CS 70 Discrete Mathematics and Probability Theory
Fall 2020

Dis 05B

1 Berlekamp-Welch Warm Up

Let $P(i)$, a polynomial applied to the input i, be the original encoded polynomial before sent, and let r_i be the received info for the input i which may or may not be corrupted.

(a) When does $r_i = P(i)$? When does r_i not equal $P(i)$?

(b) If you want to send a length-n message, what should the degree of $P(x)$ be? Why?

(c) If there are at most k erasure errors, how many packets should you send? If there are at most k general errors, how many packets should you send? (We will see the reason for this later.) Now we will only consider general errors.

(d) What do the roots of the error polynomial $E(x)$ represent? Does the receiver know the roots of $E(x)$? If there are at most k errors, what is the maximum degree of $E(x)$? Using the information about the degree of $P(x)$ and $E(x)$, what is the degree of $Q(x) = P(x)E(x)$?

(e) Why is the equation $Q(i) = P(i)E(i) = r_iE(i)$ always true? (Consider what happens when $P(i) = r_i$, and what happens when $P(i)$ does not equal r_i.)

(f) In the polynomials $Q(x)$ and $E(x)$, how many total unknown coefficients are there? (These are the variables you must solve for. Think about the degree of the polynomials.) When you receive packets, how many equations do you have? Do you have enough equations to solve for all of the unknowns? (Think about the answer to the earlier question - does it make sense now why we send as many packets as we do?)

(g) If you have $Q(x)$ and $E(x)$, how does one recover $P(x)$? If you know $P(x)$, how can you recover the original message?

Solution:

(a) $r_i = P(i)$ when the received packet is correct. r_i does not equal $P(i)$ the received packet is corrupted.

(b) P has degree $n-1$ since n points would determine a degree $n-1$ polynomial.

(c) We send $n+k$ packets when we have k erasures and $n+2k$ packets for k general errors.

(d) The roots of error polynomial $E(x)$ represent the locations of corrupted packets. The receiver does not know the roots of $E(x)$. $E(x)$ is a polynomial that the receiver needs to compute in order to obtain $P(x)$. If there are at most k errors, then the maximum degree of E(x) is k. The

maximum degree of Q is $(n-1)+(k) = n+k-1$ since the degree of P is $n-1$ and the degree of E is at most k.

(e) If there is no error at point i, $P(i) = r_i$ and then multiplying each side by E(i) gives $P(i)E(i) = r_iE(i)$. If there is an error at point i, then $E(i) = 0$, which means $P(i)E(i) = r_iE(i) = 0$.

(f) The maximum degree of Q(x) is $n+k-1$, so the number of unknowns is $n+k$. The maximum degree of E(x) is k, which would mean there would be $k+1$ unknowns. However, we know that the coefficient of x^k is 1 in E(x), so the number of unknowns is k.

The total number of unknowns is $(n+k)+(k) = n+2k$

There are $n+2k$ equations, which is enough to solve for $n+2k$ unknowns.

(g) We can compute P(x) using the equation: $P(x) = Q(x)/E(x)$. To recover the message, we compute $P(i)$ for $1 \leq i \leq n$.

2 Berlekamp-Welch Algorithm

In this question we will send the message $(m_0, m_1, m_2) = (1, 1, 4)$ of length $n = 3$. We will use an error-correcting code for $k = 1$ general error, doing arithmetic over GF(5).

(a) Construct a polynomial $P(x) \pmod 5$ of degree at most 2, so that

$$P(0) = 1, \qquad P(1) = 1, \qquad P(2) = 4.$$

What is the message $(c_0, c_1, c_2, c_3, c_4)$ that is sent?

(b) Suppose the message is corrupted by changing c_0 to 0. Set up the system of linear equations in the Berlekamp-Welch algorithm to find $Q(x)$ and $E(x)$.

(c) Assume that after solving the equations in part (b) we get $Q(x) = 4x^3 + x^2 + x$ and $E(x) = x$. Show how to recover the original message from Q and E.

Solution:

(a) We use Lagrange interpolation to construct the unique quadratic polynomial $P(x)$ such that $P(0) = m_0 = 1, P(1) = m_1 = 1, P(2) = m_2 = 4$. Doing all arithmetic over GF(5), so that i.e.

$2^{-1} = 3 \pmod 5$,

$$\begin{aligned}
\Delta_0(x) &= \frac{(x-1)(x-2)}{(0-1)(0-2)} = \frac{x^2-3x+2}{2} \equiv 3(x^2-3x+2) \pmod 5 \\
\Delta_1(x) &= \frac{(x-0)(x-2)}{(1-0)(1-2)} = \frac{x^2-2x}{-1} \equiv 4(x^2-2x) \pmod 5 \\
\Delta_2(x) &= \frac{(x-0)(x-1)}{(2-0)(2-1)} = \frac{x^2-x}{2} \equiv 3(x^2-x) \pmod 5 \\
P(x) &= m_0\Delta_0(x) + m_1\Delta_1(x) + m_2\Delta_2(x) \\
&= 1\Delta_0(x) + 1\Delta_1(x) + 4\Delta_2(x) \\
&\equiv 4x^2 + x + 1 \pmod 5
\end{aligned}$$

For the final message we need to add 2 redundant points of P. Since 3 and 4 are the only points in GF(5) that we have not used yet, we compute $P(3) = 0, P(4) = 4$, and so our message is $(1,1,4,0,4)$.

(b) The message received is $(c'_0, c'_1, c'_2, c'_3, c'_4) = (0,1,4,0,4)$. Let $R(x)$ be the function such $R(i) = c'_i$ for $0 \le i < 5$. Let $E(x) = x + b_0$ be the error-locator polynomial, and $Q(x) = P(x)E(x) = a_3x^3 + a_2x^2 + a_1x + a_0$. Since $Q(i) = P(i)E(i) = R(i)E(i)$ for $1 \le i < 5$, we have the following equalities $\pmod 5$

$$\begin{aligned}
Q(0) &= 0E(0) \\
Q(1) &= 1E(1) \\
Q(2) &= 4E(2) \\
Q(3) &= 0E(3) \\
Q(4) &= 4E(4)
\end{aligned}$$

which can be rewritten as a system of linear equations

$$\begin{array}{rcrcrcrcrcl}
 & & & & & & a_0 & & & = & 0 \\
a_3 & + & a_2 & + & a_1 & + & a_0 & - & b_0 & = & 1 \\
8a_3 & + & 4a_2 & + & 2a_1 & + & a_0 & - & 4b_0 & = & 8 \\
27a_3 & + & 9a_2 & + & 3a_1 & + & a_0 & & & = & 0 \\
64a_3 & + & 16a_2 & + & 4a_1 & + & a_0 & - & 4b_0 & = & 1
\end{array}.$$

(c) From the solution, we know

$$\begin{aligned}
Q(x) &= 4x^3 + x^2 + x, \\
E(x) &= x + b_0 = x.
\end{aligned}$$

Since $Q(x) = P(x)E(x)$, the recipient can compute $P(x) = Q(x)/E(x) = 4x^2 + x + 1$, the polynomial $P(x)$ from part (a) used by the sender. The error locating polynomial $E(x)$ is degree one, so there is only one error, and as $E(x) = x = x - 0$, the corrupted bit was the first one. To

correct this error we evaluate $P(0) = 1$ and combine this with the two uncorrupted bits m_1, m_2, to get the original message

$$(m_0, m_1, m_2) = (1, 1, 4).$$

Note: initially there was a typo in this part of the problem, asking that we assume $Q(x) = 2x^3 + 2x^2$. From this assumption, we deduce that $P(x) = Q(x)/E(x) = 2x^2 + 2x$. The intended message would then be

$$(m_0, m_1, m_2) = (P(1), P(1), P(2)) = (0, 4, 2)$$

3 Green Eggs and Hamming

The *Hamming distance* between two length-n bit strings b_1 and b_2 is defined as the minimum number of bits in b_1 you need to flip in order to get b_2. For example, the Hamming distance between 101 and 001 is 1 (since you can just flip the first bit), while the Hamming distance between 111 and 000 is 3 (since you need to flip all three bits).

(a) Sam-I-Am has given you a list of n situations, and wants to know in which of them you would like green eggs and ham. You are planning on sending him your responses encoded in a length n bit string (where a 1 in position i says you would like green eggs and ham in situation i, while a 0 says you would not), but the channel you're sending your answers over is noisy and sometimes corrupts a bit. Sam-I-Am proposes the following solution: you send a length $n+1$ bit string, where the $(n+1)$st bit is the XOR of all the previous n bits (this extra bit is called the parity bit). If you use this strategy, what is the minimum Hamming distance between any two valid bit strings you might send? Why does this allow Sam-I-Am to detect an error? Can he correct the error as well?

(b) If the channel you are sending over becomes more noisy and corrupts two of your bits, can Sam-I-Am still detect the error? Why or why not?

(c) If you know your channel might corrupt up to k bits, what Hamming distance do you need between valid bit strings in order to be sure that Sam-I-Am can detect when there has been a corruption? Prove as well that that your answer is tight—that is, show that if you used a smaller Hamming distance, Sam-I-Am might not be able to detect when there was an error.

(d) Finally, if you want to *correct* up to k corrupted bits, what Hamming distance do you need between valid bit strings? Prove that your condition is sufficient.

Solution:

(a) The minimum Hamming distance is 2. In order to prove this, we need to show both that there exits two valid strings we could send that have a Hamming distance of 2 from eachother, and that there are no valid strings we could send that are at Hamming distance 1. The former is easy: if we do not like green eggs and ham in any situation, our answer to Sam-I-Am would

just be $00\ldots0$. Since the XOR of n zeros is 0, our $(n+1)$st digit would also be a zero, so we would send a message containing $n+1$ zeros. In addition, if we only like green eggs and ham in the nth situation, but not any of the previous ones, our answer to Sam-I-Am would be $00\ldots01$, meaning that our message would be $n-1$ zeros followed by two ones (since the XOR of $0\cdots010\cdots0$ is one). These two bit strings are at Hamming distance 2 from one another, which is what we wanted to show.

To show that there is no pair of valid strings at Hamming distance 1, there are two cases to consider: either the bit that is different between them is in the first n, or it is the last (parity) bit. In the former case, flipping one of the n bits in our answer also flips the XOR of all our bits, meaning that the parity bit would have to change. However, both strings have to have the same parity bit, so one of the strings must have an incorrect parity bit and thus be invalid. In the second case, we know that the two strings share the same first n bits, but have different parity bits. However, the parity bit is uniquely determined by the first n bits, so one of the two strings must have an incorrect parity bit. Thus, in either case, one of the two strings must be invalid, so we can conclude that now two valid strings can have Hamming distance 2.

Finally, why is this relevant to error detecting? Let's call the list of possible strings you could send to Sam-I-Am your *codebook*. If no two strings in your codebook are within Hamming distance 2 of one another, then upon receiving a string with at most one bit flipped, Sam-I-Am can detect the error: if a bit was flipped, the new string is at Hamming distance 1 from the intended string, and thus is no longer in the codebook.

(b) Sam-I-Am cannot necessarily still detect the error. Because two valid strings can be a Hamming distance of two from eachother, it is possible that when the channel flips two bits, we end up at another valid string. As an example, say $n = 3$, Sam-I-Am received the string 1100. This message could have either resulted from us sending 1100 and not having any bits flipped or from sending 0000 and having the first two bits flipped. Thus, Sam-I-Am has no idea whether or not an error occurred.

(c) You need a Hamming distance of $k+1$ between valid strings. If no valid strings are closer than $k+1$, then there is no way to get from one valid code word to another by corrupting at most k bits. Thus, if Sam-I-Am gets a valid string, he can conclude that your message got through without error; otherwise, he has to ask you to send it again.

We also have to prove that this is tight, meaning that if our minimum Hamming distance was k or smaller, Sam-I-Am might not be able to detect an error. If there existed two valid strings b_1 and b_2 at a Hamming distance of k or less from each other and Sam-I-Am received b_1, he wouldn't know if we had sent b_1 and it had gotten through uncorrupted, or if we had tried to send b_2 and the proper bits had gotten flipped to make it look like b_1. Thus, if the minimum Hamming distance was k or less, Sam-I-Am wouldn't necessarily be able to detect whether or not error(s) occurred.

(d) You need a minimum Hamming distance of $2k+1$. The idea is that, upon receiving your possibly corrupted message, Sam-I-Am finds the string in the codebook closest in Hamming distance to the one he received. To prove that this works, we need to know that if we start at a bit string b and flip $\ell \leq k$ different bits, the Hamming-nearest string in our codebook is still

b. Call the corrupted string b' and assume (looking for contradiction) that some string c in our codebook is closer to b' than b is to b'. This means c can be obtained from b' by flipping $m \leq \ell \leq k$ bits, and b' can be obtained from b by flipping $\ell \leq k$ bits. Concatenating these sequences of bit flips, we can get c from b by flipping no more than $m + \ell \leq 2k$ bits. But this means the Hamming distance between b and c is less than $2k+1$, and we assumed that the minimum such distance was $2k+1$.