Linear Error-Correcting Codes

Lecturer: Peter Shor

1 Linear Error Correcting Codes

We wish to transmit a message to a destination through a channel reliably. Errors may occur during transmission and we hope to detect and correct these transmitted errors. In order to do so, we must introduce some redundancy in our transmitted message.

We saw that Shannon's Noisy Coding Theorem tells us that if the redundancy is above a certain level (constrained by the noise of the channel and the type of channel), then we can drive the probability of error (i.e. not recovering the original message) to zero with increasing blocklength. (The blocklength refers to the length n of the encoded message or codeword being sent; in actual transmission, longer messages are first cut into blocks of length k before being encoded and transmitted.) The proof of Shannon's noisy coding theorem was based on a random code which is highly impractical. For easy encoding and decoding, we need structure in our encoding/decoding functions. We will study families of *linear block codes*, and their structure offers several advantages. Linearity will allow an easier analysis of the error correcting ability of the code. Furthermore, the use of matrices to encode/decode messages means that the code can be concisely described and that the encoding and decoding will be much easier than using a random codebook. The simplest linear code we will consider is the Hamming code, and afterwards we will consider Reed-Solomon codes.

In Shannon's noisy coding theorem, we had considered only messages, codewords and received messages that were binary strings, although we could have extended the analysis to larger alphabets. Here, to use linearity, we consider more generally strings (or vectors) whose elements come from a finite field $\mathbb{F}$, i.e. a field with a finite number of elements (see notes on Modular Arithmetic and Elementary Algebra). The binary setting corresponds to $\mathbb{F} = \mathbb{Z}_2$ (the corresponding codes are called binary codes), and this is what we will use for the Hamming (binary) code. We have seen finite fields $\mathbb{Z}_p$ with a prime number p of elements, but in fact there exist finite fields $\mathbb{F}$ with a number of elements equal to any prime power, $|\mathbb{F}| = p^k$ for some integer k. For Reed-Solomon codes, we will need a field $\mathbb{F}$ with a large number of elements.

In Shannon's noisy coding theorem, the errors/noise introduced were random; for example in the binary symmetric channel with parameter p, each bit of the codeword is flipped with probability p and this results in a number of errors tighly concentrated around np (as n grows). Here, however, in the analysis of linear codes, we assume that the errors are *adversarial*. We allow the channel to modify up to say t symbols. In this context, we say that a code can correct up to t errors or is an *error-correcting code* for t errors if no matter which (up to) t elements are corrupted, the transmitted sequence can still be decoded correctly. Allowing adversarial errors is a stronger requirement.

In a *linear code* over a field $\mathbb{F}$, messages are elements of $\mathbb{F}^k$, codewords are elements of $\mathbb{F}^n$ (where $n \geq k$) and the encoding function $c : \mathbb{F}^k \to \mathbb{F}^n$ is linear. In other words, for any two messages

$m_1, m_2 \in \mathbb{F}^k$ and for any $\alpha_1, \alpha_2 \in \mathbb{F}$ we have

$$c(\alpha_1 m_1 + \alpha_2 m_2) = \alpha_1 c(m_1) + \alpha_2 c(m_2).$$

In particular, this means that the all-0 vector (of dimension n) must be a codeword and that corresponds to the all-0 message (of dimension k). Linearity also implies that

$$c\left(\sum_{i=1}^{\ell} \alpha_i m_i\right) = \sum_{i=1}^{\ell} \alpha_i c(m_i),$$

for any $m_i \in \mathbb{F}^k$ and any $\alpha_i \in \mathbb{F}$ for $i = 1, \cdots, \ell$. The (n, k) specify the parameters for the linear code.

Let e_i be the k-dimensional vector with 1 (the identity over $\mathbb{F}$) in the ith position and 0 everywhere else. The e_i's for $i = 1, \cdots, k$ form a basis of the message space. This means that for a linear code, once we know the codewords $c(e_i)$ corresponding to these e_i's,we know entirely the encoding function. Indeed for $m \in \mathbb{F}^k$, we have that

$$c(m) = \sum_{i=1}^{k} m_i c(e_i).$$

Thus, there is no need for a codebook with 2^k entries. All we need is a matrix G that gives the linear transformation for all weight one messages into codewords; G is a $k \times n$ matrix (with entries in $\mathbb{F}$) and row i is the codeword coresponding to e_i, i.e. $c_i := c(e_i)$. This matrix G is called the *generating matrix*. Given G, a message m then gets encoded as $c = mG$. The codebook or code (or set of codewords) corresponding to the generating matrix G is

$$C = \{mG \in \mathbb{F}^n | m \in \mathbb{F}^k\}.$$

Of course, we would like the codewords to be all distinct, and for this, the only requirement is that the matrix G be of rank k (over the field $\mathbb{F}$), i.e. that the rows of G be linear independent.

Lemma 1. *Let $\mathbb{F}$ be a field, and let $G \in \mathbb{F}^{k \times n}$ and assume that the rows of G are linearly independent over $\mathbb{F}$ (i.e. that the rank of G is k). Then for any $m_1, m_2 \in \mathbb{F}^k$ with $m_1 \neq m_2$, we have that $m_1 G \neq m_2 G$.*

Proof. Suppose to the contrary that $m_1 G = m_2 G$ where $m_1 \neq m_2$. This means that $(m_1 - m_2)G = 0$, and letting $\alpha_i = (m_1 - m_2)_i \in \mathbb{F}$ be the ith component, we get that $\sum_i \alpha_i c_i = 0$ where the c_i's for $i = 1, \cdots, k$ are the k rows of G. Since $\alpha_i \neq 0$ for some i (as $m_1 \neq m_2$), this gives a contradiction with the fact that the rows of G are linearly independent. □

In linear algebra terminology, the set of codewords $C = \{mG : m \in \mathbb{F}^k\}$ correspond to the row space of G (over the field $\mathbb{F}$). We can assume that the generating matrix G takes a particular form. Indeed, if we perform elementary row operations on G (over $\mathbb{F}$, this means multiplying rows by a nonzero element or adding a row to another), we do not change the row space of G, i.e. the set C of codewords in our code. We can even permute columns without fundamentally changing the properties of our code. Therefore, using Gaussian elimination, we can assume that G takes the simpler form:

$$G = [I_{k \times k}\ S], \tag{1}$$

where S is $k \times (n-k)$.

This means that the first k elements of the codeword is just the message itself, and then there are $n-k$ redundacy elements. These are called *parity check elements*, or in the case of bits ($\mathbb{F} = \mathbb{Z}_2$), they are usually called *parity check bits*.

Instead of receiving a codeword $c \in \mathbb{F}^n$, we receive $\tilde{c} \in \mathbb{F}^n$. We define the *Hamming distance* $d_H(c_1, c_2)$ between two vectors $c_1, c_2 \in \mathbb{F}^n$ to be the number of entries in which they differ. For example, the Hamming distance d_H between the codewords $c_1 = (101201)$ and $c_2 = (102220)$ is 3. Denote d^* to be the minimum Hamming distance between any two distinct codewords of a code C as

$$d^* = d_{\min} = \min_{c_i, c_j \in C, c_i \neq c_j} d_H(c_i, c_j). \tag{2}$$

A code with minimum distance d^* between codewords can correct $\frac{d^*-1}{2}$ errors with nearest neighbor decoding. Thus, to correct one error, the minimum distance of the code must be at least 3.

The (Hamming) *weight* $w(s)$ of a vector $s \in \mathbb{F}^n$ is defined as the number of non-zero entries in s; this is also equal to $d_H(s, 0)$. For a linear code, the definition of the minimum distance d^* can be simplified. Indeed, for any two codewords c_i and c_j, we have that $d_H(c_i, c_j) = d_H(c_i - c_j, 0)$ and $c_i - c_j$ is a codeword. Thus, for a linear code, we have

$$d^* = \min_{c_i \in C, c_i \neq 0} w(c_i),$$

where as defined above, $w(c)$ denotes the Hamming weight.

2 Hamming Code

We first start with describing linear codes over Z_2 that can correct one error. This means that the weight of any nonzero codeword must be at least 3. Suppose our matrix G (in the form (1)) generates a single error correcting code. Then there are no codewords of weight 1 or 2, and the parity check portion S of G must satify the following 2 conditions:

- The weight of each row in S must be at least 2, since each row of the I part has weight exactly one.
- No two rows of S are identical. Otherwise, summing those two rows will give a codeword of weight 2.

These two conditions ensure that the minimum weight of any codeword is at least 3 (since summing 3 or more rows will give a codeword of weight at least 3 in the identity part). Thus, they are (necessary and) sufficient for defining a single error correcting code over $\mathbb{Z}_2$. S has $m = n - k$ parity checkbits, and the discussion above means that for fixed m, we will get the greatest possible k (and thus n) by having a row in S for each possible m-bit string with weight at least 2. There are exactly $k = 2^m - m - 1$ such strings, and thus $n = 2^m - 1$. This is what is known as the *Hamming* code with $(n, k) = (2^m - 1, 2^m - 1 - m)$, where $m = n - k$ is the number of check bits in the codeword. The simplest non-trivial code is for $m = 3$, which is the $(7, 4)$ Hamming code.

Example. The $(7, 4)$ Hamming code is given by the generating matrix:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

We know how to encode with any linear code. The codeword generated by message m is simply $c = mG$. But how about decoding? If no errors were introduced, the first k bits constitute the message. But errors might have been introduced.

Before we discuss how to decode when only one error has been introduced, let us first consider a general linear code (not necessarily a Hamming code) with generating matrix $G = [I_{k\times k}\ S] \in \mathbb{F}^{k\times n}$. Consider the so-called *parity matrix* H defined by:

$$H = \begin{bmatrix} -S \\ I_{(n-k)\times(n-k)} \end{bmatrix}.$$

When the field is just $\mathbb{Z}_2$, we can also write $H = \begin{bmatrix} S \\ I \end{bmatrix}$, as $S = -S$ over $\mathbb{Z}_2$. Since S has size $k \times (n-k)$, we have that the identity matrix is of size $(n-k)\times(n-k)$, and H has size $n \times (n-k)$. The important observation is that $GH = -S + S = 0$ (this is a $k \times (n-k)$ matrix consisting of all 0's). Therefore, since any codeword c can be written as $c = mG$ for some message m, we have that $cH = mGH = 0$. In fact, the parity-check matrix also defines the code in this way:

Theorem 1. *Let $\mathbb{F}$ be any finite field. Consider $S \in \mathbb{F}^{k\times(n-k)}$, and let $G = [I\ S]$ and $H^T = [-S^T\ I]$. Then,*

$$C = \{mG : m \in \mathbb{F}^k\} = \{c \in F^n : cH = 0\}.$$

Proof. We have already argued that $\{mG : m \in \mathbb{F}^k\} \subseteq \{c \in F^n : cH = 0\}$. Now consider $c \in \mathbb{F}^n$ such that $cH = 0$, and let m be the first k components of c, and q be the last $n-k$ components of c. $cH = 0$ means that $-mS + q = 0$, i.e. that $q = mS$. Therefore $c = [m\ mS] = mG$, as desired. □

As we said we could do row operations on G with changing the codebook $\{mG : m \in \mathbb{F}^k\}$, we can do column operations on the parity-check matrix H without changing the codebook $\{c \in F^n : cH = 0\}$.

Now that we have introduced the parity-check matrix for any linear code, let us consider the decoding in the case of a single error (as can be tolerated for the Hamming code). Assume that the codeword c has been corrupted in one place during the transmission. So instead of receiving c we have received $\tilde{c} = c + e_i$ for some (unknown) i. We can find the position of the error i by computing the *syndrome* $\tilde{c}H$ as this is supposed to be $cH + e_iH = e_iH$. Thus, $\tilde{c}H$ must be one of the rows of H and the index of the row indicates which bit has been corrupted. Once we know which bit was corrupted, we can recover the message.

Example: A $(7, 4)$ Hamming Code with the following generator matrix (the last 3 columns of each row must have at least two 1's, and no two rows are identical):

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Suppose we have the received string $\tilde{c} = (1110111)$. Then the syndrome is

$$\tilde{c}H = (111)$$

The string (111) corresponds to the 4th row of H, so we know that an error occurred in the 4th bit position. The correct received string should be (1111111) and the information bits sent are (1111), which are the first four bits in the received sequence. If $\tilde{c}H = (000)$, then no errors occurred and the information bits would be (1110).

Remark: To make the decoding slightly simpler, we could use the following observation: All the $n = 2^{n-k} - 1$ rows of H are distinct and nonzero, and thus corresponds to all possible nonzero binary strings of length $n - k$. We could then permute the rows of H (and similarly the columns of G) so that the ith row of H is the binary representation of i. With these permuted G and H, we can find the location of the error i since the binary representation of i is simply given by the syndrome $\tilde{c}H$.

Hamming codes can only tolerate any single error (while in Shannon's noisy coding, we were considering a (random) fraction p of errors), but their rate is almost 1. Their rate R is $\frac{k}{n} = \frac{2^m-1-m}{2^m-1} = 1 - \frac{m}{2^m-1}$ and converges to 1 as m grows. A repetition code in which every symbol is repeated 3 times could also correct a single error, but its rate is only 1/3.

Exercise: Suppose we would like to construct an error-correcting code over $\mathbb{Z}_p$ (p prime) able to correct one error. Extend the Hamming code construction above to $\mathbb{Z}_p$. (The main question is how to construct as many rows of S (over $\mathbb{Z}_p^m$) as possible so as to guarantee a minimum weight of 3 for any nonzero codeword.)

3 Perfect Binary Codes

Given that a binary code (linear or non-linear) has length n codewords, an error can occur in any of the n positions or no error has occurred. The number of check bits must be able to tell us this. Thus, for an (n, k) code, $2^{n-k} \geq n+1$. If this bound is achieved with equality, $n = 2^{n-k}-1$, then the code is a *perfect code*. For a single error correcting code with $n-k$ check bits, the number of information bits encoded cannot exceed $[2^{n-k}-1-(n-k)]$. For codes of which $(n, k) = (2^{n-k}-1, 2^{n-k}-1-(n-k))$,

note that every one of the 2^n possible received sequences is within 1 bit of some codeword. These types of codes are called *perfect single error correcting codes*. Every binary Hamming code is a perfect code. This is what is used in the solution of the hat problem mentioned in lecture or in the pset.

A code (non necessarily linear) of length n can correct t errors if every string of length n is within Hamming distance of t to at most one codeword. The minimum distance between codewords is $2t+1$. This t-error correcting code is perfect if every string of length n is within Hamming distance t of exactly one codeword. In other words, there is no space left in the codeword space after it is partitioned into the decoding spheres for each each codeword. There are very few perfect codes which correct more than one error.

4 Reed-Solomon Codes

We now discuss Reed-Solomon codes which allow to correct more errors than Hamming codes. They are used extensively in CDs, DVDs, satellite communications, etc. They are also linear codes, but they need a field with a larger size than $\mathbb{Z}_2$.

The basic idea of Reed-Solomon codes is the fundamental theorem of algebra that a polynomial $p(x) = a_0 + a_1 x + \cdots + a_d x^d = \sum_{i=0}^{d} a_i x^i$ of degree d with coefficients in a field $\mathbb{F}$ has at most d roots, unless the polynomial is the zero polynomial (of degree 0, with 0 as unique coefficient). This was proved in the lecture notes on modular arithmetic and algebra.

Theorem 2. *A polynomial of degree d over a field $\mathbb{F}$ is either the (identically) 0 polynomial or has at most d roots.*

One can then view the message to be encoded as the coefficients of the polynomial. If the message to be transmitted is $m = (m_0, m_1, \cdots, m_{k-1})$ consisting of k elements of $\mathbb{F}$, we want to consider the polynomial

$$p(x) = \sum_{i=0}^{k-1} m_i x^i,$$

of degree $k-1$. The Reed-Solomon encoding of it is its evaluation in n distinct points in F, say $b_0, \cdots, b_{n-1} \in \mathbb{F}$, thus the codeword is

$$c = (p(b_0), p(b_1), \cdots, p(b_{n-1})).$$

To be able to choose n distinct points we need that $|\mathbb{F}| \geq n$. To be able to transmit c, we want to consider a finite field $\mathbb{F}$. We have seen that $\mathbb{Z}_p$ with modular arithmetic forms a finite field with p elements if p is prime. Finite fields are known to exist if and only if $|\mathbb{F}|$ is a prime power p^k (where p is prime and k is an integer), but for simplicity, we will only discuss the case of $\mathbb{F} = \mathbb{Z}_p$ in these notes (although for practical purposes in digital transmission, the case of $|\mathbb{F}| = 2^k$ is more important). For $\mathbb{Z}_p$, we can choose $b_i = i$ for $i = 0, \cdots, n-1$ and, for those to be distinct, we need $p \geq n$.

Notice that Reed-Solomon codes are linear. Indeed, we have that $c = mG$ where G is the

following $k \times n$ matrix over $\mathbb{F}$:

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & \cdots & n-1 \\ 0 & 1^2 & 2^2 & \cdots & (n-1)^2 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 1^{k-1} & 2^{k-1} & \cdots & (n-1)^{k-1} \end{pmatrix}.$$

This generating matrix defines $|\mathbb{F}|^k$ codewords and for those to be distinct we need the rows of G to be linearly independent (over $\mathbb{Z}_p$). This can be shown to be the case for $p \geq n$, and this will be implied by the following theorem giving the minimum distance between the codewords of any two distinct messages.

Theorem 3. *The (Hamming) distance between the codewords of any two distinct messages is at least $n - k + 1$.*

Proof. By linearity, the minimum distance between the codewords corresponding to two distinct messages is equal to the minimum Hamming weight of the codeword c of a non-zero message m. This codeword c is $(p(0), p(1), \cdots, p(n-1))$ where p is the nonzero polynomial $p(x) = \sum_{i=0}^{k-1} m_i x^i$ of degree $k-1$. By Theorem 2, this polynomial $p(x)$ has at most $k-1$ roots, and thus there can be at most $k-1$ zeroes in the entries of c. Therefore the number of nonzeroes in c is at least $n - (k-1) = n - k + 1$. □

This means that if we choose $n = k + 2e$ then the minimum distance is $2e + 1$ and therefore the Reed-Solomon code can correct up to e errors by decoding to the closest codeword (to state the obvious, e, here, denotes the number of errors, and not the base of the natural logarithm...).

But how can we decode efficiently (rather than looking for the closest codeword)? Suppose we send the codeword $c = (p(0), p(1), \cdots, p(n-1))$ and we receive $\tilde{c} = (r_0, r_1, \cdots, r_{n-1})$. The next theorem is key for efficient decoding.

Theorem 4. *If $d_H(c, \tilde{c}) \leq e = \frac{n-k}{2}$ then there exist non-zero polynomials $f(x)$ of degree at most e and $q(x)$ of degree at most $k - 1 + e$ such that*

$$q(i) = r_i f(i) \text{ for all } i = 0, 1, \cdots, n-1.$$

Proof. Assume the errors/changes in $\tilde{c}$ occur in positions $i_1, i_2, \cdots, i_\ell$ where $\ell \leq e$. Define

$$f(x) = \prod_{j=1}^{\ell} (x - i_j),$$

and $q(x) = p(x)f(x)$. $f(x)$ is a low degree polynomial that vanishes on all the indices of the errors. Notice that $f(x)$ is not identically zero and so is $q(x)$. Furthermore, the degree of $f(x)$ is $\ell \leq e$ and the degree of $q(x)$ is $\leq k - 1 + e$.

To prove the claim, consider any index i. If i is not an error then $r_i = p(i)$ and thus $q(i) = p(i)f(i) = r_i f(i)$. If i is an error then $f(i) = 0$ by definition and thus $q(i) = 0 = r_i f(i)$. In both cases, we have that $q(i) = r_i f(i)$. □

To find $f(x)$ and $q(x)$ we can view their coefficients as variables and the relationships $q(i) = r_i f(i)$ for $i = 0, \cdots, n-1$ as linear equations in these variables. We thus have a system of n equations in $(e+1)+(k+e) = n+1$ variables, and we can find a non-zero solution to it (whose existence is guaranteed by the above theorem whenever $d_H(c, \tilde{c}) \leq e$).

Once we have a solution $\hat{f}(x)$ and $\hat{q}(x)$ satisfying $\hat{q}(i) = r_i \hat{f}(i)$ for all $0 \leq i \leq n-1$, we claim that $\hat{q}(x) = p(x)\hat{f}(x)$ for all $x \in \mathbb{Z}_p$. To see this, observe that $\hat{q}(x) - p(x)\hat{f}(x)$ is a polynomial of degree $k+e-1$ that vanishes in all i for which $p(i) = r_i$. That means that it has at least $n-e = k+e$ roots. Since it has more roots than its degree, $\hat{q}(x) - p(x)\hat{f}(x)$ must be the 0 polynomial. So we can recover $p(x)$ (and thus the initial message m) by dividing $\hat{q}(x)$ by $\hat{f}(x)$. The quotient is guaranteed to be $p(x)$ and there will be no remainder.

Example. Consider the following example with $\mathbb{F} = \mathbb{Z}_7$ ($n = 7$), and $k = 3$. So it can tolerate up to $e = \frac{7-3}{2} = 2$ errors. Consider the message $m = (2, 0, 5) \in \mathbb{Z}_7^3$. This corresponds to the polynomial $p(x) = 2 + 5x^2$, and therefore to the codeword $c = (p(0), p(1), p(2), p(3), p(4), p(5), p(6)) = (2, 0, 1, 5, 5, 1, 0)$ (all the operations have been done over $\mathbb{Z}_7$). Instead of c, suppose we receive $r = (2, 2, 1, 0, 5, 1, 0)$. Observe that $d_H(c, r) = 2 \leq e$, so we should be able to recover our message m from r.

To decode r, we use Theorem 4. The unknown polynomial $f(x)$ of degree at most 2 is say $f(x) = f_0 + f_1 x + f_2 x^2$, while the unknown $q(x)$ of degree $k-1+e = 4$ is $q(x) = q_0 + q_1 x + q_2 x^2 + q_3 x^3 + q_4 x^4$. We have thus 8 unknowns, f_0, f_1, f_2 and q_0, q_1, q_2, q_3 and q_4. We can write 7 linear constraints:

$$\begin{array}{lll}
q(0) = r_0 f(0): & q_0 & = 2f_0 \\
q(1) = r_1 f(1): & q_0 + q_1 + q_2 + q_3 + q_4 & = 2(f_0 + f_1 + f_2) \\
q(2) = r_2 f(2): & q_0 + 2q_1 + 4q_2 + q_3 + 2q_4 & = 1(f_0 + 2f_1 + 4f_2) \\
q(3) = r_3 f(3): & q_0 + 3q_1 + 2q_2 + 6q_3 + 4q_4 & = 0 \\
q(4) = r_4 f(4): & q_0 + 4q_1 + 2q_2 + q_3 + 4q_4 & = 5(f_0 + 4f_1 + 2f_2) \\
q(5) = r_5 f(5): & q_0 + 5q_1 + 4q_2 + 6q_3 + 2q_4 & = 1(f_0 + 5f_1 + 4f_2) \\
q(6) = r_6 f(1): & q_0 + 6q_1 + q_2 + 6q_3 + q_4 & = 0
\end{array}$$

That is a system of 7 linear equations in 8 unknowns, and so it must have a nonzero solution (which we can find for example by Gaussian elimination over $\mathbb{Z}_7$). One of these solutions is $q^* = (2, 2, 1, 5, 4)$ and $f^* = (1, 5, 5)$, corresponding to $q^*(x) = 2 + 2x + x^2 + 5x^3 + 4x^4$ and $f^*(x) = 1 + 5x + 5x^2$. We could have taken another solution, for example $3q^*$ for q and $3f^*$ for f, as our system is homogeneous (the constant term is always 0). Now consider the polynomial $q^*(x) - p(x)f^*(x)$ of degree 4 (we still don't know $p(x)$ but that's OK). This polynomial has at least $n - e = 5$ roots and therefore must be the zero polynomial. But this means that we can recover $p(x)$ by

$$p(x) = \frac{q^*(x)}{f^*(x)} = \frac{4x^4 + 5x^3 + x^2 + 2x + 2}{5x^2 + 5x + 1} = 5x^2 + 2,$$

over $\mathbb{Z}_7$. This means that the original message was $m = (2, 0, 5)$.

5 Faster Decoding

The algorithm we have seen so far requires the solution of a system of linear equations of size n (by $n+1$) and this is the bottleneck operation. And even though we focused on the case when the field F is $\mathbb{Z}_p$, the same algorithm would work as well for any other finite field.

Here we will describe a faster decoding algorithm for the specific case in which $F = \mathbb{Z}_p$; this algorithm was patented by Berlekamp and Welch in 1983. In the case of $\mathbb{Z}_p$, we can evaluate our polynomial at the specific values $b_i = i$ for $i = 0, \cdots, n-1$, and this will considerably simply decoding (while we did not exploit this in the previous section).

Before we discuss the decoding algorithm, we go on a little detour, and consider basic properties of polynomials and, in particular, how to reconstruct the coefficients of a polynomial from its evaluation at a a number of equidistant values. Given a sequence (finite or infinite) $a = a_0, a_1, a_2, \cdots$, we define its *first difference* $\Delta a = a_1 - a_0, a_2 - a_1, \cdots,$. For now, we consider the sequence a to be infinite, but if it was finite then Δa is simply one unit shorter than a. One important observation is given by the following lemma.

Lemma 2. *If the sequence a satisfies $a_i = p(i)$ (for any $i \geq 0$) where $p(x)$ is a polynomial of degree d, then there exists a polynomial $q(x)$ of degree $d-1$ such that $(\Delta a)_i = q(i)$.*

The lemma simply follows from observing that $q(x) = p(x+1) - p(x)$ is indeed a polynomial, its degree is one less than the degree of p, and its values over the integers correspond to Δa.

We can apply this first difference operator multiple times, and define inductively $\Delta^k a = \Delta(\Delta^{k-1} a)$ for any $k \geq 2$ (where $\Delta^1 = \Delta$). Apply this to the above lemma, we get:

Lemma 3. *If the sequence a satisfies $a_i = p(i)$ where $p(x)$ is a polynomial of degree d then $\Delta^{d+1} a$ is the 0 sequence.*

Proof. By applying Lemma 2 d times, we get that $\Delta^d a$ corresponds to the values of a polynomial of degree 0 (thus a constant), meaning that $(\Delta^d a)_i = (\Delta^d a)_{i+1}$ for any i. Applying the Δ operator one more time, we get the 0 sequence. □

For example, if we consider the polynomial $p(x) = 2x^3 + x$ (over the field $\mathbb{R}$), we have that:

a	0	3	18	57	132	255	438	$\cdots$
Δa	3	15	39	75	123	183	$\cdots$	
$\Delta^2 a$	12	24	36	48	60	$\cdots$		
$\Delta^3 a$	12	12	12	12	$\cdots$			
$\Delta^4 a$	0	0	0	$\cdots$				

As an exercise, compute the polynomials that correspond to $\Delta^k a$. Now suppose, we are given the above calculations, but we do not know the polynomial corresponding to the sequence a. How could we reconstruct it? More precisely, suppose we know that $a_i = p(i)$ for some polynomial of degree d, how can we reconstruct $p(x)$ from the knowledge of a_0, $(\Delta a)_0, (\Delta^2 a)_0, \cdots, (\Delta^d a)_0$ (and $(\Delta^{d+1} a)_0 = 0$)? This is given by the following theorem (which is true for any field):

Theorem 5. *Let $p(x)$ be a polynomial of degree d and $a_i = p(i)$ for $i \in \mathbb{N}$. Then*

$$p(x) = a_0 C_0(x) + (\Delta a)_0 C_1(x) + (\Delta^2 a)_0 C_2(x) + \cdots + (\Delta^d a)_0 C_d(x),$$

where

$$C_k(x) = (k!)^{-1}[x(x-1)(x-2)\cdots(x-k+1)],$$

and $C_0(x) = 1$.

We won't give a detailed proof, but the argument follows by showing that if $b_i = C_k(i)$ for every $i \in N$ then $(\Delta^\ell b)_0 = \delta_{k\ell}$. Theorem 5 is just expressing the polynomial $p(x)$ in the basis of the $\{C_k(x)\}$, rather than the more usual basis of the monomials $\{x^k\}$.

For the example above, we have $a_0 = 0$, $(\Delta a)_0 = 3$, $(\Delta^2 a)_0 = 12$ and $(\Delta^3 a)_0 = 12$, and thus the theorem gives (over the reals):

$$p(x) = 3C_1(x) + 12C_2(x) + 12C_3(x) = 3x + 12\frac{1}{2}x(x-1) + 12\frac{1}{6}x(x-1)(x-2) = 2x^3 + x.$$

We are now ready to go back to the faster decoding algorithm for Reed-Solomon codes over $\mathbb{Z}_p$. Consider the (unknown) polynomials $f(x)$ and $q(x)$ given in Theorem 4. From this theorem, we know that the sequence $a_i = r_i f(i)$ where $i = 0, 1, \cdots, n-1$ corresponds to a polynomial $(q(x))$ of degree $\leq k - 1 + e$, and therefore we have that

$$(\Delta^{k+e} a)_i = 0$$

for $0 \leq i \leq n - 1 - (k + e) = e - 1$. But we also know that $f(x)$ is a polynomial of degree at most e, and therefore

$$f(x) = v_0 + v_1 x + v_2 x^2 + \cdots + v_e x^e.$$

Therefore the sequence a can be expressed as

$$a = \sum_{j=0}^{e} v_j b^{(j)}, \tag{3}$$

where the sequence $b^{(j)}$ is defined by $b_i^{(j)} = r_i i^j$ for $i \in \mathbb{N}$. We do not know (yet) the v_j's but we do know the sequences $b^{(j)}$. Applying the operator Δ to equation (3), we obtain:

$$0 = (\Delta^{k+e} a)_i = \sum_{j=0}^{e} v_j (\Delta^{k+e} b^{(j)})_i,$$

for $0 \leq i \leq e - 1$. This can be viewed as e linear equations in the $e + 1$ unknowns v_j, and therefore one can find a non-zero solution v to this system of equation. This is a much smaller system (of order e) compared to the system we solved in the general Reed-Solomon decoding algorithm (which was of order n). For example, if $e = 0.05k$ then $n = 1.1k$ and this system is more than 20 times smaller. But we are not done yet. We have determined the polynomial $f(x)$ but we still need to determine $q(x)$. For this purpose, we will use Theorem (5). Indeed, we know that $q(x)$ is a polynomial of degree at most $k - 1 + e$ and therefore we can reconstruct $q(x)$ by using Theorem (5) and the fact that

$$(\Delta^\ell a)_0 = \sum_{j=0}^{e} v_j (\Delta^\ell b^{(j)})_0,$$

and these are quantities we can evaluate. Once we have $q(x)$ and also $f(x)$, we can then compute $p(x)$ by doing the division of $q(x)$ by $f(x)$.

This decoding algorithm for Reed-Solomon over $\mathbb{Z}_p$ has the main advantage that, instead of solving a linear system of order n, one can simply solve a smaller linear system of size e.

Example. Consider the same example as in the previous section. Suppose $k = 3$, $e = 2$ and thus $n = 7$. Our field $F = \mathbb{Z}_7$, which is large enough for $n = 7$. Consider the message

$m = (2, 0, 5) \in \mathbb{Z}_7^3$. This corresponds to the polynomial $p(x) = 2 + 5x^2$, and therefore to the codeword $c = (p(0), p(1), p(2), p(3), p(4), p(5), p(6)) = (2, 0, 1, 5, 5, 1, 0)$. Instead of c, suppose we receive $r = (2, 2, 1, 0, 5, 1, 0)$. Observe that $d_H(c, r) = 2 \leq e$, so we should be able to recover our message m from r.

The faster decoding algorithm proceeds as follows. Compute $b^{(j)} := (r_i i^j)_{0 \leq i \leq 6}$ for $j = 0, 1, e = 2$, and $k + e = 5$ levels of repeated differences:

$b^{(0)}$	2	2	1	0	5	1	0
$\Delta b^{(0)}$	0	6	6	5	3	6	
$\Delta^2 b^{(0)}$	6	0	6	5	3		
$\Delta^3 b^{(0)}$	1	6	6	5			
$\Delta^4 b^{(0)}$	5	0	6				
$\Delta^5 b^{(0)}$	**2**	**6**					

$b^{(1)}$	0	2	2	0	6	5	0
$\Delta b^{(1)}$	2	0	5	6	6	2	
$\Delta^2 b^{(1)}$	5	5	1	0	3		
$\Delta^3 b^{(1)}$	0	3	6	3			
$\Delta^4 b^{(1)}$	3	3	4				
$\Delta^5 b^{(1)}$	**0**	**1**					

$b^{(1)}$	0	2	4	0	3	4	0
$\Delta b^{(1)}$	2	2	3	3	1	3	
$\Delta^2 b^{(1)}$	0	1	0	5	2		
$\Delta^3 b^{(1)}$	1	6	5	4			
$\Delta^4 b^{(1)}$	5	6	6				
$\Delta^5 b^{(1)}$	**1**	**0**					

The values of $\Delta^5 b^{(j)}$ give us a system of $e = 2$ equations in $e + 1 = 3$ unknowns:

$$\begin{cases} 2v_0 + 0v_1 + 1v_2 = 0 \\ 6v_0 + 1v_1 + 0v_2 = 0 \end{cases}$$

Choose any non-zero solution to this system, for example choosing $v_0 = 1$ implies that $v_1 = 1$ and $v_2 = 5$. This means that $f(x) = 1 + 5x + 5x^2$. (We do not need to factor $f(x)$, but if we were, we would find that $f(x) = 5(x - 1)(x - 3)$, implying that the errors were in positions 1 and 3 (with 0 as the starting position).)

Now that we know the v_j's, we can reconstruct $q(x)$ from Theorem 5. Letting $a_i = q(i)$, we can reconstruct

$$\Delta^k a = v_0 \Delta^k b^{(0)} + v_1 \Delta^k b^{(1)} + v_2 \Delta^k b^{(2)},$$

i.e.

$$(\Delta^0 a, \Delta^1 a, \Delta^2 a, \Delta^3 a, \Delta^4 a) = (2, 5, 4, 6, 5).$$

To use Theorem 5, we need to expand the $C_k(x) = (k!)^{-1}[x(x-1)(x-2)\cdots(x-k+1)]$ (which we can do just once, irrespctive of the message in advance):

$$\begin{cases} C_0(x) = 1 \\ C_1(x) = x \\ C_2(x) = 4x^2 + 3x \\ C_3(x) = 6x^3 + 3x^2 + 5x \\ C_4(x) = 5x^4 + 5x^3 + 6x^2 + 5x. \end{cases}$$

Now we can find $q(x)$:

$$q(x) = 2C_0(x) + 5C_1(x) + 4C_2(x) + 6C_3(x) + 5C_4(x) = 4x^4 + 5x^3 + x^2 + 2x + 2.$$

Finally, we can derive $p(x)$ by

$$p(x) = \frac{q(x)}{f(x)} = \frac{4x^4 + 5x^3 + x^2 + 2x + 2}{5x^2 + 5x + 1} = 5x^2 + 2,$$

over $\mathbb{Z}_7$. This means that the original message was $m = (2, 0, 5)$.

18.200 Principles of Discrete Applied Mathematics

Spring 2024