \geq n$, and a set $S = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}$, we define the Reed-Solomon code*

$$\mathrm{RS}_{\mathbb{F},S}[n,k] = \{\, (p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n)) \in \mathbb{F}^n \mid p \in \mathbb{F}[X] \text{ is a polynomial of degree } \leq k-1 \,\}.$$

A natural interpretation of the $\mathrm{RS}_{\mathbb{F},S}[n,k]$ code is via its encoding map. To encode a message $m = (m_0, m_1, \ldots, m_{k-1}) \in \mathbb{F}^k$, we interpret the message as the polynomial

$$p(X) = m_0 + m_1 X + \cdots + m_{k-1} X^{k-1} \in \mathbb{F}[X].$$

We then evaluate the polynomial p at the points $\alpha_1, \alpha_2, \ldots, \alpha_n$ to get the codeword corresponding to m.

To evaluate the polynomial p on the points $\alpha_1, \alpha_2, \ldots, \alpha_n$, we multiply the message vector m on the left by the $n \times k$ Vandermonde matrix

$$G = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{k-1} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{k-1} \end{pmatrix}.$$

The matrix G is a generator matrix for $\mathrm{RS}_{\mathbb{F},S}[n,k]$, so we immediately obtain that Reed-Solomon codes are linear codes over $\mathbb{F}$.

1.1 Properties of the code

Let's now examine the parameters of the above Reed-Solomon code. The block length of the code is clearly n. As we will see, the code $\mathrm{RS}_{\mathbb{F},S}[n,k]$ has minimum distance $n-k+1$. This also means that the encoding map is injective and therefore the code has dimension equal to k.

The key to establishing the minimum distance of Reed-Solomon codes is the 'degree mantra' that we saw in the previous lecture: *A non-zero polynomial of degree d with coefficients from a field $\mathbb{F}$ has at most d roots in $\mathbb{F}$.*

Theorem 2 *The Reed-Solomon code $\mathrm{RS}_{\mathbb{F},S}[n,k]$ has distance $n-k+1$.*

PROOF: Since $\mathrm{RS}_{\mathbb{F},S}[n,k]$ is a linear code, to prove the theorem it suffices to show that any non-zero codeword has Hamming weight at least $n-k+1$.

Let $(m_0, m_1, \ldots, m_{k-1}) \neq 0$. The polynomial $p(X) = m_0 + m_1 X + \cdots + m_{k-1}X^{k-1}$ is a non-zero polynomial of degree at most $k-1$. So by our degree mantra, p has at most $k-1$ roots, which implies that $(p(\alpha_1), \ldots, p(\alpha_n))$ has at most $k-1$ zeros.

By the Singleton bound, the distance cannot exceed $n-k+1$, and therefore must equal $n-k+1$. The upper bound on distance can also be seen by noting that the codeword corresponding to the polynomial $\prod_{i=1}^{k-1}(X-\alpha_i)$ has Hamming weight exactly $n-k+1$. □

Note that the minimum distance of Reed-Solomon codes meets the Singleton bound. This is quite interesting: Reed-Solomon codes are a simple, natural family of codes based only on univariate polynomials, and yet their rate is optimal.

In our definition above, we have presented Reed-Solomon codes in the most general setting, where S can be any arbitrary subset of $\mathbb{F}$ of size n. This presentation highlights the flexibility of Reed-Solomon codes. In practice, however, there are two common choices of S used to instantiate Reed-Solomon codes:

1. Take $S = \mathbb{F}$, or
2. Take $S = \mathbb{F}^*$ to be the set of non-zero elements in $\mathbb{F}$.

These two choices attain the best possible trade-off between the field size and the block length.

1.2 Alternative characterization

We presented Reed-Solomon codes from an encoding point of view. It is also possible to look at these codes from the "parity-check" point of view. This approach is used in many textbooks, and leads to the following characterization of Reed-Solomon codes.

Theorem 3 (Parity-check characterization) *For integers $1 \leq k < n$, a field $\mathbb{F}$ of size $|\mathbb{F}| = q = n+1$, a primitive element $\alpha \in \mathbb{F}^*$, and the set $S = \{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$, the Reed-Solomon code over $\mathbb{F}$ with evaluation set S is given by*

$$\mathrm{RS}_{\mathbb{F},S}[n,k] = \{\, (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}^n \mid c(X) = c_0 + c_1 X + \cdots + c_{n-1}X^{n-1} \text{ satisfies } c(\alpha) = c(\alpha^2) = \cdots = c(\alpha^{n-k}) = 0 \,\}. \quad (1)$$

In other words, Theorem 3 states that the codewords of the Reed-Solomon code with evaluation points $1, \alpha, \ldots, \alpha^{n-1}$ correspond to the polynomials of degree $n-1$ that vanish at the points $\alpha, \alpha^2, \ldots, \alpha^{n-k}$.

The characterization of Reed-Solomon codes in Theorem 3 has the same dimension as the code obtained with our original definition; to complete the proof of Theorem 3, we only need to check that every codeword obtained in Definition 1 satisfies the parity-check condition (1).

Exercise 1 *Complete the proof of Theorem 3. (Hint: The proof uses the fact that for every $x \neq 1$ in $\mathbb{F}^*$, $\sum_{\ell=0}^{n-1} x^\ell = \frac{1-x^n}{1-x} = 0$.)*

1.3 Applications

Reed-Solomon codes were originally introduced by Reed and Solomon in 1960 [6]. There have been many other codes introduced since – we will see some of those more recent codes soon – and yet Reed-Solomon codes continue to be used in many applications. Most notably, they are extensively used in storage devices like CDs, DVDs, and hard-drives.

Why are Reed-Solomon codes still so popular? One important reason is because they are optimal codes. But they do have one downside: Reed-Solomon codes require a large alphabet size. In a way, that is unavoidable; as we saw in Notes 4, any code that achieves the Singleton bound must be defined over a large alphabet.

The large alphabet brings to the fore an important issue: if we operate on bits, how do we convert the codewords over the large field in the binary alphabet? There is one obvious method. Say, for example, that we have a code defined over $\mathbb{F}_{256}$. Then we can write an element in this field as an 8-bit vector.

More precisely: if we have a message that corresponds to the polynomial $p(X)$ in $\mathbb{F}[X]$, its encoding in the Reed-Solomon code is the set of values $p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_m)$. We can simply express these values in a binary alphabet with $\log|\mathbb{F}|$ bits each. So provided that the Reed-Solomon code is defined over a field that is an extension field of $\mathbb{F}_2$, then this simple transformation yields a code over $\mathbb{F}_2$. In fact, there is way to represent field elements as bit vectors so that the resulting code is a binary *linear* code.

This method is in fact what is done in practice. But then it leads to the natural question: What are the error correction capabilities of the resulting binary code?

Let's look at an example: say we have a Reed-Solomon code with $n = 256$ and $k = 230$. The distance of this code is $d = 27$, so the code can correct 13 errors. The transformation to a binary code yields a binary code where $n' = 256 \cdot 8$ and $k' = 230 \cdot 8$, since all we have done in the transformation is scale everything. And at worst the distance of the resulting binary code is $d' \geq d = 27$, so the binary code can also correct at least 13 errors.

Let us now generalize the example. If we have a $[N, K, D]_{\mathbb{F}}$ code where $|\mathbb{F}| = N$ and N is a power of 2, then the transformation described above yields a $[N \log N, K \log N, D']_2$ binary linear code, where $D' \geq D$. Writing $n = N \log N$ and considering the case where $K = N - D + 1$, we observe that the transformation of a Reed-Solomon code to a binary code results in a $[n, n-(D-1)\log N, \geq D]_2$ code.

The resulting binary code has a decent rate, but it is not optimal: *BCH codes* are even better, as they are $[n, n - \lceil \frac{D-1}{2} \rceil \log(n+1), \geq D]_2$ codes. BCH codes are very interesting in their own right, and we will examine them in the next section. But first we return to the question that we posed at

the beginning of this section: why are Reed-Solomon codes still so popular? If BCH codes have the same distance guarantees as Reed-Solomon codes and a better rate, one would expect these codes to have completely replaced Reed-Solomon codes.

The main reason that Reed-Solomon are still frequently used is that in many applications – and in particular in storage device applications – errors often occur in bursts. Reed-Solomon codes have the nice property that bursts of consecutive errors affect bits that correspond to a much smaller number of elements in the field on which the Reed-Solomon code is defined. For example, if a binary code constructed from the $\mathrm{RS}_{\mathbb{F}_{256}}[256, 230]$ code is hit with 30 consecutive errors, these errors affect at most 5 elements in the field $\mathbb{F}_{256}$ and this error is easily corrected.

2 BCH codes

BCH codes were discovered by independently by Bose and Ray-Chaudhuri [1] and by Hocquenghem [3] in the late 1950s. As we saw in the previous section, BCH codes have better rate than binary codes constructed from Reed-Solomon codes. In fact, as we will see later in the section, the rate of BCH codes is optimal, up to lower order terms.

BCH codes can be defined over any field, but for today's lecture we will focus on binary BCH codes:

Definition 4 (Binary BCH codes) *For a length $n = 2^m - 1$, a distance D, and a primitive element $\alpha \in \mathbb{F}_{2^m}^*$, we define the binary BCH code*

$$\mathrm{BCH}[n, D] = \{\, (c_0, \ldots, c_{n-1}) \in \mathbb{F}_2^n \mid c(X) = c_0 + c_1 X + \cdots + c_{n-1} X^{n-1} \text{ satisfies } c(\alpha) = c(\alpha^2) = \cdots = c(\alpha^{D-1}) = 0 \,\}.$$

This definition should look familiar: it is almost exactly the same as the alternative characterization of Reed-Solomon codes in Theorem 3. There is one important difference: in Theorem 3, the coefficients $c_0, \ldots, c_{n-1}$ could take any value in the extension field, whereas here we restrict the coefficients to take values only from the base field (i.e., the coefficients each take values from $\mathbb{F}_2$ instead of $\mathbb{F}_{2^m}$).

The BCH codes form linear spaces. The definition gives the parity-check view of the linear space, as it defines the constraints over the elements. The constraint $c(\alpha) = 0$ is a constraint over the extension field $\mathbb{F}_{2^m}$, but it can also be viewed as a set of m linear constraints over $\mathbb{F}_2$.

The last statement deserves some justification. That each constraint over $\mathbb{F}_{2^m}$ corresponds to m constraints over $\mathbb{F}_2$ is clear from the vector space view of extension fields. That the resulting constraints are linear is not as obvious but follows from the argument below.

Consider the (multiplication) transformation $\mathrm{Mult}_\alpha : x \mapsto \alpha x$ defined on $\mathbb{F}_{2^m}$. This map is $\mathbb{F}_2$-linear, since $\alpha(x + y) = \alpha x + \alpha y$. Using the additive vector space structure of $\mathbb{F}_{2^m}$, we can pick a basis $\{\beta_1 = 1, \beta_2, \ldots, \beta_m\} \subset \mathbb{F}_{2^m}$ of $\mathbb{F}_{2^m}$ over $\mathbb{F}$, and represent each element $x \in \mathbb{F}_{2^m}$ as the (column) vector $(x_1, x_2, \ldots, x_m)^T \in \mathbb{F}_2^m$ where $x = x_1\beta_1 + x_2\beta_2 + \cdots + x_m\beta_m$. The $\mathbb{F}_2$-linear multiplication map Mult_α then corresponds to a linear transformation of this vector representation, mapping $x = (x_1, \ldots, x_m)^T$ to $M_\alpha x$ for a matrix $M_\alpha \in \mathbb{F}_2^{m \times m}$. And the coefficients $c_i \in \mathbb{F}_2$ correspond to vectors $(c_i, 0, \ldots, 0)^T \in \mathbb{F}_2^m$ so the constraint $c(\alpha) = c_0 + c_1\alpha + c_2\alpha^2 + \cdots + c_{n-1}\alpha^{n-1} = 0$ is

equivalent to the constraint

$$\begin{pmatrix} c_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + M_\alpha \begin{pmatrix} c_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \cdots + M_\alpha^{n-1} \begin{pmatrix} c_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

which yields m linear constraints over $\mathbb{F}_2$.

2.1 Parameters of BCH codes

The block length of the BCH$[n, D]$ code is n, and its distance is at least D. The latter statement is seen most easily by noting that the BCH code is a subcode of Reed-Solomon codes (i.e., the codewords of the BCH code form a subset of the codewords of the corresponding Reed-Solomon code), so the distance of the BCH code is bounded below by the distance of the Reed-Solomon code.

The dimension of the BCH$[n, D]$ code is a bit more interesting. The dimension of the code is at least $n - (D-1)\log(n+1)$, since in our definition we have $D-1$ constraints on the extension field that each generate $m = \log(n+1)$ constraints in the base field. But this bound on the dimension is not useful: it is (almost) identical to the dimension of Reed-Solomon codes converted to the binary alphabet (for a similar distance and block length), so if this bound were tight we would have no reason for studying BCH codes. This bound, however, can be tightened, as the following more careful analysis shows.

Lemma 5 *For a length $n = 2^m - 1$ and a distance D, the dimension of the* BCH$[n, D]$ *code is at least $n - \lceil \frac{D-1}{2} \rceil \log(n+1)$.*

Proof: In order to establish the tighter bound on the dimension of BCH codes, we want to show that some of the constraints in Definition 4 are redundant. We do so by showing that for any polynomial $c(X) \in \mathbb{F}_2[X]$ and any element $\gamma \in \mathbb{F}_2$, if we have $c(\gamma) = 0$, then we must also have $c(\gamma^2) = 0$. We establish this fact below.

Let $c(X) \in \mathbb{F}_2[X]$ and $\gamma \in \mathbb{F}_2$ be such that $c(\gamma) = 0$. Then we also have $c(\gamma)^2 = 0$, so

$$(c_0 + c_1\gamma + c_2\gamma^2 + \cdots + c_{n-1}\gamma^{n-1})^2 = 0.$$

For any two elements $\alpha, \beta \in \mathbb{F}_{2^m}$, $(\alpha+\beta)^2 = \alpha^2 + \beta^2$, so $c(\gamma) = 0$ also implies

$$\begin{aligned} & c_0^2 + (c_1\gamma)^2 + (c_2\gamma^2)^2 + \cdots + (c_{n-1}\gamma^{n-1})^2 &&= 0 \\ \Leftrightarrow \quad & c_0^2 + c_1^2\gamma^2 + c_2^2(\gamma^2)^2 + \cdots + c_{n-1}^2(\gamma^2)^{n-1} &&= 0. \end{aligned}$$

Since the coefficients $c_0, c_1, \ldots, c_{n-1}$ are in $\mathbb{F}_2$, $c_i^2 = c_i$ for all $i = 0, 1, \ldots, n-1$. Therefore, $c(\gamma) = 0$ implies that

$$c_0 + c_1\gamma^2 + c_2(\gamma^2)^2 + \cdots + c_{n-1}(\gamma^2)^{n-1} = c(\gamma^2) = 0,$$

which is what we wanted to show.

To complete the proof, we now observe that the fact we just proved implies that the constraints $c(\alpha^{2j}) = 0$ for $j = 1, 2, \ldots, \lfloor \frac{D-1}{2} \rfloor$ are all redundant (and implies by $c(\alpha^j) = 0$); we can remove these constraints from the definition without changing the set of codewords in a BCH code. Doing this operation leaves $\lceil \frac{D-1}{2} \rceil m = \lceil \frac{D-1}{2} \rceil \log(n+1)$ constraints. $\square$

Remark 6 *The bound in Lemma 5 is asymptotically tight; the 2 can not be improved to $2 - \epsilon$ for any $\epsilon > 0$.*

The asymptotic tightness of the bound in Lemma 5 follows from the Hamming bound.

2.2 Alternative characterization

Another interpretation of the BCH$[n, D]$ code is that it is equivalent to taking the definition of Reed-Solomon codes, and modifying it to keep only the polynomials where all the evaluations lie in the base field. In fact, this interpretation leads to the following corollary to Theorem 3. The proof follows immediately from Theorem 3 and Definition 4 of BCH codes.

Corollary 7 *BCH codes are subfield subcodes of Reed-Solomon codes. Specifically,*

$$\text{BCH}[n, D] = \text{RS}_{\mathbb{F}, \mathbb{F}^*}[n, n - D + 1] \cap \mathbb{F}_2^n.$$

An important implication of Corollary 7 is that any decoder for Reed-Solomon codes also yields a decoder for BCH codes. Therefore, in later lectures, we will only concentrate on devising efficient algorithms for decoding Reed-Solomon codes; those same algorithms will immediately also give us efficient decoding of BCH codes.

2.3 Analysis and applications

Just like Hamming codes, BCH codes have a very good rate but are only useful when we require a code with small distance (in this case, BCH codes are only useful when $D \leq \frac{n}{\log n}$). In fact, there is a much closer connection between Hamming codes and BCH codes:

Exercise 2 *For $n = 2^m - 1$, show that the* BCH$[n, 3]$ *code is the same (after perhaps some coordinate permutation) as the Hamming code* $[2^m - 1, 2^m - 1 - m, 3]_2$.

As we mentioned above, we are not particularly interested in BCH codes from an algorithmic point of view, since efficient decoding of Reed-Solomon codes also implies efficient decoding of BCH codes. But there are some applications where the improved bound in the dimension of BCH code is crucial.

In particular, one interesting application of BCH codes is in the generation of k-wise independent distributions. A distribution over n-bit strings is k-wise independent if the strings generated by this distribution look completely random if you look only at k positions of the strings. The simplest way to generate a k-wise independent distribution is to generate strings by the uniform distribution. But this method requires a sample space with 2^n points. Using BCH codes, it is possible to generate k-wise independent distributions with a sample space of only $\approx n^{k/2}$ points.

3 Reed-Muller codes

The BCH codes we introduced were a generalization of Hamming codes. We now generalize the dual of Hamming codes – Hadamard codes. The result is another old family of algebraic codes called Reed-Muller codes. We saw in Notes 1 that Hadamard codes were related to first-order Reed-Muller codes; we now obtain the full class of Reed-Muller codes by considering polynomials of larger degree.

Reed-Muller codes were first introduced by Muller in 1954 [4]. Shortly afterwards, Reed provided the first efficient decoding algorithm for these codes [5]. Originally, only binary Reed-Muller codes were considered, but we will describe the codes in the more general case. The non-binary setting is particularly important: in many applications of codes in computational complexity, Reed-Muller codes over non-binary fields have been used to obtain results that we are still unable to achieve with any other family of codes. We saw one such example, of hardness amplification using Reed-Muller codes, in the Introduction to Computational Complexity class last year.

Definition 8 (Reed-Muller codes) *Given a field size q, a number m of variables, and a total degree bound r, the $\mathrm{RM}_q[m,r]$ code is the linear code over $\mathbb{F}_q$ defined by the encoding map*

$$f(X_1,\ldots,X_m) \to \langle f(\alpha)\rangle\,|_{\alpha\in\mathbb{F}_q^m}$$

applies to the domain of all polynomials in $\mathbb{F}_q[X_1,X_2,\ldots,X_m]$ of total degree $\deg(f)\le r$.

Reed-Muller codes form a strict generalization of Reed-Solomon codes: the latter were defined based on univariate polynomials, while we now consider polynomials over many variables.

There is one term in the definition of Reed-Muller codes that we have not yet defined formally: the *total degree* of polynomials. We do so now: the total degree of the monomial $X_1^{k_1}X_2^{k_2}\cdots X_m^{k_m}$ is $k_1+k_2+\cdots+k_m$, and the total degree of a polynomial is the maximum total degree over all its monomials that have a nonzero coefficient.

3.1 Properties of the code

The block length of the $\mathrm{RM}_q[m,r]$ code is q^m, and the dimension of the code is the number of polynomials in $\mathbb{F}_q[X_1,X_2,\ldots,X_m]$ of degree at most r.

When $q=2$, the size of the $\mathrm{RM}_2[m,r]$ can be computed explicitly: there are $\binom{m}{0}+\binom{m}{1}+\cdots+\binom{m}{r}$ ($\approx m^r$) such polynomials.

In general, for any $q\ge 2$ the number of polynomials on m variables of total degree at most r is

$$\left|\left\{(i_1,\ldots,i_m)\mid 0\le i_j\le q-1,\sum_{j=1}^{m} i_j\le r\right\}\right|.$$

When $q>2$ this count does not have a simple expression.

As with Reed-Solomon codes, the interesting parameter of Reed-Muller codes is their distance. To compute the distance parameter, we look for the minimum number of zeros of any non-zero

polynomial. Since $\alpha^q = \alpha$ for $\alpha \in \mathbb{F}_q$, when considering m-variate polynomials over $\mathbb{F}_q$ which will be evaluated at points in $\mathbb{F}_q^m$, we can restrict the degree in each variable X_i to be at most $q-1$. The distance property of Reed-Solomon codes was a consequence of the following fundamental result: a non-zero univariate polynomial of degree at most d over a field has at most d roots. The Schwartz-Zippel Lemma extends the degree mantra to give a bound on the number of roots of multi-variate polynomials.

Theorem 9 (Number of zeroes of multivariate polynomials) *Let $f \in \mathbb{F}_q[X_1, \ldots, X_m] \neq 0$ be a polynomial of total degree r, with the maximum individual degree in the X_i's bounded by $q-1$. Then*

$$\Pr_{\alpha \in \mathbb{F}_q^m}[f(\alpha) \neq 0] \geq \frac{1}{q^a}\left(1 - \frac{b}{q}\right)$$

where $r = a(q-1) + b$, $0 \leq b < q-1$.

The proof of the Schwartz-Zippel Lemma follows from two slightly simpler lemmas. The first lemma provides a good bound on the number of roots of a multi-variate polynomial when its total degree is smaller than the degree of the underlying field.

Lemma 10 (Schwartz [7]) *Let $f \in \mathbb{F}_q[X_1, \ldots, X_m]$ be a non-zero polynomial of total degree at most $\ell < q$. Then*

$$\Pr_{(a_1,\ldots,a_m) \in \mathbb{F}_q^m}[f(a_1, \ldots, a_m) = 0] \leq \frac{\ell}{q}.$$

Proof: The proof of Lemma 10 is by induction on the number of variables in the polynomial. In the base case, when f is a univariate polynomial, the lemma follows directly from the degree mantra.

For the inductive step, consider the decomposition

$$\begin{aligned} f(X_1, \ldots, X_m) = X_m^{d_m}\, g_m(X_1, \ldots, X_{m-1}) + \cdots \\ + X_m\, g_1(X_1, \ldots, X_{m-1}) + g_0(X_1, \ldots, X_{m-1}) \end{aligned}$$

where d_m is the degree of f in X_m. Then g_m is a non-zero polynomial of total degree at most $\ell - d_m$. By the induction hypothesis,

$$\Pr_{(\alpha_1,\ldots,\alpha_{m-1}) \in \mathbb{F}_q^{m-1}}[g_m(\alpha_1, \ldots, \alpha_{m-1}) = 0] \leq \frac{\ell - d_m}{q}. \tag{2}$$

Also, when $g_m(\alpha_1, \ldots, \alpha_{m-1}) \neq 0$, then $f(\alpha_1, \ldots, \alpha_{m-1}, X_m)$ is a non-zero univariate polynomial of degree at most d_m, so we have

$$\Pr_{(\alpha_1,\ldots,\alpha_m) \in \mathbb{F}_q^m}[f(\alpha_1, \ldots, \alpha_m) = 0 \mid g_m(\alpha_1, \ldots, \alpha_{m-1}) \neq 0] \leq \frac{d_m}{q}. \tag{3}$$

Therefore,

$$\begin{aligned} \Pr_{(a_1,\ldots,a_m) \in \mathbb{F}_q^m}[f(a_1, \ldots, a_m) = 0] &\leq \Pr_{(\alpha_1,\ldots,\alpha_{m-1}) \in \mathbb{F}_q^{m-1}}[g_m(\alpha_1, \ldots, \alpha_{m-1}) = 0] \\ &\quad + \Pr_{(\alpha_1,\ldots,\alpha_m) \in \mathbb{F}_q^m}[f(\alpha_1, \ldots, \alpha_m) = 0 \mid g_m(\alpha_1, \ldots, \alpha_{m-1}) \neq 0] \\ &\leq \frac{\ell - d_m}{q} + \frac{d_m}{q} = \frac{\ell}{q}. \end{aligned}$$

□

Remark 11 *A version of Lemma 10 can also be stated for infinite fields (or integral domains). Specifically, the same proof shows that for any field $\mathbb{F}$ and any subset $S \subseteq \mathbb{F}$, the probability that a non-zero polynomial of total degree ℓ is zero is at most $\frac{\ell}{|S|}$ when the values of the variables are chosen independently and uniformly at random from S.*

In many computer science applications, the field size q is very large, and the bound of Lemma 10 is sufficient. As a result, that lemma is often presented as the Schwartz-Zippel Lemma. For our analysis of Reed-Muller codes, however, we also need to bound the probability that a multi-variate polynomial is zero when the degree of the underlying field is small. The following lemma gives us a good bound in this setting.

Lemma 12 (Zippel [8]) *Let $f \in \mathbb{F}_q[X_1, \ldots, X_m]$ be a non-zero polynomial with maximum degree $\deg_{X_i}(f) \leq d_i$ for $i = 1, \ldots, m$. Then*

$$\Pr_{(a_1,\ldots,a_m)\in\mathbb{F}_q^m}[f(a_1, \ldots, a_m) \neq 0] \geq \frac{\prod_{i=1}^{m}(q - d_i)}{q^m}.$$

PROOF: We again proceed with a proof by induction on the number of variables. When f is univariate, the lemma follows since a degree d polynomial has at most d zeroes.

For the inductive step, consider again the decomposition

$$f(X_1, \ldots, X_m) = X_m^{d_m} g_m(X_1, \ldots, X_{m-1}) + \cdots + X_m\, g_1(X_1, \ldots, X_{m-1}) + g_0(X_1, \ldots, X_{m-1}).$$

The decomposition say that we can think of the (multi-variate) polynomial f as a univariate polynomial in $\mathbb{F}_q[X_1, \ldots, X_{m-1}][X_m]$. That is, f can be viewed as a polynomial on the variable X_m with coefficients coming from $K = \mathbb{F}_q(X_1, \ldots, X_{m-1})$, the field of rational functions in variables $X_1, X_2, \ldots, X_{m-1}$. By the degree mantra for univariate polynomials, we get that there are at most d_m values $\beta \in K$ for which $f(X_1, X_2, \ldots, X_{m-1}, \beta) = 0$ (in the field K). Thus there are certainly at least $q - d_m$ values $\alpha \in \mathbb{F}_q$ that can be assigned to X_m such that $f(X_1, \ldots, X_{m-1}, \alpha)$ is a non-zero polynomial (on $m - 1$ variables). Applying the induction hypothesis to this polynomial completes the proof of the lemma. □

To complete the proof of Theorem 9, we can apply Lemma 12 repeatedly to a polynomial, removing one variable at a time, until the total degree ℓ of the polynomial on the remaining variables satisfies $\ell < q$, and then we can apply Lemma 10. We leave the details of the proof to the reader.

It is reasonable to ask if the bound of Theorem 9 could be improved. In general, it can't. Consider the polynomial

$$f(X_1, \ldots, X_{a+1}) = \prod_{i=1}^{a} \prod_{\alpha\in\mathbb{F}_q^*} (X_i - \alpha) \prod_{\beta\in\{\beta_1,\ldots,\beta_b\}\subset\mathbb{F}_q^*} (X_{a+1} - \beta).$$

The polynomial $f(X_1, \ldots, X_{a+1})$ has total degree $r = a(q - 1) + b$ and maximum degree $q - 1$. The value of $f(X_1, \ldots, X_{a+1})$ is non-zero only when $X_1 = \cdots = X_a = 0$ and $X_{a+1} \notin \{\beta_1, \ldots, \beta_b\}$. The first condition is satisfied with probability $\frac{1}{q^a}$ and the second with probability $(1 - \frac{b}{q})$. So the bound of Theorem 9 is tight.

4 Reed-Muller codes

We can now use the Schwartz-Zippel Lemma to establish the distance parameter of binary Reed-Muller codes.

Recall that the $\mathrm{RM}(m, r)$ binary Reed-Muller code is defined by

$$\mathrm{RM}(m, r) = \{\langle f(\alpha)\rangle_{\alpha \in \mathbb{F}_2^m} \mid f \text{ has total degree } \leq r\}.$$

The block length of this code is $n = 2^m$, and the dimension of this code is

$$k = \binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{r},$$

which is can be roughly approximated by $k \approx m^r$.

Applying Theorem 9 (or Lemma 12) to $q = 2$, we can conclude that the distance of $\mathrm{RM}(m, r)$ is at least 2^{m-r}. We will reprove with a more specialized argument and also show that the distance is exactly 2^{m-r}.

4.1 Decoding Reed-Muller codes

The Reed-Muller codes were first introduced by Muller in 1954 [4]. Muller showed that the family of codes he introduced had good distance parameters, but he did not study the problem of decoding these codes efficiently.

The naïve method of decoding the $\mathrm{RM}(m, r)$ code is to enumerate all the codewords, compute their distance to the received word and to output the one with the minimum distance. This algorithm runs in time $2^k \approx 2^{m^r} = 2^{(\log n)^r}$. The running time of the naïve decoding algorithm is therefore quasi-polynomial (but not polynomial!) in the block length n.

Reed introduced the first efficient algorithm for decoding Reed-Muller codes [5] shortly after the codes were introduced by Muller. Reed's algorithm also corrects up to half the minimum distance (i.e., up to $2^{m-r-1} - 1$ errors) and further runs in time polynomial in the block length n.

We will not cover Reed's decoding algorithm for Reed-Muller codes in this class. At a very high level, the idea of the algorithm is to apply a majority logic decoding scheme. The algorithm was covered in previous iterations of this class; interested readers are encouraged to consult those notes for more details on the algorithm.

4.2 Distance of Reed-Muller codes

Let us now give a self-contained argument proving that the distance of the $\mathrm{RM}(m, r)$ code is 2^{m-r}.

We begin by showing that the distance of binary Reed-Muller codes is at most 2^{m-r}. Since Reed-Muller codes are linear codes, we can do so by exhibiting a non-zero codeword of $\mathrm{RM}(m, r)$ with weight 2^{m-r}. Consider the polynomial

$$f(X_1, \ldots, X_m) = X_1 X_2 \cdots X_r.$$

The polynomial $f \in \mathbb{F}_2[X_1, \ldots, X_m]$ is a non-zero polynomial of degree r, and clearly $f(\alpha_1, \ldots, \alpha_m) \neq 0$ only when $\alpha_1 = \alpha_2 = \cdots = \alpha_r = 1$. There are 2^{m-r} choices of $\alpha \in \mathbb{F}_2^m$ that satisfy this condition, so $\mathrm{wt}(\langle f(\alpha) \rangle_{\alpha \in \mathbb{F}_2^m}) = 2^{m-r}$.

Let us now show that the distance of binary Reed-Muller codes is at least 2^{m-r} by showing that the weight of any non-zero codewords in $\mathrm{RM}(m, r)$ is at least 2^{m-r}. Consider any non-zero polynomial $f(X_1, \ldots, X_m)$ of total degree at most r. We can write f as

$$f(X_1, \ldots, X_m) = X_1 X_2 \cdots X_s + g(X_1, \ldots, X_m)$$

where $X_1 X_2 \cdots X_s$ is a maximum degree term in f and $s \leq r$. Consider any assignment of values to the variables $X_{s+1}, \ldots, X_m$. After this assignment, the resulting polynomial on $X_1, \ldots, X_s$ is a non-zero polynomial, since the term $X_1 X_2 \cdots X_s$ cannot be cancelled. Therefore, for each of the 2^{m-s} possible assignment of values to the variables $X_{s+1}, \ldots, X_m$, the resulting polynomial is a non-zero polynomial.

When you have a non-zero polynomial, then there is always at least one assignment of values to its variables such that the polynomial does not evaluate to 0. Therefore, for each assignment $\alpha_{s+1}, \ldots, \alpha_m$ to the variables $X_{s+1}, \ldots, X_m$, there exists at least one assignment of values $\alpha_1, \ldots, \alpha_s$ to $X_1, \ldots, X_s$ such that $f(\alpha_1, \ldots, \alpha_m) \neq 0$. This implies that $\mathrm{wt}(\langle f(\alpha) \rangle_{\alpha \in \mathbb{F}_2^m}) = 2^{m-s} \geq 2^{m-r}$.

In summary, when the maximum degree r is constant, binary Reed-Muller codes have good distance, but a poor rate ($\approx m^r 2^{-m} \to 0$ for large m). Increasing the parameter r increases the rate of the code but also decreases the distance of the code at a faster pace. So there is no setting of r that yields a code with constant rate and constant distance.

In the following section, we introduce a family of binary codes that can be constructed efficiently and has both a good rate and a good distance simultaneously.

5 Concatenated codes

Concatenated codes were introduced by Forney in his doctoral thesis in 1966 [2]. In fact, Forney proved many wonderful properties about these codes; in this lecture we only give a brief overview of the definitions and key properties of concatenated codes.

The starting point in our search for binary codes with good rate and good distance is the idea that we have already seen codes with good rate and good distance when we have large alphabets: the distance of Reed-Solomon codes meets the Singleton bound, so they in fact are optimal codes. So let's start with Reed-Solomon codes and see if we can use them to construct a family of good binary codes.

We already saw in the last lecture a simple transformation for converting Reed-Solomon codes to binary codes. In this transformation, we started with a polynomial f of degree $k-1$ and evaluated it over $\alpha_1, \ldots, \alpha_m$ to obtain the values $f(\alpha_1), \ldots, f(\alpha_m) \in \mathbb{F}_{2^m}^n$. We then encoded each of the values $f(\alpha_i)$ in the binary alphabet with m bits.

The binary code obtained with the simple transformation has block length $N = nm$ and distance $D \geq d = n - k + 1$. This distance is not very good, since the lower bound on the relative distance $\frac{D}{N} \geq \frac{n-k+1}{nm}$ is quite weak. Still, the lower bound $D \geq d$ follows from a very simple analysis; one

may hope that a better bound – ideally of the form $D \geq \Omega(dm)$ – might be obtained with a more sophisticated analysis or by applying some neat trick (like, say, by encoding the bits in some clever basis). Unfortunately, that hope is not realizable: there is a nearly tight upper bound showing that with the simple transformation, the distance of the resulting binary code is at most $D \leq 2d$.

So if we hope to obtain a binary code with good distance from the Reed-Solomon code, we need to introduce a new idea to the transformation. One promising idea is to look closely at the step where we took the values from the field $\mathbb{F}_{2^m}$ and encoded them with m bits in the binary alphabet: instead of using the minimum number of bits to encode those elements in the binary alphabet, we could use more bits – say $2m$ bits – and use an encoding that adds more distance to the final code. That is indeed the idea used to obtain concatenated codes.

5.1 Binary concatenated codes

The concatenated code $C = C_{out} \diamond C_{in}$ is defined by two codes. The *outer* code $C_{out} \subset \Sigma_1^{n_1}$ converts the input message to a codeword over a large alphabet Σ_1, and the *inner* code $C_{in} \subset \Sigma_2^{n_2}$ is a much smaller code that converts symbols from Σ_1 to codewords over Σ_2. When $\Sigma_2 = \{0, 1\}$, the code C is a binary concatenated code.

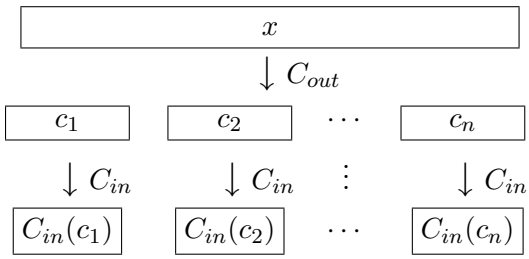


Figure 1: Binary concatenated codes.

A key observation in the definition of concatenated codes is that the inner code C_{in} is a small code, in the sense that it only needs one codeword for each symbol in Σ_1. The size of the alphabet Σ_1 is (typically) much smaller than the total number of codewords encoded by C, and will let us do a brute-force search for good inner codes in our construction of good concatenated codes. But first, let us examine the rate and distance parameters of general concatenated codes.

5.2 Facts of concatenated codes

The rate of the concatenated code $C = C_{out} \diamond C_{in}$ is

$$R(C) = \frac{\log |C_{out}|}{n_1 n_2 \log |\Sigma_2|} = \frac{\log |C_{out}|}{n_1 \log |\Sigma_1|} \cdot \frac{\log |\Sigma_1|}{n_2 \log |\Sigma_2|} = R(C_{out}) \cdot R(C_{in}),$$

where the last equality uses the fact that $|C_{in}| = |\Sigma_1|$.

The simple transformation of Reed-Solomon codes to binary codes used an inner code C_{in} with rate 1 (which did not add any redundancy). The rate equation $R(C) = R(C_{out}) \cdot R(C_{in})$ says that we can replace the trivial inner code with any other code and incur a rate cost proportional to the rate of C_{in}.

Let's now look at the distance of concatenated code. We do not get an exact formula for the distance of these codes, but a simple argument does give us a lower bound that will be sufficient to construct concatenated codes with good distance:

Proposition 13 *The distance of the concatenated code $C = C_{out} \diamond C_{in}$ satisfies*

$$\Delta(C) \geq \Delta(C_{out}) \cdot \Delta(C_{in}).$$

PROOF: Let x and y be two distinct messages. The distance property of the outer code guarantees that the encodings $C_{out}(x)$ and $C_{out}(y)$ will differ in at least $\Delta(C_{out})$ symbols. For each of the symbols where they differ, then the inner code will encode the symbols into codewords that differ in at least $\Delta(C_{in})$ places. □

The lower bound of Proposition 13 is not tight, and in general the distance of concatenated codes can be much larger. This may seem counter-intuitive at first: at the outer level, we can certainly have two codewords that differ in only $\Delta(C_{out})$ places, and at the inner level we can also have two different symbols in Σ_1 whose encodings under C_{in} differ in only $\Delta(C_{in})$ places. But the two events are not necessarily independent – it could be that when there are two codewords at the outer level that differ at only $\Delta(C_{out})$ symbols, then they must differ in a pattern that the inner code can take advantage of so that for those cases, the inner code does much better than its worst case.

In fact, a probabilistic argument shows that when the outer code is a Reed-Solomon code and the inner codes are "random projections" obtained by mapping the symbols of Σ_1 to codewords in Σ_2 with independently chosen random bases, then the resulting concatenated code reaches the Gilbert-Varshamov bound with high probability. (And thus has distance much larger than the lower bound suggested by Proposition 13.) This construction is randomized; it is an interesting problem to give a family of *explicit* codes for which the inequality of Proposition 13 is far from tight. (There are some codes called *multilevel concatenated codes* where the Zyablov bound can be improved, but this still falls well short of the GV bound.)

5.3 Constructing good concatenated codes

In this section, we construct a family of binary concatenated codes with good rate and good distance. Fix $0 < R < 1$ to be our target rate. We will build a code with rate R and distance as large as possible.

For our construction, take $C_{out} = [n, k, n-k+1]_{2^m}$ to be the Reed-Solomon code with block length $n = 2^m$. The rate of this outer code is $R_{out} = \frac{k}{n}$ and the relative distance of this code is $\delta_{out} = \frac{n-k+1}{n} \geq 1 - R_{out}$. Take C_{in} to be a binary linear code with parameters $[\frac{m}{r}, m, d]_2$, so that the rate of the inner code is $R(C_{in}) = r$. The rate of the concatenated code $C = C_{out} \diamond C_{in}$ is $R = R_{out} \cdot r$, so $R_{out} = \frac{R}{r}$.

We now have a partial construction. The outer code is the Reed-Solomon code, which we know is optimal so we're done with this part of the construction. The inner code, however, is not yet defined: we have only specified that we want C_{in} to be a linear code with rate r. For our concatenated code C to have good distance, we want the distance of C_{in} to be as large as possible.

The asymptotic Gilbert-Varshamov bound guarantees that there exists a linear code C_{in} with rate $r \geq 1 - h(\delta_{in})$. Rearranging the terms, this means that there is a code with rate r and distance $\delta_{in} \geq h^{-1}(1-r)$. So if we find an inner code that matches this distance bound, we obtain a concatenated code C with distance

$$\delta(C) \geq \delta_{out} \cdot \delta_{in} \geq (1 - R_{out}) \cdot h^{-1}(1-r) = \left(1 - \tfrac{R}{r}\right) \cdot h^{-1}(1-r).$$

The question remains: how can we find an inner code C_{in} with minimum distance $\delta_{in} \geq h^{-1}(1-r)$? Since C_{in} is a small code, so we can do a brute force search over the linear codes to find one with large distance.

We have to be a little careful in the algorithm that we use to search for C_{in}. A naïve searching algorithm simply enumerates all the possible generator matrices for C_{in} and checks the distance of each corresponding code. But there are $2^{m \times m/r} = n^{\log(n)/r}$ possible generator matrices G, so this search does not run in time polynomial in the block length of the code.

There is a more efficient algorithm for finding an inner code C_{in} with minimum distance d. The algorithm uses the greedy method to build a parity check matrix H such that every set of $d-1$ columns in H is linearly independent: Enumerate all the possible columns. If the current column is not contained by the linear span of any $d-2$ columns already in H, add it to H.

The greedy algorithm examines $2^{m/r-m} = n^{1/r-1}$ columns, and as long as $2^{m/r-m} > \sum_{i=0}^{d-2} \binom{n-1}{i}$, the process is also guaranteed to find a parity-check matrix H of distance d. So this method can be used to find a linear code that meets the Gilbert-Varshamov bound in time polynomial in the block length.

This completes our construction of a binary concatenated code with good rate and good distance. In the next section, we examine the best rate-distance trade-off obtained by optimizing the parameters of the concatenated code. But first, we mention one more useful property of the code we have constructed: it is a linear code.

Exercise 3 *Prove that the concatenated code C is linear over $\mathbb{F}_2$.*

5.4 Zyablov radius

In our construction of good concatenated codes, we are free to set the rate r of the inner code. Optimizing the value of r over all the choices that guarantee an overall rate of R for the concatenated code yields the following result.

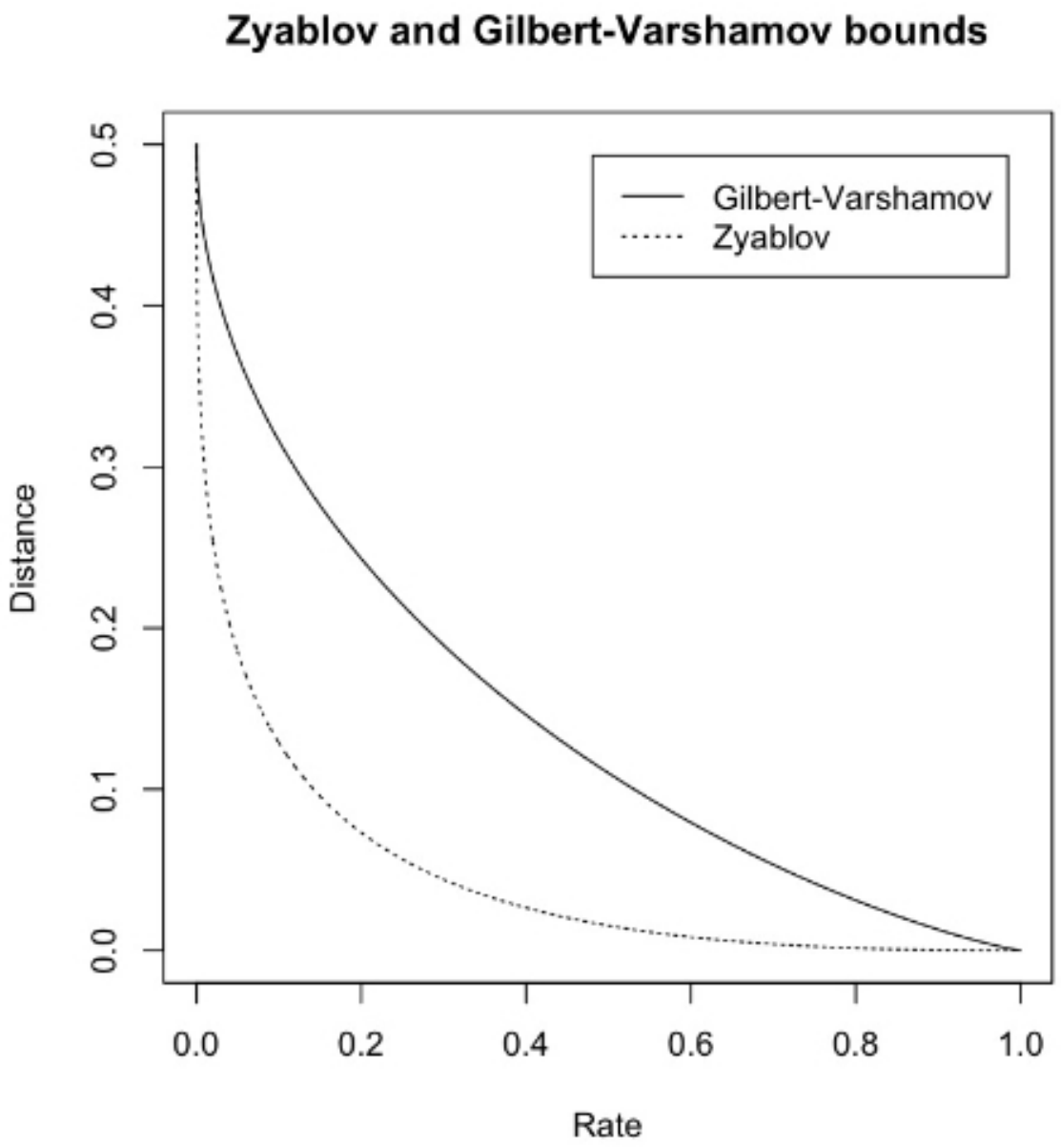
Theorem 14 *Let $R \in (0,1)$. Then it is possible to efficiently construct a code of rate R and distance*

$$\delta_{Zyablov}(R) = \max_{R \leq r \leq 1} (1 - \tfrac{R}{r}) h^{-1}(1-r).$$

The function $\delta_{Zyablov}$ is called the Zyablov trade-off curve, or sometimes the Zyablov bound, and is named after Zyablov, who first observed it in 1971 [9]. For any value of $R \in (0,1)$, the value of $\delta_{Zyablov}(R)$ is bounded away from 0, so we get the following corollary.

Corollary 15 *Asymptotically good codes of any desired rate $R \in (0,1)$ can be constructed in polynomial time.*

So how good is the resulting bound? Quite a bit weaker than the Gilbert-Varshamov bound, as the figure shows.



Another aspect of our construction of concatenated codes that is somewhat unsatisfactory is that whilt it it is constructed in polynomial time, it involves brute-force search for a code of logarithmic

block length. It would be nice to have an explicit formula or description of how the code looks like. From a complexity view point, we might want a linear code the entries of whose generator matrix we can compute in polylogarithmic time.

In the next lecture, we will see an asymptotically code that is constructedexplicitly without any brute-force search of smaller codes, and which further achievs the Zyablov trade-off between rate and relative distance for rates more than 0.31.

References

[1] Ray C. Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960. 4

[2] G. David Forney. *Concatenated codes*. MIT Press, 1966. 11

[3] Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2:147–156, 1959. 4

[4] David E. Muller. Application of boolean algebra to switching circuit design and to error detection. *IEEE Trans. on Computers*, 3:6–12, 1954. 7, 10

[5] Irving S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IEEE Trans. on Information Theory*, 4:38–49, 1954. 7, 10

[6] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *J. Soc. Indust. Appl. Math.*, 8(2):300–304, 1960. 3

[7] Jack T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980. 8

[8] Richard Zippel. Probabilistic algorithms for sparse polynomials. *Symbolic and Algebraic Computation*, Springer LNCS 72:216–226, 1979. 9

[9] Victor V. Zyablov. An estimate of the complexity of constructing binary linear cascade codes. *Probl. Peredachi Inf.*, 7(1):5–13, 1971. 14