Berlekamp-Massey Algorithm

Erin Casey

University of Minnesota
REU Summer 2000

Berlekamp-Massey Algorithm

Erin Casey

Introduction

The main purpose of this paper is to acquaint the reader with the Berlekamp-Massey algorithm, it's proof and some of it's applications. In these last six weeks, I have studied many different topics ranging from first year Calculus reviews to second and third year graduate school topics. I chose to study the Berlekamp-Massey Algorithm more closely because this experience has been my first introduction to coding and I think it is very interesting. I have particular interest the binary BCH codes because of the simplicity of their design. Since the Berlekamp-Massey Algorithm is one of the most widely used algorithms used to decode them, I felt that studying it would be something I would enjoy.

However, because of the time frame of the project I was limited to studying Berlekamp's original algorithm and the mathematical concepts used in the proof of the algorithm rather than the relatively more exciting variations and applications. I primarily used Berlekamp's own work as my source on the algorithm and it's capabilities. Unfortunately, I was unfamiliar with the many of the ideas used by Berlekamp in his algorithm and therefore I spent much of the six weeks trying to get acquainted with their basic concepts. I have included short summaries of a few of these areas in this paper to provide a short background for the basis of the algorithm. Therefore, since I spent much of my time studying background for the algorithm and attempting to understand the processes used in the algorithm, I was not able to look more closely at the algorithm when it is actually applied. Even so, I am coming out of this experience with a very clear understanding of the Berlekamp-Massey Algorithm, different ways to proof it and also different ways in which the algorithm can be applied.

The Point of the Algorithm

The main purpose of the Berlekamp-Massey Algorithm is to evaluate Binary BCH codes. Berlekamp published his algorithm in 1968 and it was followed shortly by Massey's publication of a variation on the algorithm in 1969. The algorithm is most widely used as a fast way to invert matrices with constant diagonals. It works over any field, but the finite fields that occur most in coding theory are the most often used. The algorithm is specifically helpful for decoding various algebraic codes. Berlekamp's publication of the algorithm uses a "key equation" to input a known number of coefficients of the generating function and then determine the remaining coefficients of the polynomial. This process is equivalent to finding the linear complexity of the system. What is useful about this algorithm is that one only needs a small portion of the encoded message to be able to decode it. The crucial step is to reformulate the problem in a way that avoids thinking about n by n matrices explicitly since the work and storage volume of such an operation is too great. This reformulation was done by Berlekamp and his key equation and again done by Massey and his variation of the algorithm.

The applications and implementation of this algorithm were advanced and extended by Massey who used the physical interpretation of a linear feedback shift register (LFSR) as a tool to better understand the algorithm. What the variation does is synthesize LFSR's that have a specified output sequence. This physical interpretation of LFSR's provides a physical explanation of the length of the encoded message needed to be able to decode it using the algorithm. The length of message needed is only twice the length of the LFSR used or $2n$. Now that we have a handle on what the algorithm is trying to do, we can see where it is useful.

Applications of the Algorithm

As stated above, the algorithm deals with the decoding of Binary BCH codes, Reed-Solomon codes included. Fortunately for Berlekamp, at the time of publication of his algorithm Reed-Solomon codes were widely used in a variety of different fields. His algorithm's more efficient way of decoding these type of codes was of immediate interest. The use of his algorithm as a decoding tool has ranged from the standard for NASA deep space communication to the decoding done in compact disc players. In addition, many variations and similar algorithms are also being used in similar decoding procedures.

The topics of Linear Feedback Shift Registers, Cyclotomic Polynomials, and Primitive Roots will be briefly discussed to provide a basis for the derivation of the key equation and for the proof of the algorithm.

Linear Feedback Shift Registers

Linear Feedback Shift Registers are used in cryptography as tools to make ciphers more efficient. The terminology used to describe them is very appropriate when one evaluates what a feedback shift register actually does. In very basic terms, linear feedback shift registers are tools used in the encoding and decoding of a sequence by the use of a simple linear formula.

To create a linear feedback shift register, we first fix a size N, this N is the number of coefficients chosen and also the size of the initial seed. Also choose a modulus m (usually m=2). Then choose coefficients $c = (c_0, c_1, ..., c_{N-1})$ and also choose the initial state or seed $s = (s_0, s_1, ..., s_{N-1})$. From these simple definitions, we can recursively define s_{n+1} when $n + 1 \geq N$.

$$s_{n+1} = c_0 s_n + c_1 s_{n-1} + c_2 s_{n-2} + ... + c_{N-1} s_0 \quad \text{reduced modulo } m$$

This recursive definition that can be used to define the entire key stream can be written very easily in terms of matrices. As an example, let us choose N=5 and m=2. So our coefficients are $c = (c_0, c_1, c_2, c_3, c_4)$ and we can then make the matrix

$$C = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 & c_4 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

and from this matrix, the recursion relation can be written as

$$\begin{pmatrix} s_{n+1} \\ s_n \\ s_{n-1} \\ s_{n-2} \\ s_{n-3} \end{pmatrix} = C * \begin{pmatrix} s_n \\ s_{n-1} \\ s_{n-2} \\ s_{n-3} \\ s_{n-4} \end{pmatrix} \quad \text{(all modulo } m\text{=2)}$$

From this equation, one can see that different choices for $c = (c_0, c_1, c_2, c_3, c_4)$ and for $s = (s_0, s_1, s_2, s_3, s_4)$ different key-streams will be generated and the key-streams will have different *periods*.

(Example adapted from Garrett, Cryptography).

The Berlekamp-Massey Algorithm then solves for all $c = (c_0, c_1, ..., c_{N-1})$ using only $2n$ terms where n =the length of the period of the LFSR associated with a particular key-stream. Before we can give a proof of the algorithm or even attempt to understand it, we need to understand how Berlekamp arrived at his "key equation". Once we have established this, we can move on to how to solve the key equation.

Before starting the derivation, I am going to give a brief section on cyclotomic polynomials. This is a topic I was unfamiliar with prior to this project and therefore my statement of it here is partly to help familiarize the reader with the concept in order to understand the proof of the algorithm given and partly as a record of my studies.

Cyclotomic Polynomials and Primitive Roots

We can start by finding all complex zeros of $x^n - 1$. Let $\omega = \cos(\frac{2\pi}{n}) + i \sin(\frac{2\pi}{n})$. It follows from DeMoivre's Theorem which states

For every positive integer n, and every real number θ, $(\cos\theta + i\sin\theta)^n = \cos n\theta + i \sin n\theta$

that $\omega^n = 1$ and $\omega^k \neq 1$ for $1 \leq k < n$. Therefore, each of $1, \omega, \omega^2, ..., \omega^{n-1}$ is a zero of $s^n = 1$ and by a corollary to the division algorithm for polynomials which states

A polynomial of degree n over a field has at most n zeros counting multiplicity

that there are no other zeros. The complex number ω is called a *primitive n^{th} root of unity.* In order to use this notion of primitive roots to show how the cyclotomic polynomials are computed, it will be helpful to have a more formal definition of primitive roots. Primitive roots are similar to generators of cyclic groups. Consider the ω as defined above, it is a generator of a cyclic group of order n under the operation of multiplication. This cyclic group can be denoted as $< \omega >$ and from the following theorem about the generators of cyclic groups, we know that the generators of $< \omega >$ are the elements of the form ω^k where $1 \leq k \leq n$ and $\gcd(n,k) = 1$. These generators are called the *primitive n^{th} roots of unity.*

Theorem: Generators of Cyclic Groups
Let $G =< a >$ be a cyclic group of order n. Then $G =< a^k >$ if and only if $gcd(n,k) = 1$.

(Theorem adapted from Gallian).

Now that we have a better grasp of what a primitive root is, we can move on to cyclotomic polynomials and why they are important. Recalling the Euler function $\phi(n)$ as the notation for the number of positive integers less than or equal to n and relatively prime to n. Therefore, for every positive integer n there are exactly $\phi(n)$ primitive n^{th} roots of unity. The polynomials whose zeros fall on the $\phi(n)$ primitive n^{th} roots of unity are called cyclotomic polynomials.

Definition: Cyclotomic Polynomials
For any positive integer n, let $\omega_1, \omega_2, ..., \omega_{\phi(n)}$ denote the primitive n^{th} roots of unity. The n^{th} cyclotomic polynomial over Q is the polynomial $\Phi_n(x) = (x - \omega_1)(x - \omega_2) * * * (x - \omega_{\phi(n)})$.

Note: $\Phi_n(x)$ is monic and has degree $\phi(n)$ where $\phi(n)$ is Euler's funcion.

Rather than using this definition of $\Phi_n(x)$ to compute it, it is easier in practice to use the following formula and then make recursive use of the result.

For every positive integer n, $x^n - 1 = \prod_{d|n} \Phi_d(x)$ where the product runs over all positive divisors d of n.

Cyclotomic polynomials and primitive roots are two very widely used and basic concepts in abstract algebra and a basic knowledge of both is necessary to understand much of higher mathematics. Now that we are more familiar with primitive roots and cyclotomic polynomials we can move onto the key equation and how Berlekamp arrived at it as the basis for his algorithm.

Derivation of Key Equation

Adapted from Berlekamp's Algebraic Coding Theory, Revised edition, 1984.

In order to construct a binary BCH code capable of correcting t or fewer errors, first select an irreducible binary factor of the cyclotomic polynomial Φ_n over Q. The degree of the irreducible binary factor is the multiplicative order of 2 mod n which is denoted by m. Since we know the degree of the irreducible binary factor is m, and that $2^m \mod n = 1$ in the finite field $GF(2^m)$ we can from here, let $\alpha \, \epsilon \, GF(2^m)$ be a root of the irreducible binary polynomial. Once we have established a root of our irreducible binary polynomial, we can construct our binary BCH code.

Determination of the key equation for decoding binary BCH codes

Suppose the encoder transmits a binary BCH codeword

$$C(x) = \sum_{i=0}^{n-1} C_i x^i$$

In the transmission of the word it is possible that the channel noise could cause additive errors in the coefficients of the binary polynomial. These errors can be represented by the following equation

$$E(x) = \sum_{i=0}^{n-1} E_i x^i$$

These errors then change the received code word to $R(x) = C(x) + E(x)$ or

$$R(x) = \sum_{i=0}^{n-1} R_i x^i = \sum_{i=0}^{n-1} C_i x^i + \sum_{i=0}^{n-1} E_i x^i \qquad \textbf{Equation A}$$

For $j = 1, 2, ..., 2t$ the codeword is a multiple of the minimal polynomial of α^j. Let $M^{(j)}(x)$, be the minimal polynomial of α^j given that $(1 \leq j \leq 2t)$. So then the received word can be written as

$$R(\alpha^j) = 0 + \sum_{i=0}^{n-1} E_i \alpha^{ji} = \sum_{k=1}^{e} X_k^j = S_j \qquad \textbf{Equation B}$$

Here, t is the maximum number of errors that the code can correct, e is the number of errors that have occurred. The Galois field error location $X_1, X_2, ..., X_e$ denote positions where $E_i = 1$ (where an error has occurred). Now, if the received word $R(x)$ from Equation A is divided by $M^{(j)}(x)$ then $S_j = R(\alpha^j)$ may be computed from the remainder $r^{(j)}(\alpha^j)$ This can be done using a parity check matrix.

As an example: if the received word $R(x)$ is represented as above, and S_1 gives the sum of the error locations, and S_2 gives the sum of the squares of the error locations, then $S_1 = R(\alpha)$ and $S_2 = R(\alpha^2)$ So to compute $S - 1$, we divide $R(x)$ by $M^{(1)}(x)$and obtain remainder $r^{(1)}(x)$ and therefore $S_1 = r^{(1)}(\alpha)$. Similarly, to compute S_2, we divide $R(x)$ by $M^{(2)}(x)$, and obtain the remainder $r^{(2)}(x)$. Then $S_2 = r^{(2)}(\alpha^2)$. A further discussion of parity check matrices is located in *Garrett, Error Correcting Codes.*

After calculating $S_1, S_2, ..., S_{2t}$, the question is to find $X_1, X_2, ..., X_e$ from the equations

$$\sum_{i=1}^{e} X_i^j = S_j \text{ where } j = 1, 2, ..., 2t \qquad \textbf{Equation C}$$

These equations have many solutions, each corresponding to a different error pattern in the same coset of the additive group of codewords. For obvious reasons, we want to find the solution with the minimal value of e. To do this I must introduce Berlekamp's error-locator polynomial

$$\sigma(z) = \prod_{i=1}^{e} (1 - X_i z) = 1 + \sum_{j=1}^{e} \sigma_j z_j \qquad \textbf{Equation D}$$

Berlekamp's method for finding this equation is too extensive for this paper but can be found in his book, Algebraic Coding Theory, Revised Edition, 1984 Ch.1.

Once the decoder has found the error-locator polynomial $\sigma(z)$, the reciprocal roots *(multiplicative inverses)* can be found by completing a Chien search. Once this is completed, the errors can be corrected. The most difficult part of the decoding is finding the $\sigma(z)$'s from the S's. To obtain a relationship between the $\sigma(z)$'s and the S's, we need to bring in the generating function

$$S(z) = \sum_{j=1}^{\infty} S_j z^j = \sum_{j=1}^{\infty} \sum_{i=1}^{e} X_i^j z^j = \sum_{i=1}^{e} \frac{X_i z}{1 - X_i z} \qquad \textbf{Equation E}$$

We can now eliminate the fractions in the last sum by multiplying the Equation E through by $\sigma(z)$. This then yields the following equation.

$$S(z)\sigma(z) = \sum_{i=1}^{e} \frac{X_i z}{1 - X_i z} \prod_{j=1}^{e} (1 - X_j z) = \sum_{i=1}^{e} X_i z \prod_{j \neq i} (1 - X_j z)$$

From here adding $\sigma(z)$ to both sides and defining the polynomial $\omega(z) = \sum_{i=1}^{e} \omega_k z^k$ by the following equation.

$$\omega(z) = \sigma(z) + \sum_{i=1}^{e} X_i z \prod_{j \neq i} (1 - X_j z) \qquad \textbf{Equation F}$$

Then we have the equation $[1 + S(z)]\sigma(z) = \omega(z)$ Generally, the decoder only knows a limited number of the coefficients of the powers of z in $S(z)$. (Namely only the first $2t$ powers). So the unknown terms are $S_{2t+1}, S_{2t+2}, S_{2t+3}, ...$ So although $S(z)$ is unknown, because of modular arithmetic, the decoder does know $S(z)$ mod z^{2t+1} This fact, reveals Berlekamp's Key Equation.

KEY EQUATION $[1+S(z)]\sigma(z) \equiv \omega(z) \bmod z^{2t+1}$ **Equation G**

Here S(z) is the known generating function and its coefficients are the known inputs to the problem. σ(z) and ω(z) are two unknown polynomials with degree $\leq e$ (e is the number of errors that actually occurred). The coefficients of these polynomials will become the algorithm's outputs. The challenge Berlekamp faced was to find a faster algorithm to find the coefficients of these polynomials.

Massey's Interpretation of the Key Equation

Under Massey's interpretation S(x) is the generating function for the terms along the successive diagonals of the traditional matrix. In this "physical interpretation" the key equation may be written as

$$S_k + \sum_{i=1}^{k-1} \sigma_i S_{k-i} + \sigma_k = \omega_k$$

or similarly, it could be written as

$$S_k = \omega_k - \sum_{i=1}^{k-1} \sigma_i S_{k-i} - \sigma_k$$

This equation gives the k^{th} output of a feedback shift register, as discussed earlier. These feedback shift registers are wired in accordance to the coefficients of $\sigma(z)$ and initially loaded with the coefficients of $\omega(z)$. Therefore, the key equation is actually a mathematical problem in the field of feedback-shift-register synthesis. We are given the output sequence $1 + S(z)$ and want to determine the connections $\sigma(z)$ and the initial conditions $\omega(z)$ of the shortest feedback shift register with our given output sequence.

Note: For our decoding purpose, the polynomial $\sigma(z)$ is our only interest but there are other applications where $\omega(z)$ is important.

Proof of Berlekamp-Massey Algorithm

Adapted from Berlekamp's Algebraic Coding Theory, Revised Edition, 1984

Solving the Key Equation

Now that we have derived the key equation, we need to develop an algorithm to solve it over any field.

KEY EQUATION $[1+S(z)]\sigma(z) \equiv \omega(z) \bmod z^{2t+1}$ **Equation G**

To solve the key equation for the polynomials, $\sigma(z)$ and $\omega(z)$ given $S(z)$ mod z^{2t+1} we must break the problem up into manageable pieces. Consider the sequence of equations

$$(1+S)\sigma^{(k)} \equiv \omega^{(k)} \bmod z^{k+1} \qquad \textbf{Equation 1}$$

So for each $k = 0, 1, 2, ..., 2t$ find polynomials

$$\sigma^{(k)} = \sum_i \sigma_i^{(k)} z^i \qquad \text{and} \qquad \omega^{(k)} = \sum_i \omega^{(k)} z^i$$

which solve Equation 1.

As I stated before, these equations for $\sigma^{(k)}$ and $\omega^{(k)}$ may have many solutions. Since the degree of σ is the number of errors, a good solution for decoding is one where the degrees of σ and ω are relatively small. If our solutions, $\sigma^{(k)}$ and $\omega^{(k)}$ work for the above congruence given in Equation 1, we would like to say that they also work for the related statement.

$$(1+S)\sigma^{(k)} \equiv \omega^{(k)} \bmod z^{k+2} \qquad ?$$

Unfortunately, we do not know this for sure, but by the properties of modular arithmetic we do know that by adding a multiple of z^{k+1} we can adapt the it to

$$(1+S)\sigma^{(k)} \equiv \omega^{(k)} + \Delta_1^{(k)} z^{k+1} \bmod z^{k+2} \qquad \textbf{Equation 2}$$

where $\Delta_1^{(k)}$ is the coefficient of z^{k+1} in the product $(1+S)\sigma^{(k)}$. If $\Delta_1^{(k)} = 0$, then we can take the statements $\sigma^{(k+1)} = \sigma^{(k)}$ and $\omega^{(k+1)} = \omega^{(k)}$ to be true. If $\Delta_1^{(k)} \neq 0$, then we must find alternate definitions for the successive σ's and ω's, specifically $\sigma^{(k+1)}$ and $\omega^{(k+1)}$. Consider auxiliary polynomials $\tau^{(k)}$ and $\gamma^{(k)}$ which we can choose to then solve the auxiliary equation

$$(1+S)\tau^{(k)} \equiv \gamma^{(k)} + z^k \bmod z^{k+1} \qquad \textbf{Equation 3}$$

Obviously, since the degrees of functions $\tau^{(k)}$ and $\gamma^{(k)}$ are still related to the number of errors that actually occurred, the degrees should be relatively small for both functions. Now, using these auxiliary polynomials, we can define the successive σ's and ω's the following way.

$$\sigma^{(k+1)} = \sigma^{(k)} - \Delta_1^{(k)} z\tau^{(k)} \qquad \text{and} \qquad \omega^{(k+1)} = \omega^{(k)} - \Delta_1^{(k)} z\gamma^{(k)} \qquad \textbf{Equations 4 and 5}$$

From these definitions, it can now be seen that $\sigma^{(k+1)}$ and $\omega^{(k+1)}$ satisfy the equation

$$(1+S)\sigma^{(k+1)} \equiv \omega^{(k+1)} \bmod z^{(k+1)+1}$$

And so if $\sigma^{(k)}$ and $\omega^{(k)}$ satisfy Equation 1 and $\tau^{(k)}$ and $\gamma^{(k)}$ satisfy Equation 3, than there are two choices for the definitions of $\tau^{(k+1)}$ and $\gamma^{(k+1)}$. *Either*

$$\tau^{(k+1)} = z\tau^{(k)} \qquad \text{and} \qquad \gamma^{(k+1)} = z\gamma^{(k)} \qquad \textbf{Equation 6}$$

or

$$\tau^{(k+1)} = \frac{\sigma^{(k)}}{\Delta_1^{(k)}} \qquad \text{and} \qquad \gamma^{(k+1)} = \frac{\omega^{(k)}}{\Delta_1^{(k)}} \qquad \textbf{Equation 7}$$

Either of these choices will satisfy the equation

$$(1+S)\tau^{(k+1)} \equiv \gamma^{(k=1)} + z^{k+1} \bmod z^{k+2}$$

if $\sigma^{(k)}$ and $\omega^{(k)}$satisfy Equation 1 and $\tau^{(k)}$ and $\gamma^{(k)}$ satisfy Equation 3. If $\Delta_1^{(k)} = 0$, then the second definition is meaningless, and we must use the first definition. However if $\Delta_1^{(k)} \neq 0$ then our choice of definition depends on our need to minimize the degree of $\tau^{(k+1)}$ and $\gamma^{(k+1)}$. The equations for the degrees of $\sigma^{(k+1)}$, $\tau^{(k+1)}$, $\omega^{(k+1)}$, and $\gamma^{(k+1)}$ are given by Berlekamp and restated here.

$$\deg \sigma^{(k+1)} = \left\{ \begin{array}{ll} \deg \sigma^{(k)} & \text{if } \Delta_1^{(k)} = 0 \text{ or if } \deg \sigma^{(k)} > 1 + \deg \tau^{(k)} \\ 1 + \deg \tau^{(k)} & \text{if } \Delta_1^{(k)} \neq 0 \text{ and if } \deg \tau^{(k)} > \deg \sigma^{(k)} - 1 \end{array} \right\} \qquad \textbf{Equation 8}$$

$$\deg \sigma^{(k+1)} \leq \text{ either of above if } \Delta_1^{(k)} \neq 0 \text{ and if } \deg \sigma^{(k)} = 1 + \deg \tau^{(k)}$$

$$\deg \tau^{(k+1)} = \left\{ \begin{array}{ll} 1 + \deg \tau^{(k)} & \text{if we use Equation 6} \\ \deg \sigma^{(k)} & \text{if we use Equation 7} \end{array} \right\} \qquad \textbf{Equation 9}$$

$$\deg \omega^{(k+1)} = \left\{ \begin{array}{ll} \deg \omega^{(k)} & \text{if } \Delta_1^{(k)} = 0 \text{ or if } \deg \omega^{(k)} > 1 + \deg \gamma^{(k)} \\ 1 + \deg \gamma^{(k)} & \text{if } \Delta_1^{(k)} \neq 0 \text{ and if } \deg \gamma^{(k)} > \deg \omega^{(k)} - 1 \end{array} \right\} \qquad \textbf{Equation 10}$$

$$\deg \omega^{(k+1)} \leq \text{ either of above if } \Delta_1^{(k)} \neq 0 \text{ and if } \deg \omega^{(k)} = 1 + \deg \gamma^{(k)}$$

$$\deg \gamma^{(k+1)} = \left\{ \begin{array}{ll} 1 + \deg \gamma^{(k)} & \text{if we use Equation 6} \\ \deg \omega^{(k)} & \text{if we use Equation 7} \end{array} \right\} \qquad \textbf{Equation 11}$$

These equations are not as precise as we need them to be however.

An example of this is the degree of $\sigma^{(k+1)}$ is subject to an unavoidable decrease if $\deg \sigma^{(k)} = 1 + \deg \tau^{(k)}$ and the leading coefficients of $\sigma^{(k)}$ and $\Delta_1^{(k)}\tau^{(k)}$ are equal. In order to avoid such situations, we must base our choice of Equation 6 or 7 not on the actual degrees of $\sigma^{(k)}$, $\tau^{(k)}$, $\omega^{(k)}$, and $\gamma^{(k)}$ but instead on an upper bound $D(k)$ which is independent of such things to make sure accidental situations, like the one given above, will not arise. We will define the integral valued function $D(k)$ so that

$$\deg \sigma^{(k)} \leq D(k) \qquad \textbf{Equation 12}$$

$$\deg \tau^{(k)} \leq k - D(k) \qquad \textbf{Equation 13}$$

and then from Equation 8 for the degrees of $\sigma^{(k+1)}$ and Equations 12 and 13, we can give a recursive definition for $D(k)$

$$D(k+1) = \left\{ \begin{array}{ll} D(k) & \text{if } \Delta_1^{(k)} = 0 \text{ or if } D(k) \geq \frac{k+1}{2} \\ k + 1 - D(k) & \text{if } \Delta_1^{(k)} \neq 0 \text{ and } D(k) \leq \frac{k+1}{2} \end{array} \right\} \qquad \textbf{Equation 14}$$

It can be easily seen that if $\deg \sigma^{(k)} \leq D(k)$ and $\deg \tau^{(k)} \leq k - D(k)$, then $\deg \sigma^{(k+1)} \leq D(k+1)$. This similarly holds for ω and γ. However, to ensure that $\deg \tau^{(k+1)} \leq (k+1) - D(k+1)$ and that $\gamma^{(k+1)} \leq (k+1) - D(k+1)$, we must establish yet another criteria for choosing between equations 6 and 7.

$$\text{Use } \left\{ \begin{array}{ll} \text{Equation 6} & \text{if } \Delta_1^{(k)} = 0 \text{ or if } D(k) > \frac{k+1}{2} \\ \text{Equation 7} & \text{if } \Delta_1^{(k)} \neq 0 \text{ and } D(k) < \frac{k+1}{2} \end{array} \right\} \qquad \textbf{Rule 15}$$

If $\Delta_1^{(k)} \neq 0$ and $D(k) = \frac{k+1}{2}$, then either equation 6 or 7 will give us polynomials that satisfy the equations,

$$\deg \tau^{(k+1)} \leq k + 1 - D(k+1) \qquad \text{and} \qquad \deg \gamma^{(k+1)} \leq k + 1 - D(k+1)$$

If you are having trouble deciding between which equation 6 or 7 to use, don't decide right away and wait until you have examined all of the given criterion.

To review, the initial equations are

$$(1 + S)\sigma^{(0)} \equiv \omega^{(0)} \bmod z$$

$$(1 + S)\tau^{(0)} \equiv \gamma^{(0)} + 1 \bmod z$$

The easiest way to solve these equations is with the following initializations

$$\sigma^{(0)} = \tau(0) = \omega^{(0)} = 1 \qquad \text{and} \qquad \gamma^{(0)} = 0 \qquad \text{and} \qquad D(0) = 0 \qquad \textbf{Equation 16}$$

From these initalizaitions, we notice that $\deg \sigma^{(0)} = \deg \tau^{(0)} = \deg \omega^{(0)} = 0 = D(0)$ but that $\deg \gamma^{(0)} = -\infty < D(0)$. *(Since the degree of the product of several polynomials is the sum of their degrees, we must define* $\deg 0 = -\infty$*)* Thus, at least initially, we can do better than our restrictions

$$\deg \omega^{(k)} \leq D(k)$$

$$\deg \gamma^{(k)} \leq k - D(k)$$

In actuality, at least one of these equations must be satisfied for a strict inequality. To accomplish this we can introduce the Boolean function $B(k)$, where the initial value is $B(0) = 0$. [We will not worry about the actual Boolean function and how it works but rather just note that, in general, $B(k) = 0$ or $B(k) = 1$.] So once this function is defined a new relation can be established.

$$\deg \omega^{(k)} \leq D(k) - B(k) \qquad \textbf{Equation 17}$$

$$\deg \gamma^{(k)} \leq k - D(k) - [1 - B(k)] \qquad \textbf{Equation 18}$$

From here it can be seen that if the proper choice is made between Equations 6 and 7 in the case where $\Delta_1^{(k)} \neq 0$ and $D(k) = \frac{k+1}{2}$ and then if we define B(k) "carefully", then we can be sure that above two equations hold for all k. By referring back to Equations 10 and 11 we can see that the *proper choice* is

$$\text{Use} \begin{cases} \text{Equation 6} & \text{if } \Delta_1^{(k)} \neq 0, D(k) = \frac{k+1}{2}, \text{ and } B(k) = 0 \\ \text{Equation 7} & \text{if } \Delta_1^{(k)} \neq 0, D(k) = \frac{k+1}{2}, \text{ and } B(k) = 1 \end{cases} \qquad \textbf{Rule 19}$$

$$B(k+1) = \begin{cases} B(k) & \text{when using Equation 6} \\ 1 - B(k) & \text{when using Equation 7} \end{cases} \qquad \textbf{Equation 20}$$

Re-cap of Proof

Start from the initial conditions in Equation 16. Then proceed recursively. Start by Defining $\Delta_1^{(k)}$ from Equation 2, $\sigma^{(k+1)}$ from Equation 4, $\omega^{(k+1)}$ from Equation 5, and finally define $D(k+1)$ by Equation 14. From here according to Rules 15 and 19 we can define $\tau^{(k+1)}$ and $\gamma^{(k+1)}$ by Equations 6 or 7 and similarly define $B(k+1)$ by Equation 20. Each of these polynomials is therefore defined in this recursive manner so that they satisfy Equations 1, 3, 12, 13, 17 and 18.

Statement of the Berlekamp-Massey Algorithm

Copied from Berlekamp's Algebraic Coding Theory, Revised Edition, 1984 Ch. 7

AN ALGORITHM FOR SOLVING THE KEY EQUATION OVER ANY FIELD

Initially define $\sigma^{(0)} = 1, \tau^{(0)} = 1, \omega^{(0)} = 1, \gamma^{(0)} = 0, D(0) = 0, B(0) = 0$. Proceed recursively as follows. If S_{k+1} is unknown, stop; otherwise define $\Delta_1^{(k)}$ as the coefficient of z^{k+1} in the product $(1+S)\sigma^{(k)}$ and let

$$\sigma^{(k+1)} = \sigma^{(k)} - \Delta_1^{(k)} z\tau^{(k)}$$
$$\omega^{(k+1)} = \omega^{(k)} - \Delta_1^{(k)} z\gamma^{(k)}$$

If $\Delta_1^{(k)} = 0$, or if $D(k) > \frac{k+1}{2}$, or if $\Delta_1^{(k)} \neq 0$ and $D(k) = \frac{k+1}{2}$ and $B(k) = 0$, set

$$D(k+1) = D(k)$$
$$B(k+1) = B(k)$$
$$\tau^{(k+1)} = z\tau^{(k)}$$
$$\gamma^{(k+1)} = z\gamma^{(k)}$$

But if $\Delta_1^{(k)} \neq 0$ and either $D(k) < \frac{k+1}{2}$ or $D(k) = \frac{k+1}{2}$ and $B(k) = 1$, set

$$D(k+1) = k + 1 - D(k)$$
$$B(k+1) = 1 - B(k)$$
$$\tau^{(k+1)} = \frac{\sigma^{(k)}}{\Delta_1^{(k)}}$$
$$\gamma^{(k+1)} = \frac{\omega^{(k)}}{\Delta_1^{(k)}}$$

Conclusion

As you can see from the proof and expression of the algorithm, understanding it (much less implementing it) is a complex process. Through this project I have been able to make a complete effort toward understanding the Berlekamp-Massey Algorithm and Berlekamp's proof of it. I now feel prepared to look at implementations of the algorithm. I plan to further look into this subject on my own in my next year of study.

Bibliography

Berlekamp, Elwyn R. *Algebraic Coding Theory, Revised Edition.* Aegean Park Press, Laguna Hills, CA 1984.

Elwyn Berlekamp's Homepage ¡http://math.berkley.edu/ berlek/¿

Gallian, Joseph A. *Contemporary Abstract Algebra.* Houghton Mifflin Company, Boston, 1998.

Garrett, Paul. *Error Correcting Codes.* Notes 1999-2000.

Garrett, Paul. *Introduction to Cryptography.* Notes. 2000.

Pan, Victor. *"New Techniques for the Computation of Linear Recurrence Coefficients" Finite Fields and Their Applications.* Vol 6. 2000, p.93-118.

Riesel, Hans. *Prime Numbers and Computer Methods for Factorization.* Birkhauser, Boston, 1994.