1 Error Correcting Codes

In this note, we will discuss the problem of transmitting messages across an unreliable communication channel. The channel may cause some parts of the message ("packets") to be lost, or dropped; or, more seriously, it may cause some packets to be corrupted. We will learn how to "encode" the message by introducing redundancy into it in order to protect against both of these types of errors. Such an encoding scheme is known as an "error correcting code." Error correcting codes are a major object of study in mathematics, computer science and electrical engineering; they belong to a field known as "Information Theory." In addition to the beautiful theory underlying them (which we will glimpse in this note), they are of great practical importance: every time you use your cellphone, satellite TV, DSP, cable modem, disk drive, CD-ROM, DVD player etc., or send and receive data over the internet, you are using error correcting codes to ensure that information is transmitted reliably.

There are, very roughly speaking, (at least) two distinct flavors of error correcting codes: algebraic codes, which are based on polynomials over finite fields, and combinatorial codes, which are based on graph theory. In this note we will focus on algebraic codes, and in particular on so-called Reed-Solomon codes (named after two of their inventors). In doing so, we will be making essential use of the properties we learned about polynomials in the last lecture.

2 Erasure Errors

We will consider two situations in which we wish to transmit information on an unreliable channel. The first is exemplified by the Internet, where the information (say a file) is broken up into packets, and the unreliability is manifest in the fact that some of the packets are lost during transmission, as shown below:

We refer to such errors as *erasure errors*. Suppose that the message consists of n packets and suppose that at most k packets are lost during transmission. We will show how to encode the initial message consisting of n packets into a redundant encoding consisting of $n+k$ packets such that the recipient can reconstruct the message from *any* n received packets. Note that in this setting the packets are labeled with headers, and thus the recipient knows exactly which packets were dropped during transmission.

2.1 Repetition Coding

Let us first consider a very simple error correcting scheme. Since we are concerned about packets being dropped, we can simply repeat every packet "enough" times — so long as at least one copy of every packet gets through, our recipient will be able to determine the full message. This strategy is known as *repetition coding*.

While this idea will certainly work, it turns out to not be very efficient. We have to send all our packets before finding out which ones are dropped, so we can't simply say "resend a packet if it is dropped". Instead, in

order to protect against k erasures, we have to send $k+1$ copies of every packet; if we sent any fewer, it is possible for all copies of one packet to be dropped, at which point our recipient will not be able to recover the entire message. Thus, in order to guarantee recovery of an n packet message, we need to send $n(k+1)$ total packets.

The problem with repetition coding is that each extra packet we send only depends on a single part of the original message. Hence, if we lose all the extra packets corresponding to one part, that part is irretrievable, requiring us to send a large number of packets for *each part* of the message. In order to improve upon this, we need the values in the extra packets to depend on more than just one part of the message; ideally, we would like each packet to encode information about *every part* of the message all at once. This may seem impossible at first, but it turns out that the polynomials we studied in note 8 provide a way to do this!

2.2 Reed-Solomon Coding

For the purposes of encoding messages using polynomials, we can assume without loss of generality that the content of each packet is a number modulo q, where q is a prime. For example, the content of the packet might be a 32-bit string and can therefore be regarded as a number between 0 and $2^{32}-1$; then we could choose q to be any prime larger than 2^{32}.

We now describe our scheme. In order to encode a message $(m_1, m_2, ..., m_n)$ (where each m_i is a number in $GF(q)$), we will interpolate a polynomial p of degree at most $n-1$ through the points $(1, m_1)$, $(2, m_2)$, ..., (n, m_n); from note 8, we know that we can create such a polynomial using Lagrange interpolation. We then send $n+k$ packets, where the ith packet is $p(i)$.[1]

Now suppose our recipient receives at least n of the packets we sent. Because they know which packets they received (and hence know the x values of each of them), they can interpolate a degree at most $n-1$ polynomial $p'(x)$ through the points they got. They then interpret the message as $(p'(1), ..., p'(n))$.

To see why this works, notice that p' agrees with our original polynomial p on at least n points — in particular, they agree on the points corresponding to the n packets that made it through the channel. Two distinct degree $n-1$ polynomials can only agree on $n-1$ points, so we must have that p and p' are not distinct, meaning that $p = p'$. Thus, for each $1 \leq i \leq n$, we have that $p'(i) = p(i) = m_i$, so our recipient successfully got all n packets we were trying to send!

Example

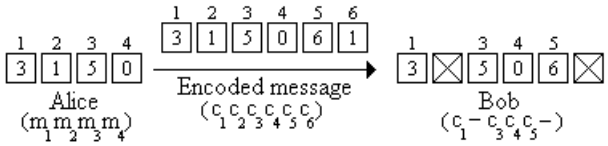
Suppose Alice wants to send Bob a message of $n = 4$ packets and she wants to guard against $k = 2$ lost packets. Then, assuming the packets can be coded up as integers between 0 and 6, Alice can work over $GF(7)$ (since $7 \geq n+k = 6$; of course, in real applications, we would be working over a much larger field!). Suppose the message that Alice wants to send to Bob is $m_1 = 3$, $m_2 = 1$, $m_3 = 5$, and $m_4 = 0$. The unique polynomial of degree $n-1 = 3$ described by these 4 points is $P(x) = x^3 + 4x^2 + 5$.

Exercise. We derived this polynomial using Lagrange interpolation mod 7. Check this derivation, and verify also that indeed $P(i) = m_i$ for $1 \leq i \leq 4$.

Since $k = 2$, Alice must evaluate $P(x)$ at 2 extra points: $P(5) = 6$ and $P(6) = 1$. Now, Alice can transmit the encoded message which consists of $n+k = 6$ packets, where $c_j = P(j)$ for $1 \leq j \leq 6$. So Alice will send $c_1 = P(1) = 3$, $c_2 = P(2) = 1$, $c_3 = P(3) = 5$, $c_4 = P(4) = 0$, $c_5 = P(5) = 6$, and $c_6 = P(6) = 1$.

[1] Note here that the extra packets sent do indeed depend on *all* of the original packets. Indeed, if we change the value of a single original packet, the values of every extra packet will change as well.

Now suppose packets 2 and 6 are dropped, in which case we have the following situation:



From the values that Bob received (3, 5, 0, and 6), he uses Lagrange interpolation and computes the following basis polynomials (where everything should be interpreted mod 7):

$$\Delta_1(x) = \frac{(x-3)(x-4)(x-5)}{-24} \equiv 2(x-3)(x-4)(x-5) \pmod 7$$
$$\Delta_3(x) = \frac{(x-1)(x-4)(x-5)}{4} \equiv 2(x-1)(x-4)(x-5) \pmod 7$$
$$\Delta_4(x) = \frac{(x-1)(x-3)(x-5)}{-3} \equiv 2(x-1)(x-3)(x-5) \pmod 7$$
$$\Delta_5(x) = \frac{(x-1)(x-3)(x-4)}{8} \equiv (x-1)(x-3)(x-4) \pmod 7.$$

(Note that we have used the fact here that the inverses of -24, 4, -3 and 8 (mod 7) are 2, 2, 2 and 1 respectively.) He then reconstructs the polynomial $P(x) = 3 \cdot \Delta_1(x) + 5 \cdot \Delta_3(x) + 0 \cdot \Delta_4(x) + 6 \cdot \Delta_5(x) = x^3 + 4x^2 + 5 \pmod 7$. Bob then evaluates $m_2 = P(2) = 1$, which is the packet that was lost from the original message. More generally, no matter which two packets were dropped, following exactly the same method Bob can always reconstruct $P(x)$ and thus the original message.

Exercise. Check Bob's calculation above, and verify that he really does reconstruct the correct polynomial, as claimed. Remember that all arithmetic must be done mod 7.

2.3 Reed-Solomon Optimality

Now that we have a scheme that works, our natural next question should be *can we do better?* As it turns out, the answer is no — you cannot create a perfectly reliable scheme with fewer than k extra packets. To see why this is, consider any scheme which sends only $k-1$ extra packets to protect against k erasures. What happens if the channel drops all $k-1$ extra packets, and in addition drops the nth packet of the message? At this point, all the recipient gets is the first $n-1$ packets of the message, which contain no information about what the nth packet was! Thus, no matter how clever we are in our use of extra packets, there is always the possibility that our recipient will receive something ambiguous, from which they cannot determine the full message we intended to send.

3 General Errors

Let us now consider a much more challenging scenario. Suppose that Alice wishes to communicate with Bob over a noisy channel (say, via a modem). Her message is $m_1, \ldots, m_n$, where we may think of the m_i's as characters (either bytes or characters in the English alphabet). The problem now is that some of the characters are *corrupted* during transmission due to channel noise. Thus Bob receives exactly as many characters as Alice transmits, but k of them are corrupted, and Bob has no idea which k these are! Recovering from such general errors is much more challenging than recovering from erasure errors, though once again

polynomials hold the key. As we shall see, Alice can still guard against k general errors, at the expense of transmitting only $2k$ additional characters (twice as many as in the erasure case we saw above).

Let us first consider what would happen if we tried applying our Reed-Solomon encoding exactly as before. Suppose that the first n packets through the channel are consistent with the message $(m_1, ..., m_{n-1}, m_n)$ but the k extra packets are instead consistent with $(m_1, ..., m_{n-1}, m'_n)$ for some $m'_n \neq m_n$. There are (at least) two possibilities here: either the original message was $(m_1, ..., m_{n-1}, m_n)$ and the last k packets got corrupted, or the original message was $(m_1, ..., m_{n-1}, m'_n)$ and only the nth packet got corrupted. Just from the packets received and the knowledge that at most k of them were corrupted, it's impossible to know what the intended message was!

In fact, we can say something more general than this: no scheme, no matter how clever, that uses fewer than $2k$ extra packets can guarantee recovery from up to k corruptions. To see why this is, consider any any scheme which uses only $2k-1$ packets. As before, there are two possible messages that might have been sent: either $(m_1, ..., m_{n-1}, m_n)$ or $(m_1, ..., m_{n-1}, m'_n)$. Let the $2k-1$ extra packets for the first message be $(e_1, ..., e_{2k-1})$, while those for the second message are $(e'_1, ..., e'_{2k-1})$.

Now what happens if the first n packets we receive are $(m_1, ..., m_n)$, while the $2k+1$ additional packets are $(e_1, ..., e_{k-1}, e'_k, ..., e'_{2k-1})$? There are two possibilities here: either the first message was sent but the last k extra packets were corrupted, or the second message was sent but the last original packet and the first $k-1$ extra ones are wrong.

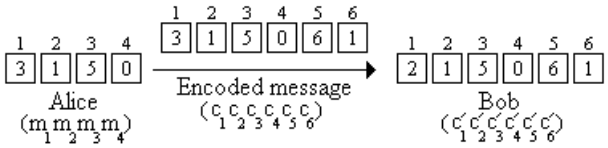
$$(m_1, m_2, ..., m_{n-1}, m_n, e_1, ..., e_{k-1}, \boxed{e_k, ..., e_{2k-1}})$$
$$(m_1, m_2, ..., m_{n-1}, \boxed{m'_n, e'_1, ..., e'_{k-1}}, e'_k, ..., e'_{2k-1})$$
$$\downarrow$$
$$(m_1, m_2, ..., m_{n-1}, m_n, e_1, ..., e_{k-1}, e'_k, ..., e'_{2k-1})$$

Just from the information available to us, we can't differentiate between these two cases, so we cannot be guaranteed to determine which of the two possible messages was the intended one.

3.1 Reed-Solomon Codes For General Errors

What the last section tells us is that the failure of our original Reed-Solomon protocol in the case of general errors might not be caused by the scheme being bad, but simply by it not sending enough extra packets. Indeed, by only sending k extras, it had no hope of being able to recover from k corruptions, no matter how good of a scheme it was! Thus, we might naturally wonder what happens if we run the same protocol as Section 2.2 but sending $2k$ extra packets instead of just k, meaning that our encoded message includes all the way out to $p(n+2k)$ instead of just out to $p(n+k)$.

As an example of how this might work, suppose that Alice wishes to send $n = 4$ characters to Bob via a modem in which $k = 1$ of the characters is corrupted, she must redundantly send an encoded message consisting of 6 characters. Suppose she wants to transmit the same message (3051) as in our erasure example above, and that c_1 is corrupted from 3 to 2. This scenario can be visualized in the following figure:



From Bob's viewpoint, the problem of reconstructing Alice's message is the same as reconstructing the polynomial $P(x)$ from the $n+2k$ received characters $r_1, r_2, \ldots, r_{n+2k}$. In other words, Bob is given $n+2k$

values, $r_1, r_2, \ldots, r_{n+2k}$ modulo q, with the promise that there is a polynomial $P(x)$ of degree $n-1$ over $GF(q)$ such that $P(i) = r_i$ for $n+k$ distinct values of i between 1 and $n+2k$. Bob must reconstruct $P(x)$ from this data (in the above example, $n+k=5$, and $r_2 = P(2) = 1$, $r_3 = P(3) = 5$, $r_4 = P(4) = 0$, $r_5 = P(5) = 6$, and $r_6 = P(6) = 1$). Note, however, that Bob does not know *which* of the $n+k$ values are correct!

Does Bob even have sufficient information to reconstruct $p(x)$? As it turns out, he does! All Bob needs to do is find a set of $n+k$ of the received points that all lie on some degree at most $n-1$ polynomial $p'(x)$. In order to show that this works, we need two things: first that there will always exist such a set of $n+k$ points, and second that the polynomial p' that Bob finds will actually be p.

The former point is rather straightforward: since at most k of the $n+2k$ packets get corrupted, the $n+k$ uncorrupted packets will all still lie on $p(x)$, which is indeed degree at most $n-1$ by construction.

For the latter point, suppose that we have some degree at most $n-1$ polynomial p' that goes through some set of $n+k$ received points. At most k of those received points can be corrupted, so n of them must be uncorrupted. But that means both p and p' pass through those n points. Two distinct polynomials of degree $n-1$ can only agree on $n-1$ points, so the only way for p and p' to agree on n points is if $p' = p$!

What this means is that recovery is always possible, no matter what k packets get corrupted. But how does Bob know which $n+k$ points to interpolate a degree $n-1$ polynomial through? He has no idea *a priori* which points were corrupted, so he just has to try all possible sets of $n+k$ points to see which ends up being the uncorrupted set. But this is highly inefficient — there are $\binom{n+2k}{k} \approx (\frac{n+2k}{k})^k$ different sets to try, which is highly impractical for real-world values of n and k.

Is there a way for us to more efficiently recover the original polynomial p? This was an open problem in Computer Science for over 25 years; the fact that recovery is possible was discovered in 1960, but it wasn't until 1986 that Elwyn Berlekamp and Lloyd Welch formulated an efficient recovery algorithm, which we describe in the next section.

3.2 Berlekamp-Welch Algorithm

The key to the Berlekamp-Welch Algorithm is a clever use of the so-called *error-locator polynomial*

$$e(x) = (x-e_1)(x-e_2)\cdots(x-e_k)$$

where e_i is the position of the ith error. Note that when performing the algorithm, we do not explicitly know this polynomial, as we do not know the error positions. Instead, we will use the polynomial symbolically in such a way that we are eventually able to solve for the values of all its coefficients!

Let us make a simple but crucial observation about this polynomial:

$$p(i)e(i) = r_i e(i) \qquad \text{for } 1 \leq i \leq n+2k \tag{1}$$

where r_i is the received value of the ith packet. To see this, note that it holds at points i at which no error occurred since at those points $p(i) = r_i$; and it is trivially true at points i at which an error occurred since then $e(i) = 0$. Looking more closely at the equalities in (1), we will show that they in fact correspond to $n+2k$ linear equations in $n+2k$ unknowns, from which the locations of the errors and coefficients of $p(x)$ can be easily deduced.

To this end, define $q(x) := p(x)e(x)$. We know that p has degree $n-1$ and e has degree k, so q has degree $n+k-1$. Thus, q can be described by $n+k$ coefficients $a_0, a_1, \ldots, a_{n+k-1}$:

$$q(x) = a_{n+k-1}x^{n+k-1} + \ldots + a_1 x + a_0$$

Similarly, the error-locator polynomial $e(x) = (x - e_1) \cdots (x - e_k)$ has degree k and is described by $k+1$ coefficients $b_0, b_1, \ldots, b_k$:

$$e(x) = b_k x^k + \ldots + b_1 x + b_0$$

But we can notice that based on how e is defined, its leading coefficient b_k will always be 1. Hence, only the remaining k coefficients are unknown to us.

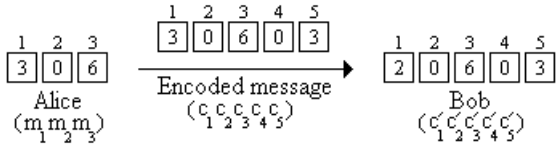
Once we fix a value i for x, the received value r_i is fixed. Also, $q(i)$ is now a linear function of the $n+k$ coefficients $a_{n+k-1} \ldots a_0$, and $e(i)$ is a linear function of the k coefficients $b_{k-1} \ldots b_0$. Therefore the equation $q(i) = r_i e(i)$ (which just comes from rewriting (1) using $q(x) = p(x)e(x)$) is a linear equation in the $n+2k$ unknowns $a_{n+k-1}, \ldots, a_0$ and $b_{k-1}, \ldots, b_0$. We thus have $n+2k$ linear equations, one for each value of i, and $n+2k$ unknowns. We can solve these equations and get $e(x)$ and $q(x)$. We can then compute the ratio $\frac{q(x)}{e(x)}$ to obtain $p(x)$. This is best illustrated by an example.

Example. Suppose we are working over $GF(7)$ and Alice wants to send Bob the $n = 3$ characters "3," "0," and "6" over a modem. Turning to the analogy of the English alphabet, this is equivalent to using only the first 7 letters of the alphabet, where $a = 0, \ldots, g = 6$. So the message which Alice wishes for Bob to receive is "dag". Then Alice interpolates (e.g., using Lagrange: exercise!) to find the polynomial

$$p(x) = x^2 + x + 1,$$

which is the unique polynomial of degree 2 such that $p(1) = 3$, $p(2) = 0$, and $p(3) = 6$.

Suppose Alice knows that $k = 1$ character will be corrupted; then she needs to transmit the $n + 2k = 5$ characters $p(1) = 3$, $p(2) = 0$, $p(3) = 6$, $p(4) = 0$, and $p(5) = 3$ to Bob. Suppose $p(1)$ is corrupted, and Bob receives the character 2 instead of 3 (i.e., Alice sends the encoded message "dagad" but Bob instead receives "cagad"). Summarizing, we have the following situation:



Let $e(x) = (x - e_1) = x + b_0$ be the error-locator polynomial—remember, Bob doesn't know what $e_1 = -b_0$ is yet since he doesn't know where the error occurred—and let $q(x) = a_3 x^3 + a_2 x^2 + a_1 x + a_0$ (where again the coefficients a_i are unknown).

Now Bob just substitutes $i = 1$, $i = 2$, …, $i = 5$ into the equations $q(i) = r_i e(i)$ from (1) above and simplifies to get five linear equations in five unknowns (recall that we are working modulo 7 and that r_i is the value Bob received for the i-th character):

$$\begin{aligned}
a_3 + a_2 + a_1 + a_0 + 5b_0 &= 2 \\
a_3 + 4a_2 + 2a_1 + a_0 &= 0 \\
6a_3 + 2a_2 + 3a_1 + a_0 + b_0 &= 4 \\
a_3 + 2a_2 + 4a_1 + a_0 &= 0 \\
6a_3 + 4a_2 + 5a_1 + a_0 + 4b_0 &= 1
\end{aligned}$$

Bob then solves this linear system and finds that $a_3 = 1$, $a_2 = 0$, $a_1 = 0$, $a_0 = 6$ and $b_0 = 6$ (all mod 7). (As a check, this implies that $E(x) = x + 6 = x - 1$, so the location of the error is position $e_1 = 1$, which is correct since the first character was corrupted from a "d" to a "c".) This gives him the polynomials $q(x) = x^3 + 6$ and $e(x) = x - 1$. He can then find $p(x)$ by computing the quotient $p(x) = \frac{q(x)}{e(x)} = \frac{x^3+6}{x-1} = x^2 + x + 1$. Bob notices

that the first character was corrupted (since $e_1 = 1$), so now that he has $p(x)$, he just computes $p(1) = 3 =$ "d" and obtains the original, uncorrupted message "dag".

Exercise. Verify the derivation of the above system of equations from the equalities $q(i) = r_i e(i)$ for $i = 1,2,3,4,5$. Also, solve the equations and check that your solution agrees with the one above. Remember that all the arithmetic should be done mod 7.

Exercise. Redo the example above for the case where the second character is corrupted from a "0" to a "5", and all other characters are uncorrupted.

3.3 Finer Points

Two points need further discussion. How do we know that the $n+2k$ equations are *consistent*? What if they have no solution? This is simple. The equations must be consistent since $q(x) = p(x)e(x)$ together with the actual error locator polynomial $e(x)$ gives a solution.

A more interesting question is this: how do we know that the $n+2k$ equations are *independent*, i.e., how do we know that there aren't other spurious solutions in addition to the real solution that we are looking for? Put more mathematically, suppose that the solution we construct is $q'(x), e'(x)$; how do we know that this solution satisfies the property that $e'(x)$ divides $q'(x)$ and that $\frac{q'(x)}{e'(x)} = \frac{q(x)}{e(x)} = p(x)$?

To see that this is true, we note first that, based on our method for calculating $q'(x), e'(x)$, we know that $q'(i) = r_i e'(i)$ for $1 \le i \le n+2k$; and of course we also have, by definition, $q(i) = r_i e(i)$ for the same values of i. Multiplying the first of these equations by $e(i)$ and the second by $e'(i)$, we get

$$q'(i)e(i) = q(i)e'(i) \qquad \text{for } 1 \le i \le n+2k, \tag{2}$$

since both sides are equal to $r_i e(i)e'(i)$. Equation (2) tells us that the two polynomials $q(x)e'(x)$ and $q'(x)e(x)$ are equal at $n+2k$ points. But these two polynomials both have degree $n+2k-1$, so they are completely determined by their values at $n+2k$ points. Therefore, since they agree at $n+2k$ points, they must be the same polynomial, i.e., $q(x)e'(x) = q'(x)e(x)$ for all x.[2] Now we may divide through by the polynomial $e(x)e'(x)$ (which by construction is not the zero polynomial) to obtain $\frac{q'(x)}{e'(x)} = \frac{q(x)}{e(x)} = p(x)$, which is what we wanted. Hence we can be sure that any solution we find is correct.

Exercise. Is the solution $q'(x), e'(x)$ always unique? The above analysis tells us that the ratio must always satisfy $\frac{q'(x)}{e'(x)} = \frac{q(x)}{e(x)} = p(x)$, but does not guarantee that $q'(x) = q(x)$ and $e'(x) = e(x)$. Hint: What happens in our example above when in fact none of the characters is corrupted (although Alice still assumes that $k = 1$ character may be corrupted)? Try writing out and solving the equations in this case.

---

[2] Note that this is a much stronger statement than equation (2). Equation (2) says that the *values* of the polynomials agree at certain points i; this statement says that the two polynomials are equal everywhere, i.e., they are the same polynomial.