CSE 533: Error-Correcting Codes **(Autumn 2006)**

Lecture 10: Justesen Codes and Reed-Solomon Decoding

1 November 2006

Lecturer: Venkatesan Guruswami *Scribe: Paul Pham*

In the last lecture, we introduced the first binary code with asymptotically positive rate and distance as well as an explicit polynomial-time construction. We give that construction in this lecture. In the process, we introduce an efficient decoder for Reed-Solomon codes. But first, we recall the Zyablov, Gilbert-Varshamov, and MRRW (record) bounds below:

$$R_{Zyablov}(\delta - \frac{1}{2} - \varepsilon) = \Omega(\varepsilon^3) \quad (1)$$

$$\delta(R) = \max_{R<r<1}(1 - \frac{R}{r})H^{-1}(1-r) \quad (2)$$

$$R_{GV}(\frac{1}{2} - \varepsilon) = \Omega(\varepsilon^2) \quad (3)$$

$$R_{MRRW}(\frac{1}{2} - \varepsilon) = 2\varepsilon^2 \log \frac{1}{\varepsilon} \quad (4)$$

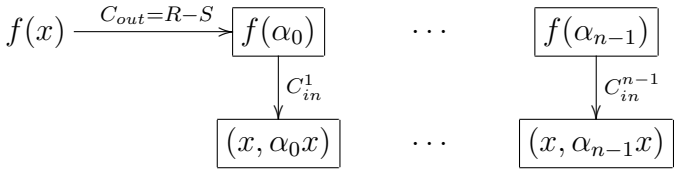
1 Justesen Code Construction

Up until now, we have seen bounds that tell us good codes should be possible but we didn't know how to construct such codes explicitly. The binary Reed-Solomon code gave us good rate but poor distance. Intuitively, we should be able to concatenate it with an inner code that would "amplify" its distance. Enumerating and searching codes takes $O(2^n)$ time, so we can't just find one good inner code and use it for all outer blocks. A key insight of the Justesen code is the use of different inner codes from an easy-to-find ensemble, most of which have good distance.

The Justesen code concatenates Reed-Solomon on the outside with an ensemble of codes over a field of 2^m elements on the inside. We denote n as the number of non-zero field elements, $2^m - 1$, and therefore each field element can be encoded into binary with $m = O(\log n)$ bits. The associated Reed-Solomon polynomial is mk-bits long:

$$f(x) = z_0 + z_1 x + \ldots + z_{k-1}x^{k-1} \quad (5)$$

We then evaluate $f(x)$, as well as a correlated polynomial $xf(x)$ at all non-zero field elements ($\{\alpha_i\}$). This gives us $2n$ field elements, $(f(\alpha_0), \alpha_0 f(\alpha_0), \ldots, f(\alpha_{n-1}), \alpha_{n-1}f(\alpha_{n-1})$, which can be viewed as a $2nm$-bit string.



In the above figure, the inner codes functions are

$$C_{in}^i(x) = (x, \alpha_i x \tag{6}$$

Note that these functions are defined over $\mathbb{F}_{2^m} \rightarrow \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$. However, we can easily convert this to a binary function by embedding each field element as an m-bit vector while preserving linearity, as before: $\sigma : \mathbb{F}_{2^m} \rightarrow \mathbb{F}_2^m$ where $\sigma(x+y) = \sigma(x) + \sigma(y)$. Now we can define our C_{in}^i functions over $\mathbb{F}_2^m \rightarrow \mathbb{F}_2^{2m}$. And of course, we should use binary Reed-Solomon on the outside.

Now we will argue that the rate and distance for this concatenated binary code are asymptotically good.

2 Justesen Code Rate and Distance

First, the rate: $R = \frac{k}{2n}$, or in other words, half the rate of Reed-Solomon. This makes sense because we are transmitting the same message with twice as many symbols. Alternatively, note that each $C_i n^i$ defined above is a rate $\frac{1}{2}$ linear code.

If all but an ε-fraction of inner codes have distance d, then the concatenated code has distance $\geq (D - \varepsilon n)d$. Now we need to show that greater than a constant fraction of the inner ensemble has good distance.

Fact 2.1 (Most inner codes $C_{in,i}$ have distance $\delta > H^{-1}(\frac{1}{2}) - \varepsilon$.). *Proof: Let y be a non-zero element of $\mathbb{F}_2^{2m}$ which we can write as a tuple of elements (y_1, y_2) where $y_1, y_2 \in \mathbb{F}_2^m$.*

If y is in some particular code C_{in}^i then $y_2 = \alpha_{i-1} y_1$ by construction in the previous section. Then this code is fixed by the ratio $y2/y1$. Note that neither y_1 nor n_2 can both be zero, otherwise the whole codeword will be zero contrary to our assumption. If one is zero and the other is not, then it is not in any code. Otherwise, if y is not in any code, then we can mark it as an erasure and throw it out.

From the above argument, each non-zero element $y \in \mathbb{F}_2^{2m}$ is in at most one of the codes C_{in}^i.[1] *Therefore, we can bound the number of "bad" codes (εn codes with distance $\leq d$) is bounded*

[1] In fact, our proof still goes through as long as each element is in at most some constant number of codes.

above by the number of bad codewords (which are $2m$ bits long).

$$\begin{aligned}
\textit{\# codes with dist } < d &\leq Vol(0, d \cdot 2m) && (7)\\
&\leq 2^{H(d)2m} \textit{ by Lemma 3.3 in Lecture 3} && (8)\\
&= 2^{(\frac{1}{2}-\varepsilon)2m} && (9)\\
&= \frac{2^m}{2^{\varepsilon \cdot 2m}} && (10)\\
&< \varepsilon(2^m - 1) && (11)\\
&= \varepsilon n && (12)
\end{aligned}$$

This decreases □

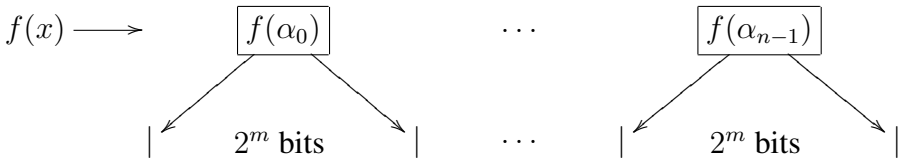
Therefore, the Justesen code has the distance below:

$$\begin{aligned}
\text{distance} &\geq ((1-R)n - \varepsilon n)d \cdot 2m && (13)\\
&= (1 - R - \varepsilon)nH^{-1}(\frac{1}{2} - \varepsilon)2m && (14)\\
\delta &\geq (1 - R - \varepsilon)H^{-1}(\frac{1}{2} - \varepsilon) && (15)
\end{aligned}$$

The last equation above is equivalent to the Zyablov bound given at the beginning of these notes. In conclusion, the code can be used for any rate $< \frac{1}{2}$ and still be on the Zyablov bound.

3 Digression: Reed-Solomon ⋄ Hadamard

Leaving behind Justesen codes for a moment, we will examine a construction with some interesting properties. We concatenate Reed-Solomon as the outer code with Hadamard as the inner code, where $\mathrm{Had}[a](x) = (a \cdot x), \forall x \in \mathbb{F}_{2m}$. We call the outer rate ε and take the outer distance to be $\delta_{out} = 1 - \varepsilon$ and the inner distance to be $\delta_{in} = \frac{1}{2}$.



The concatenated distance is $\delta = \frac{1}{2}(1 - \varepsilon)$, which implies that all non-zero codewords have relative weight within the range $[\frac{1}{2} - \varepsilon, \frac{1}{2}]$, otherwise known as an ε-biased space.

Alon, Goldreich, Håstad, and Peralta [9] showed that k independent variables (a message) could be used to simulate a "small" distribution over $n \gg k$ variables (codewords) within $(k/\varepsilon)^2$ vectors, a polynomial in k. In error-correcting terms, the generator matrix for this code is a $(k/\varepsilon)^2 \times k$ matrix but is equivalent to a code of size 2^{2m} with respect to any linearity test (dot product) for a large fraction of the inputs.

If we characterize the outer Reed-Solomon code as $[n, k, n-k]_n$ and the inner Hadamard code as $[n, \log n, \frac{n}{2}]_2$, we get a concatenated code of $[n^2, k \log n, \frac{n}{2}(n-k)]$. The concatenated relative

distance can be expressed as $\delta = \frac{1}{2} - \frac{k}{n}$. An interesting fact is that we can choose the outer rate k/n to get a constant total distance but poor rate, starting with an outer code that gave us constant rate at the expense of poor distance. However, the inner code has both poor rate and distance. Intuitively, it seems that if we concatenate an inner code with good distance and the Reed-Solomon code with good rate, we should be able to get an asymptotically good code.[2]

Now back to our regularly-scheduled programming.

4 General Linear Erasure Decoding

Recall that an erasure is an error that we know with certainty, whereas in general we may not be able to detect all errors. From the first lecture, we know that in a code with distance d, any received message with $(d-1)$ erasures or fewer is consistent with at most one codeword. Therefore, we can "fill in" indeterminate symbols up to $(d-1)$ erasures.

How can we do this efficiently? It's always easy for a linear code if we are given the generator matrix G. If the message is $x \in \mathbb{F}_q^k$, compute and send $Gx \in \mathbb{F}_q^n$. Suppose that some symbols may be corrupted in transit, and the receiver gets $y \in \mathbb{F}_q^n$. Call the subset of *uncorrupted* positions $S \subseteq \{1, \ldots, n\}$, which we know by assumption. Then we merely have to solve a linear system, where subscript y_S means y restricted to the positions in S and likewise for Gx_s.

$$Gx_S = y_S \tag{16}$$

Without loss of generality, we can reorder the rows of G and y so that the positions in S are at the top and the corrupted positions are at the bottom. Then we simply solve the linear system restricted to the top $|S|$ rows of G and y.

$$\left[\begin{array}{c} G_S \\ \hline G_{\overline{S}} \end{array}\right] \left[\begin{array}{c} \\ x \\ \\ \end{array}\right] = \left[\begin{array}{c} y_S \\ \hline y_{\overline{S}} \end{array}\right] \tag{17}$$

If there are few enough erasures ($|S| \geq k$), we will get a unique solution. If there are more erasures, we can only output a linear space of solutions, which is a form of list-decoding.

5 Reed-Solomon Decoding

The last piece in our code is the decoding algorithm, which itself depends on being able to decode Reed-Solomon efficiently. First, we'll summarize the work leading up to the R-S decoder we know and love today. Then we'll describe R-S erasures-only decoding as a subroutine to the final R-S errors-and-erasures decoder in the final section.

[2]These insightful remarks came from Madhu Sudan's MIT course notes for 6.897 in Fall 2001.

5.1 History

The history Reed-Solomon decoding can be traced back to the first efficient algorithm for binary BCH codes by Peterson in 1960 [1], since R-S codes are a special case of BCH codes. This algorithm was able to correct errors up to half the distance ($e \geq \lfloor \frac{d-1}{2} \rfloor$) in $O(n^3)$ time using error locator polynomials. Interestingly, the original Reed-Solomon paper appeared earlier that year, when digital technology was not advanced enough to implement the decoding procedure.

For Reed-Solomon codes, the first practical decoder is attributed to Berlekamp in 1968 [2] which solved the key equation iteratively. It was reformulated in a shift register setting by Massey in 1969 [3] which was easy to implement. This procedure, often called the Berlekamp-Massey algorithm, is used in all commercial CD and DVD players, transmissions from the Voyager satellite, and many other applications in mass storage and communication. The corresponding patent by Berlekamp and Welch in 1986 [7] was a key asset of Berlekamp's company Cyclotomics. A succinct description can be found in the appendix of [8] by Gemmell and Sudan in 1992. This algorithm runs in $O(n^2)$ time and will be presented in the next section.

An alternate $O(n^2)$ algorithm using Euclid's GCD method to solve the key equation was discovered by Sugiyama, Kasahara, Hirasawa, and Namekawa in 1975 [4]. This was generalized to a single algorithm for both errors and erasures in [5] a year later, and in the same journal issue was published Justesen's algorithm [6] with the best known upper bound to date, $O(n \log^2 n)$.

5.2 Reed-Solomon Erasure Decoding

Similar to the process for general linear codes above, erasure decoding for Reed-Solomon codes is straightforward. Each code has an associated polynomial $f(x)$ for $x \in \mathbb{F}_q^k$. Suppose the sender transmits $(f(\alpha_0), \ldots, f(\alpha_{n-1}))$, and again we know the uncorrupted positions $f(\alpha_i)$ for $i \in S$ with $|S| \geq n - (d-1) = k$.

Since $\deg(f(x)) \leq k-1$, we can recover f using k or more distinct points using polynomial interpolation according to the unisolvence theorem. This can be done using FFT in $O(n \log n)$ time or with Lagrange interpolation in $O(n^2)$ time. The latter method computes n basis polynomials p_j for $0 \leq j < k$.

$$p_j(x) = \prod_{i=0, i \neq j}^{k} \frac{x - \alpha_i}{\alpha_j - \alpha_i} \tag{18}$$

The interpolated polynomial is then:

$$f(x) = \sum_{i=0}^{k} f(\alpha_i) p_i(x) \tag{19}$$

Now we must deal with the case of some of the received symbols $f(\alpha_i)$ being in error.

5.3 Reed-Solomon Error Decoding Problem

As above, our sender's encoded message was $(f(\alpha_0), \ldots, f(\alpha_{n-1})) \in \mathbb{F}_q^n$ and the receiver gets $y = (y_0, y_1, \ldots, y_{n-1}) \in \mathbb{F}_q^n$. If a symbol in position i was received correctly (without error), then $y_i = f(\alpha_i)$. Let's say that a fraction e of the received symbols have been corrupted, that is $y_i \neq f(\alpha_i)$. We assume $2e < d = n - k + 1$, otherwise we have no chance for uniquely correcting the error. If we can figure out which positions are in error, we can throw them out and just do erasure decoding as in the previous section.

More formally, our problem can be stated as follows. Given n pairs of $(\alpha_i, y_i) \in \mathbb{F}_q, i \in \{0, 1, \ldots, n-1\}$, determine if there is a polynomial $f(x) \in \mathbb{F}_q[x]$ of degree $\leq k-1$ such that $f(\alpha_i) \neq y_i$ for at most a fraction e of the n pairs. If there is, find it. In our case, the points α_i must be distinct for us to apply polynomial interpolation, but the problem still makes sense without this condition.

5.4 Reed-Solomon Error Decoding Algorithm

Recall that if there is no error in position i, then $y_i = f(\alpha_i)$. We define an error locator polynomial, which we don't know *a priori*.

$$E(x) \triangleq \prod_{f(\alpha_i) \neq y_i} x - \alpha_i \tag{20}$$

It is easy to verify that for $0 \leq i < n$, $E(\alpha_i)y_i = E(\alpha_i)f(\alpha_i)$. Define the key equation:

$$N(x) \triangleq E(x)f(x) \tag{21}$$

Then for $0 \leq i < n$, we have the linear system of equations $N(\alpha_i) - E(\alpha_i)y_i = 0$. Note that $E(x)$ has degree $\leq e$ and $f(\alpha_i)$ has degree $\leq k-1$. Call one particular solution of this system x and define the two polynomials below:

$$E_1(x) = a_0 + a_1 x + \ldots + a_e x^e \tag{22}$$

$$N_1(x) = b_0 + b_1 x + \ldots + b_{e+k-1} x^{e+k-1} \tag{23}$$

This is a homogenous linear system in the unknowns a_i for $0 \leq i \leq e$ and b_j $0 \leq j < e + k$. There may be other solutions x, but the ratio between the two polynomials will be the same.

1. If $E_1(x)$ does not divide $N_1(x)$, output "too many errors." Otherwise, define $f(x) = N_1(x)/E_1(x)$.
2. If $\Delta(RS(f(x)), y) > e$, output "too many errors" otherwise return $f(x)$.

We know that a non-zero solution x must exist by a root/degree counting argument. Define another polynomial:

$$R(x) = N_1(x) - f(x)E_1(x) = 0 \tag{24}$$

The degree of $R(x)$ is $\leq e + k - 1$. Assume that $\leq e$ errors happen. For the positions with no errors, $y_i = f(\alpha_i)$ and we have:

$$\begin{aligned} R(\alpha_i) &= N_1(\alpha_i) - f(\alpha_i)E_1(\alpha_i) && (25) \\ &= N_1(\alpha_i) - y_i E_1(\alpha_i) && (26) \end{aligned}$$

$R(\alpha_i) = 0$ for at least $n - e$ values of i. If the number of roots $n - e$ is greater than the degree, $R(x)$ is the zero polynomial, which implies a non-zero solution because all the α_i's are non-zero. Rearranging the inequality, we get:

$$2e < n - k + 1 = D \tag{27}$$

where D is the distance of Reed-Solomon. This shows that we can correct errors and erasures up to half the distance. □

References

[1] W.W. Peterson. "Encoding and error-correction procedures for the Bose-Chaudhuri codes." *IRE Transactions on Information Theory*, Vol. 6, pp. 459-470, 1960.

[2] E.R. Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, New York, 1968.

[3] J.L. Massey. "Shift register synthesis and BCH decoding." *IEEE Transactions on Information Theory*, Vol. IT-17, pp. 122-127, 1969.

[4] Y. Sugiyama, M. Kasahara, S. Hirasawa, T. Namekawa. "A method for solving key equation for decoding Goppa codes." *Information and Control* Vol. 27, pp. 87-89, 1975.

[5] Y. Sugiyama, M. Kasahara, S. Hirasawa, T. Namekawa. "An erasures-and-errors decoding algorithm for Goppa codes." *IEEE Transactions on Information Theory*, Vol. 22, No. 2, March 1976.

[6] J. Justesen. "On the complexity of decoding Reed-Solomon codes." *IEEE Transactions on Information Theory*, Vol. 22, No. 2, March 1976.

[7] E.R. Berlekamp and L.R. Welch. "Error Correction of Algebraic Block Codes." U.S. Patent No. 4,633,470. 30 December 1986.

[8] P. Gemmell and M. Sudan. "Highly-Resilient Correctors for Polynomials." *Information Processing Letters*, 1992.
`http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/CSD-92-661.pdf`

[9] N. Alon, O. Goldreich, J. Hstad, and R. Peralta. "Simple constructions of almost k-wise independent random variables." *Random Structures and Algorithms*, Vol. 3, pp. 289-304, 1992.