Rose-Hulman Undergraduate Mathematics Journal

Volume 7
Issue 2

Article 7

Linear Feedback Shift Registers and Cyclic Codes in SAGE

Timothy Brock
United States Naval Academy, tbbrock@nps.edu

Linear Feedback Shift Registers in SAGE

Timothy Brian Brock

May 16, 2006

> ... random numbers should not be generated by a method chosen at random.
>
> - Donald Knuth, *The Art of Computer Programming.*

1 Introduction

Let $\mathbf{F} = \mathbf{F}_q$ denote a finite field having q elements. A **general feedback shift register** is a map $f : \mathbf{F}^d \to \mathbf{F}^d$ of the form

$$f(x_0, ..., x_{n-1}) = (x_1, x_2, ..., x_n),$$
$$x_n = C(x_0, ..., x_{n-1}),$$

where $C : \mathbf{F}^d \to \mathbf{F}$ is a given function. When C is of the form

$$C(x_0, ..., x_{n-1}) = a_0 x_0 + ... + a_{n-1} x_{n-1},$$

for some given constants $a_i \in \mathbf{F}$, the map is called a **linear feedback shift register (LFSR)**. It turns out such sequences are periodic.

The main focus of this paper will be to discuss Massey's algorithm [4] for deciphering a stream cipher given by a linear feedback shift register. This algorithm has been implemented (by the author) in the computer algebra system SAGE [9] using Python (www.python.org)[1].

Here, we shall work over any finite field, instead of only dealing with the binary case. However, this paper will begin by discussing some of the history of LFSRs in applications.

1.1 Background on Linear Feedback Shift Registers

A (non-)linear feedback shift register can be easily implemented in hardware or software and is used to create a pseudo-random sequence of numbers for many different applications. These applications include uses in consumer electronics, such as cellphones and digital cable [7]; multiple access and polling techniques; secure and privacy communications; error detecting and correcting codes; and

[1] SAGE is a free and open-source computer algebra system written primarily in Python. Please see section 4 below for more details.

cryptographic systems [1]. The fascinating notes of Körner [2] also mention cable TV scrambling and the use of periodic sequences for modeling the behaviour of the British and German WWII codes. (We refer to Sherman [8] for a rigorous mathematical interpretation of the Enigma cipher machine.)

In consumer electronics, a Linear Feedback Shift Register can be used as a counter [7]. When used in this manner, LFSRs are desirable because they perform the function with less resources and usually much faster than the conventional counters, such as binary counters or Gray Code counters. Though it goes against intuition, LFSRs can also be used to generate pseudo-noise which is used by such consumer electronics as cellphones and digital cable to increase the reliability of the signal [7]. LFSRs can also be used in spread spectrum systems [7]. A spread spectrum system utilizes the entire bandwidth a signal may use to send information by spreading the data frequency over many frequencies in the bandwidth. The next frequency to be utilized is determined by the LFSR sequence. Other more common applications of a LFSR is the use in white noise machines (such as the one shown below) and music synthesizers, where they are used to make the electrically-produced music sound more natural.

White noise machine (photo: public domain)

Another application of LFSRs is the creation of pseudo-random sequences that can be used in cryptography. The LFSR sequence is a pseudo-random number sequence that can be applied to a message as a cipher, as explained in Example 3 below. Throughout this paper, the cipher will be a sequence of binary terms that is added to the binary message to provide the encoded message, also known as the *ciphertext*. The cipher encodes the message so that only someone with the *key* knows the proper way to decode the message and is then able to read the message, anyone without the key receives the ciphertext and reads only nonsense. The key is a piece of information that allows a user to determine the specific cipher used in encrypting the message. In digital communication, the enciphering of a message with a LFSR sequence is the same as adding in pseudo-random noise. The proper recipient with a key removes the noise from the message, but a third party without the key interprets the message only as noise.

1.2 How to Create a LFSR Sequence

A *linear feedback shift register sequence* is a pseudo-random sequence of numbers that is often created in a hardware implementation of a linear feedback shift register. When a LFSR is implemented in hardware, a LFSR sequence is recursively generated by taking the output from the last "stage" of a given LFSR to compute the next "stage." An example of a LFSR implemented in hardware is included in Figure 1 (from Massey [4]). This LFSR is of length L, and each state cell's current state is used as the input to the mod 2 adder. This adder is implemented in hardware with an exclusive-or function. Since this is a shift register, each iteration of the register causes the state of each state cell to shift to the next cell (in this case, to the right). We use the output of the last state cell to provide the next term of the sequence after each iteration.

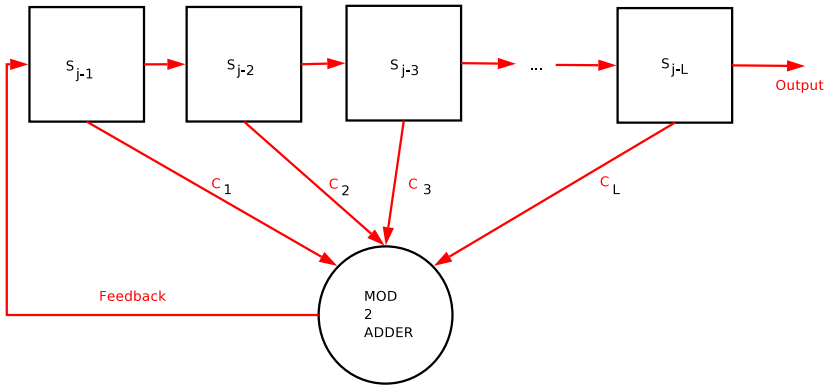

Figure 1. L bit Linear Feedback Shift Register

This hardware LFSR can be modeled mathematically to generate a LFSR sequence. In order to build this sequence, three pieces of information are needed. They are the (1) key, the (2) initial fill, and (3) an algorithm to obtain the next term of the sequence. In the hardware implementation, the connections between the state cells and the mod 2 adder determines how the outputs of the cells are used as inputs to the mod 2 adder. In the same way, the key determines how the previous terms of the LFSR sequence are used to compute the next term in the sequence. The key may be represented as a vector $\mathbf{c} = [c_1, c_2, ..., c_L]$, but is more often defined by a polynomial, known as the **connection polynomial**

$$C(x) = 1 + c_1 \cdot x + c_2 \cdot x^2 + ... + c_L \cdot x^L. \quad (1)$$

The coefficients c_i's can also be considered the key. In Figure 1, the coefficients describe which cells were used as inputs to the modulo 2 adder. The degree of the polynomial also describes how many cells (or bits) are needed to create the minimal linear feedback shift register that will generate the given LFSR sequence.

According to Massey [4], the initial fill is the list of initial values of the state cells, $s_0, s_1, s_2, ..., s_{L-1}$ the initial contents of the L stages of Figure 1 above. In

the binary case, the LFSR sequence is defined by the following recursion relation

$$s_j = \sum_{i=1}^{L} c_i \cdot s_{j-i} \mod 2, \tag{2}$$

for $j \geq L$.

Example 1 *If we are given the key as a vector* $\mathbf{c} = [1, 0, 0, 1]$ *and the initial fill as a vector* $\mathbf{s} = [1, 1, 0, 1]$ *in the finite field* $GF(2)$*, we can create the sequence* $1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, ...$ *Note that the first four terms of the sequence are the same as the terms given to us by the vector* $\mathbf{s}$ *(namely* $s_0 = 1, s_1 = 1, s_2 = 0, s_3 = 1$*). The next term* (s_4) *is found by using the recursion function to give*

$$\begin{aligned} s_4 = \textstyle\sum_{i=1}^{4} c_i \cdot s_{4-i} &= c_1 \cdot s_3 + c_2 \cdot s_2 + c_3 \cdot s_1 + c_4 \cdot s_0 \\ &= 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 = 0. \end{aligned}$$

We know that $L = 4$ *since the length of vectors* $\mathbf{c}$ *and* $\mathbf{s}$ *is 4. This sequence satisfies Golomb's three randomness conditions given in §2.2. Fortunately, this process can be easily automated and a function has been written for the computer algebra system SAGE which will quickly generate terms of a LFSR sequence of any length defined by the user. The inputs for this function are two vectors representing the key and the initial fill and an integer* $n > L$ *representing the desired number of terms in the output.*

Example 2 *More generally, let*

$$f(x) = a_0 + a_1 x + ... + a_n x^n + ...,$$

$$g(x) = b_0 + b_1 x + ... + b_n x^n + ...,$$

be given polynomials in $\mathbf{F}_3[x]$ *and let*

$$h(x) = \frac{f(x)}{g(x)} = c_0 + c_1 x + ... + c_n x^n + ... \ .$$

We can compute a recursion formula which allows us to rapidly compute the coefficients of $h(x)$ *(take* $f(x) = 1$*):*

$$c_n = \sum_{i=1}^{n} \frac{-b_i}{b_0} c_{n-i}.$$

The coefficients of $h(x)$ *can, under certain conditions on* $f(x)$ *and* $g(x)$*, be considered "random" from certain statistical points of view.*

For instance, if we consider a case other than binary and if

$$f(x) = 1, \qquad g(x) = 2 \cdot x^4 + x + 1,$$

then

$$h(x) = 1 + 2 \cdot x + x^2 + 2 \cdot x^3 + 2 \cdot x^4 + x^6 + x^7 + x^8 + ... \ .$$

The coefficients of h are

$$2,1,2,2,0,1,1,1,2,2,2,2,0,2,0,$$
$$2,1,1,2,0,1,0,2,1,0,0,2,2,1,2,\ldots .$$

The sequence of $0,1,2$'s is periodic with period $P = 3^4 - 1 = 80$. However, this sequence of period 80 can be "cracked" (i.e., a procedure to reproduce $g(x)$) by knowing only 8 terms! This is the result of Massey's algorithm [4], which is implemented in SAGE by the author, and described in detail below.

2 Linear Feedback Shift Registers in Cryptography

A stream cipher is a sequence of binary digits called a **cryptographic bitstream** [6]. The stream cipher is then added to the message to create a ciphertext (encryption) and can be added to the ciphertext to obtain the original message (decryption).

2.1 Four Tenets of Security

Imagine that Alice and Bob are sending messages back and forth to one another. If the content of these messages was not very secret and Alice and Bob did not care if anyone else read the message, they would not bother with any kind of encryption. Then, if an evil eavesdropper, say Eve, intercepts the message she can read it without any difficulty. If, on the other hand, Alice and Bob were exchanging information they wanted to keep secret, they would need to employ some kind of encryption system to encode their messages. Depending upon the type of encryption they employ, Alice and Bob would receive a certain level of security against the actions of third parties.

No matter what type of encryption Alice and Bob use, there will be several objectives that Alice and Bob want the encryption method to achieve. Out of a long list of objectives, there are four that form a framework upon which all the others are built [5], [2]. The four security objectives that would apply to this message include:

1. *Secrecy*: This objective ensures that the information is available only to those people who are authorized to have it.

2. *Integrity*: This ensures that no third party can make unauthorized alterations to the data.

3. *Non-repudiation*: This prevents the sender of information from denying that they sent that information. This also allows the receiver to prove to a third party that the information was sent by the sender [5].

4. *Authentication*: Authentication is related to identification. Two parties that are communicating with each other need to be able to identify one another and the receiver can be sure that the person sending the information is who the receiver thinks it is.

Does the LFSR sequence, as an encryption method, meet the above four security objectives? First, a binary LFSR sequence stream cipher achieves *secrecy* by adding the sequence to the binary representation of the message. The LFSR sequence appears as noise added to the message, thus frustrating anyone who intercepts the message. The problem with relying on LSFR sequences for secrecy is that the minimal connection polynomial of the sequence is easily determined using the Berlekamp-Massey Algorithm, which is discussed in §3.2. The minimal connection polynomial is the key necessary to generate the LFSR sequence used as the stream cipher. With this key, the eavesdropper Eve can easily add the sequence to the ciphertext to retrieve the original message. One method Alice and Bob can use to have more secrecy in the stream cipher is by picking LFSR sequences with extremely long periods (i.e. period length $p \approx 10^{50}$). With long period lengths, Eve needs a longer amount of ciphertext to find the minimal connection polynomial and she needs more terms of the sequence to determine the correct polynomial.

LFSR sequences combined with some error-correcting codes can provide a limited amount of integrity. However, on its own, a LFSR sequence provides very little integrity. Likewise, on its own, a LFSR sequence provides very little in terms of non-repudiation.

In order to provide for authentication, Alice and Bob could each create their own stream cipher using a different LFSR sequence. Alice would then send her key and initial fill to Bob and Bob would send his key and initial fill to Alice. Alice would then add together her sequence and Bob's sequence to obtain a new sequence. She would use this sequence as the stream cipher and encode her message to send to Bob. Bob would also add together the two sequences to obtain the same stream cipher. This would allow Alice and Bob to verify that they are talking with each other since they created their stream cipher by adding their own individual sequences.

2.2 Pseudo-random Binary Sequences

Any binary sequence that has Golumb's three properties stated below is considered to be *pseudo-random*. One type of stream cipher that is used to create ciphertexts is pseudo-random binary sequences. Linear feedback shift register sequences are one type of pseudo-random binary sequence that are easily generated by linear feedback shift registers.

Ideally, a cryptographic bit-stream sequence would have infinite length and complete randomness. The reality of practical application and construction techniques necessitates the use of only finite sequences. Since finite sequences can never be truly random, there are certain properties singled out that are associated with randomness. *Golomb's properties* are [1]:

1. *Balance*: The number of 1's is approximately equal to the number of 0's. (More generally, each symbol occurs with approximately equal frequency.)

2. *Proportional runs*: The runs of consecutive 1's or 0's frequently occur with short runs more frequent than long runs. (More generally, runs of a given symbol of a shorter length occur more frequently than those of a given longer length.)

3. *Low autocorrelation*: The sequence possesses an **auto-correlation function**, which is peaked in the middle and tapering off rapidly at the ends.

The auto-correlation function is a way to quantize how random a sequence is and is defined by

$$AC(k) = \frac{1}{p}\sum_{i=1}^{p} x_i \cdot x_{i+k}$$

where p is the period of the sequence $\{x_i\}$ and the sum is taken as a real number not as an element of the finite field with 2 elements. When $0 < k < p$, $AC(k)$ is close to zero (meaning there is very little correlation of the sequence with itself) and $AC(0) = \frac{1}{2}$, since by the Balance property, $1/2$ the elements of the sequence are 0.

According to Golomb, linear feedback shift register sequences of maximal period fulfill the above three properties and are therefore pseudo-random. Linear Feedback Shift Registers are also easy to use as stream ciphers since adding the LFSR sequence to the message encodes it and adding the sequence again to the ciphertext returns the original message.

Example 3 *Next, a very simple example is provided to show how a message might be encrpypted for secrecy and then decrypted by the authorized receiver so that the information can be read. In order for Alice and Bob to send messages to each other using computers, they must convert the english syntax of the message into a binary form. For our purposes we will allow the following table define our alphabet.*

$SPACE$	=	0000	L	=	1111
A	=	0001	!	=	1000
B	=	0010	#	=	1001
E	=	0011	?	=	1010
M	=	0100	;	=	1011
R	=	0101	,	=	1100
T	=	0110	.	=	1101
Y	=	0111	Q	=	1110

Suppose Alice wanted to send the message M ="BEAT ARMY!" to Bob, but she wanted to keep it secret. By converting "BEAT ARMY!" from english to binary, we have a string of binary bits

$$M = 001000110001011000000001010101000111000$$

In order to generate a stream cipher that only Alice and Bob know, each person will generate a pseudo-random sequence of sufficient length. Alice will send her sequence to Bob and Bob will send his sequence to Alice. The two people will then add their sequences together bit-wise to produce a common stream cipher. For our purposes, suppose the resultant sequence from the two individual sequences is

$$C = 11010110010001111010110010000111101011001$$

Alice now has a message in binary format and a cipher to encode the message. Adding the cipher to the message, Alice gets the encrypted message or ciphertext, E.

$$\begin{array}{rll} & 00100011000101100000000101010100011111000 & M \\ + & \underline{11010110010001111010110010000111101011001} & = + \quad \underline{C} \\ & 11110101010100011010110111011011001000001 & E \end{array}$$

If a third party, Eve, were to intercept the ciphertext and tried to read it, knowing that the computers used the above table to talk to one another, he would decrypt the ciphertext as "LRRA?..;BA". Notice that both 'A's in the original message where changed to two different letters ('R' the first time and '.' the second time) and that 'E' and 'A' both are changed to 'R' and 'T' and '!' are both changed to 'A'. This helps prevent a cryptanalyst from breaking the code by mapping each character to something different everytime. The mapping appears random since the stream cipher used was a pseudo-random sequence.

When Bob receives the ciphertext, however, he is able to decrypt it since he has the stream cipher that he and Alice created earlier. By adding the stream cipher to the ciphertext, he will uncover the original message.

$$\begin{array}{rll} & 11110101010100011010110111011011001000001 & E \\ + & \underline{11010110010001111010110010000111101011001} & = + \quad \underline{C} \\ & 00100011000101100000000101010100011111000 & M \end{array}$$

Now Bob can use the table to decode the message from binary into english and receives "BEAT ARMY!" from Alice.

3 How to Find the Connection Polynomial

3.1 Recovering the Connection Polynomial from a LFSR Sequence

In some cases, it is necessary to recover the connection polynomial of a LFSR sequence from the sequence itself. This is true when attempting to do cryptanalysis on a piece of intercepted code. When a part of the stream cipher is intercepted, the connection polynomial can be recovered even if the number of bits of the cipher is less than the period of the sequence. Once this polynomial is known, the entire cipher can be generated and any messages that are encrypted using that particular sequence as a stream cipher can be decrypted and read by the third party.

3.2 Massey's Algorithm

An algorithm exists that provides a connection polynomial given only a few terms of a LFSR sequence. This algorithm is known as the Berlekamp-Massey algorithm. One is able to determine the connection polynomial of a LFSR sequence of period 15 with only 8 terms of the sequence. This can be generalized since if we know that a sequence has a minimal connection polynomial with degree $\leq L$, then only $2 \cdot L$ terms of the sequence need to be known in order to determine the correct connection polynomial. We can determine L if we know the period length of the sequence, since the period $p = 2^L - 1$ [3]. This is an extremely powerful tool for cryptanalysts trying to break stream ciphers generated from LFSRs, since only a relatively small sample of a long period sequence is needed to break the cipher. The algorithm as it is described by James Massey is presented below [4]:

Input: a LFSR sequence of length n.

Output: a connection polynomial $C(x)$ of the minimal LFSR.

1. Initialize the algorithm by setting $C(x) = 1$, $B(x) = 1$, $m = 1$, $b = 1$, $L = 0$, and, $N = 0$.
2. If $N = n$, then terminate, otherwise calculate the discrepancy

$$d = s_N + \sum_{i=1}^{L} c_i \cdot s_{N-i}$$

3. If $d = 0$, then $m = m + 1$, go to step 6.
4. If $d \neq 0$ and $2 \cdot L > N$, then calculate $C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x)$, $m = m + 1$, go to step 6.
5. If $d \neq 0$ and $2 \cdot L \leq N$, then set $T(x) = C(x)$, calculate $C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x)$, $L = N + 1 - L$, $B(x) = T(x)$,and set $m = 1$ and $b = d$, go to step 6.
6. Calculate $N = N + 1$ and repeat steps 2 through 6.

This algorithm determines whether or not the current connection polynomial $C(x)$ can correctly produce the next term of the given sequence. If it can, the discrepancy $d = 0$ and the algorithm leaves $C(x)$ unchanged and iterates to the next step. If $C(x)$ does not provide the next term of the sequence, the discrepancy $d \neq 0$ and a new $C(x)$ is calculated as in steps 4 and 5 above. The algorithm is complicated and mysterious enough to warrant a complete example, showing all steps.

Example 4 *The LFSR sequence used in this example:* 110101100100011. *We take the algorithm all the way out to the termination when $N = n$. Though this sequence is of length $n = 15$, we arrive at the correct connection polynomial $C(x)$*

after only 8 iterations of the algorithm. Iterations 9 through 15 return a discrepancy $d = 0$ which causes the algorithm to return the connection polynomial calculated in the previous iteration.

1. *Step 1:* $C(x) = 1$,$B(x) = 1$, $m = 1$, $b = 1$, $L = 0$, $N = 0 \neq 15 = n$.

   Step 2: Find the discrepancy

   $$d = s_0 + \sum_{i=1}^{0} c_i \cdot s_{0-i} = s_0 = 1$$

   since $d \neq 0$, *we compare* $2 \cdot L$ *to* N

   $$2 \cdot L = 2 \cdot 0 = 0 \leq 0 = N$$

   go to step 5.

   Step 5: Calculate $C(x)$

   $$T(x) = C(x) = 1$$

   $$C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x) = 1 - x$$
   $$L = N + 1 - L = 0 + 1 - 0 = 1$$
   $$B(x) = T(x) = 1$$
   $$b = d = 1$$
   $$m = 1$$

   go to step 6.

   Step 6: Increase N

   $$N = N + 1 = 0 + 1 = 1$$

2. *Step 1:* $C(x) = 1 - x$, $B(x) = 1$, $m = 1$, $b = 1$, $L = 1$, $N = 1 \neq 15 = n$.

   Step 2: Find the discrepancy

   $$d = s_1 + \sum_{i=1}^{1} c_i \cdot s_{1-i} = s_1 + c_1 \cdot s_0 = 0$$

   Step 3: Since $d = 0$, $m = m + 1 = 1 + 1 = 2$, *and we skip to step 6.*

   Step 6: Increase N

   $$N = N + 1 = 1 + 1 = 2$$

3. *Step 1:* $C(x) = 1 - x$, $B(x) = 1$, $m = 2$, $b = 1$, $L = 1$, $N = 2 \neq 15 = n$.

   Step 2: Find the discrepancy

$$d = s_2 + \sum_{i=1}^{1} c_i \cdot s_{2-i} = s_2 + c_1 \cdot s_1 = 1$$

since $d \neq 0$, *we compare* $2 \cdot L$ *and* N

$$2 \cdot L = 2 \cdot 1 = 2 \leq 2 = N$$

go to step 5.

Step 5: Calculate $C(x)$

$$\begin{gathered} T(x) = C(x) = 1 - x \\ C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x) = -x - x^2 \\ L = N + 1 - L = 2 + 1 - 1 = 2 \\ B(x) = T(x) = 1 - x \\ b = d = 1 \\ m = 1 \end{gathered}$$

go to step 6.

Step 6: Increase N

$$N = N + 1 = 2 + 1 = 3$$

4. *Step 1:* $C(x) = 1 - x - x^2$, $B(x) = 1 - x$, $m = 1$, $b = 1$, $L = 2$, $N = 3 \neq 15 = n$.

   Step 2: Find the discrepancy

$$d = s_3 + \sum_{i=1}^{2} c_i \cdot s_{3-i} = s_3 + c_1 \cdot s_2 + c_2 \cdot s_1 = 1 + 1 \cdot 0 + 1 \cdot 1 = 0$$

   since $d = 0$, $m = m + 1 = 1 + 1 = 2$, *and we skip to step 6.*

   Step 6: Increase N

$$N = N + 1 = 3 + 1 = 4$$

5. *Step 1:* $C(x) = 1 - x - x^2$, $B(x) = 1 - x$, $m = 2$, $b = 1$, $L = 2$, $N = 4 \neq 15 = n$.

   Step 2: Find the discrepancy

$$d = s_4 + \sum_{i=1}^{2} c_i \cdot s_{4-1} = s_4 + c_1 \cdot s_3 + c_2 \cdot s_2 = 0 + 1 \cdot 1 + 1 \cdot 0 = 1$$

Step 3: Since $d \neq 0$, we compare $2 \cdot L$ and N

$$2 \cdot L = 2 \cdot 2 = 4 \leq 4 = N$$

go to step 5.

Step 5: Calculate $C(x)$

$$\begin{gathered} T(x) = C(x) = 1 - x - x^2 \\ C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x) = 1 - x - x^3 \\ L = N + 1 - L = 4 + 1 - 2 = 3 \\ B(x) = T(x) = 1 - x - x^2 \\ b = d = 1 \\ m = 1 \end{gathered}$$

go to step 6.

Step 6: Increase N

$$N = N + 1 = 4 + 1 = 5$$

6. *Step 1:* $C(x) = 1 - x - x^3$, $B(x) = 1 - x - x^2$, $m = 1$, $b = 1$, $L = 3$, $N = 5 \neq 15 = n$.

   Step 2: Find the discrepancy

   $$d = s_5 + \sum_{i=1}^{3} c_i \cdot s_{5-i} = s_5 + c_1 \cdot s_4 + c_2 \cdot s_3 + c_3 \cdot s_2 = 1 + 1 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 = 1$$

   since $d \neq 0$ we compare $2 \cdot L$ and N

   $$2 \cdot L = 2 \cdot 3 = 6 > 5 = N$$

   go to step 4.

   Step 4: Calculate $C(x)$

   $$\begin{gathered} C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x) = 1 + x^2 \\ m = m + 1 = 1 + 1 = 2 \end{gathered}$$

   go to step 6.

   Step 6: Increase N

   $$N = N + 1 = 5 + 1 = 6$$

7. *Step 1:* $C(x) = 1 + x^2$, $B(x) = 1 - x - x^2, m = 2, b = 1, L = 3, N = 6 \neq 15 = n$.

   Step 2: Find the discrepancy

$$d = s_6 + \sum_{i=1}^{3} c_i \cdot s_{6-i} = s_6 + c_1 \cdot s_5 + c_2 \cdot s_4 + c_3 \cdot s_3 = 1 + 0 \cdot 1 + 1 \cdot 0 + 0 \cdot 1 = 1$$

since $d \neq 0$ *we compare* $2 \cdot L$ *and* N

$$2 \cdot L = 2 \cdot 3 = 6 \leq 6 = N$$

go to step 5.

Step 5: Calculate $C(x)$

$$\begin{gathered} T(x) = C(x) = 1 + x^2 \\ C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x) = 1 + x^3 + x^4 \\ L = N + 1 - L = 6 + 1 - 3 = 4 \\ B(x) = T(x) = 1 + x^2 \\ b = d = 1 \\ m = 1 \end{gathered}$$

go to step 6.

Step 6: Increase N

$$N = N + 1 = 6 + 1 = 7$$

8. *Step 1:* $C(x) = 1 + x^3 + x^4, B(x) = 1 + x^2, m = 1, b = 1, L = 4, N = 7 \neq 15 = n.$

   Step 2: Find the discrepancy

$$d = s_7 + \sum_{i=1}^{4} c_i \cdot s_{7-i} = s_7 + c_1 \cdot s_6 + c_2 \cdot s_5 + c_3 \cdot s_4 + c_4 \cdot s_3 = 1$$

   since $d \neq 0$ *we compare* $2 \cdot L$ *and* N

$$2 \cdot L = 2 \cdot 4 = 6 > 7 = N$$

   go to step 4.

   Step 4: Calculate $C(x)$

$$C(x) = C(x) - d \cdot b^{-1} \cdot x^m \cdot B(x) = 1 - x + x^4$$

$$m = m + 1 = 1 + 1 = 2$$

   go to step 6.

   Step 6: Increase N

$$N = N + 1 = 7 + 1 = 8$$

9. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 2$, $b = 1$, $L = 4$, $N = 8 \neq 15 = n$.

   Step 2: Find the discrepancy

$$d = s_8 + \sum_{i=1}^{4} c_i \cdot s_{8-i} = s_8 + c_1 \cdot s_7 + c_2 \cdot s_6 + c_3 \cdot s_5 + c_4 \cdot s_4 = 0$$

   Step 3: Since $d = 0$, $m = m + 1 = 2 + 1 = 3$, *and we skip to step 6.*

   Step 6: Increase N

$$N = N + 1 = 8 + 1 = 9$$

10. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 3$, $b = 1$, $L = 4$, $N = 9 \neq 15 = n$.

    Step 2: Find the discrepancy

$$d = s_9 + \sum_{i=1}^{4} c_i \cdot s_{9-i} = s_9 + c_1 \cdot s_8 + c_2 \cdot s_7 + c_3 \cdot s_6 + c_4 \cdot s_5 = 0$$

    Step 3: Since $d = 0$, $m = m + 1 = 3 + 1 = 4$, *and we skip to step 6.*

    Step 6: Increase N

$$N = N + 1 = 9 + 1 = 10$$

11. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 4$, $b = 1$, $L = 4$, $N = 10 \neq 15 = n$.

    Step 2: Find the discrepancy

$$d = s_{10} + \sum_{i=1}^{4} c_i \cdot s_{10-i} = s_{10} + c_1 \cdot s_9 + c_2 \cdot s_8 + c_3 \cdot s_7 + c_4 \cdot s_6 = 0$$

    Step 3: since $d = 0$, $m = m + 1 = 4 + 1 = 5$, *and we skip to step 6.*

    Step 6: Increase N

$$N = N + 1 = 10 + 1 = 11$$

12. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 5$, $b = 1$, $L = 4$, $N = 11 \neq 15 = n$.

    Step 2: Find the discrepancy

$$d = s_{11} + \sum_{i=1}^{4} c_i \cdot s_{11-i} = s_{11} + c_1 \cdot s_{10} + c_2 \cdot s_9 + c_3 \cdot s_8 + c_4 \cdot s_7 = 0$$

Step 3: Since $d = 0$, $m = m + 1 = 5 + 1 = 6$, *and we skip to step 6.*

Step 6: Increase N

$$N = N + 1 = 11 + 1 = 12$$

13. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 6$, $b = 1$, $L = 4$, $N = 12 \neq 15 = n$.

Step 2: Find the discrepancy

$$d = s_{12} + \sum_{i=1}^{4} c_i \cdot s_{12-i} = s_{12} + c_1 \cdot s_{11} + c_2 \cdot s_{10} + c_3 \cdot s_9 + c_4 \cdot s_8 = 0$$

Step 3: Since $d = 0$, $m = m + 1 = 6 + 1 = 7$, *and we skip to step 6.*

Step 6: Increase N

$$N = N + 1 = 12 + 1 = 13$$

14. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 7$, $b = 1$, $L = 4$, $N = 13 \neq 15 = n$.

Step 2: Find the discrepancy

$$d = s_{13} + \sum_{i=1}^{4} c_i \cdot s_{13-i} = s_{13} + c_1 \cdot s_{12} + c_2 \cdot s_{11} + c_3 \cdot s_{10} + c_4 \cdot s_9 = 0$$

Step 3: Since $d = 0$, $m = m + 1 = 7 + 1 = 8$, *and we skip to step 6.*

Step 6: Increase N

$$N = N + 1 = 13 + 1 = 14$$

15. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 8$, $b = 1$, $L = 4$, $N = 14 \neq 15 = n$.

Step 2: Find the discrepancy

$$d = s_{14} + \sum_{i=1}^{4} c_i \cdot s_{14-i} = s_{14} + c_1 \cdot s_{13} + c_2 \cdot s_{12} + c_3 \cdot s_{11} + c_4 \cdot s_{10} = 0$$

Step 3: Since $d = 0$, $m = m + 1 = 8 + 1 = 9$, *and we skip to step 6.*

Step 6: Increase N

$$N = N + 1 = 14 + 1 = 15$$

16. *Step 1:* $C(x) = 1 - x + x^4$, $B(x) = 1 + x^2$, $m = 9$, $b = 1$, $L = 4$, $N = 15 = n$. *terminate the algorithm*

At this point the algorithm outputs the last value of L *and* $C(x)$ *which are* $L = 4$ *and* $C(x) = 1 - x + x^4$ *. The algorithm terminates since* $N = n$*. Figure 2 (from Massey [4]) depicts the minimal LFSR found in this example. Since the coefficients* (c_1, c_2, c_3, c_4) *of the connection polynomial* $C(x) = 1 - x + x^4$ *are* $(1, 0, 0, 1)$ *we know that the inputs for the mod 2 adder are taken off the first and forth registers. Since* $L = 4$*, we know that the LFSR must have a minimum of four registers. It should be noted that, since this example uses the binary case of GF(2),* -1 *and* $+1$ *are the same modulo 2.*

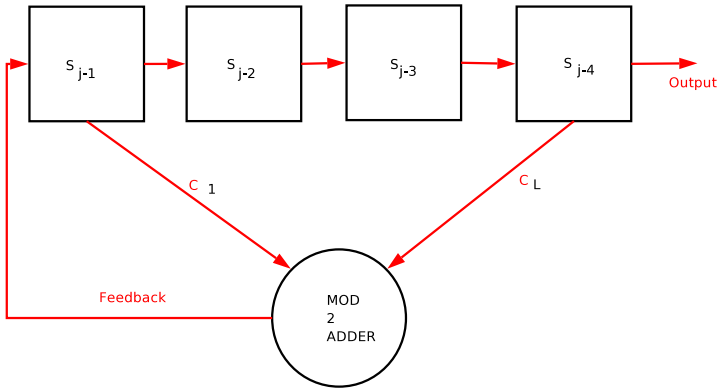

Figure 2. The minimal LFSR with C(x)=1-x+x^4 and L = 4.

This determines the sequence, as desired.

4 SAGE

SAGE (Software for Algebra and Geometry Experimentation) is a computer system by many contributors led by Professor William Stein of the University of Washington. It is a free, open source program that can be used to help research and teaching many different types of mathematics, including: algebra, geometry, and number theory. While conducting research for this paper, the author utilized SAGE to explore cryptography and wrote three functions for a crypto package that is now included in SAGE. For more information regarding SAGE, the reader is invited to visit `http://sage.scipy.org`.

Example 5 *Finally, we present the interested reader with the SAGE syntax needed to compute the above connection polynomial in the above example using SAGE [9]:*

```
sage: F = GF(2)
sage: F
Finite Field of size 2
sage: o = F(0); l = F(1)
sage: key = [l,o,o,l]; fill = [l,l,o,l]; n = 20
```

```
sage: s = lfsr_sequence(key,fill,n); s
[1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0]
sage: lfsr_connection_polynomial(s)
x^4 + x + 1
```

This implementation also works in the non-binary case. As shown below

```
sage: F = GF(3)
sage: F
Finite Field of size 3
sage: o = F(0); l = F(1); t = F(2);
sage: key = [l,o,o,t]; fill = [t,l,o,t]; n = 20
sage: s = lfsr_sequence(key,fill,n); s
[2, 1, 0, 2, 0, 1, 2, 0, 0, 1, 1, 2, 1, 0, 1, 1, 0, 0, 1, 0]
sage: lfsr_connection_polynomial(s)
2*x^4 + x + 1
```

5 Open question

It is known that any periodic sequence can be modeled using a linear feedback shift register sequence [10]; the *linear complexity* of the sequence is the length of the key of the LFSR sequence that models it.

Open Question: What is the linear complexity of each of the Enigma machines?

This does not seem to be explicitly presented in the literature.

Acknowledgements: I thank M. Lucas, R. Rivas, and Professors D. Joyner and G. Price of the US Naval Academy for very helpful discussions on this paper.

References

[1] S. W. Golomb. Shift Register Sequences. Aegean Park Press Laguna Hills, CA, USA 1981.

[2] T. W. Körner, *Codes and Cryptography* lecture notes.
Available online: `http://www.dpmms.cam.ac.uk/~twk/`

[3] R. Lidl and H. Niederreiter. Introduction to Finite Fields and Their Applications Revised Edition. Cambridge University Press, Cambridge; 1994 pp235-239.

[4] J. L. Massey, "Shift-Register Synthesis and BCH Decoding." IEEE Trans. on Information Theory, vol. 15(1), pp. 122-127, Jan 1969.

[5] A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography. CRC Press, Inc.; 1997 pp1-4. Available online at `http://www.cacr.math.uwaterloo.ca/hac/`

[6] S. Matyas and C. Meyer. Cryptography: A New Dimension in Computer Data Security-A Guide for the Design and Implementation of Secure Systems. John Wiley & Sons, Inc. 1982. pp. 53.

[7] R. Paddock, "A Guide to Online Information About: Noise/Chaos/Random Numbers and Linear Feedback Shift Registers." Circuit Cellar Online: The Magazine for Computer Applications. `http://www.designer-iii.com/cco/noise/c89r4.htm`, last modified April 17, 2005.

[8] A. Sherman, "On the Enigma cryptograph and formal definitions of cryptographic strength," Masters thesis (advisor R. Rivest), MIT, 1981.

[9] William Stein, David Joyner, *SAGE: System for Algebra and Geometry Experimentation*, Comm. Computer Algebra 39(2005)61-64. SAGE is available for download at `http://sage.scipy.org` (the article can be downloaded from `http://sage.scipy.org/sage/misc/sage_sigsam_updated.pdf`)

[10] H. van Tilborg. "Coding Theory at Work in Cryptology and Vice Versa." Handbook of Coding Theory, vol. 2. Ed. by V.S. Pless and W.C. Huffman. Elsevier Science B.V., 1998. pp. 1195-1226.