F. G. Gustavson

Analysis of the Berlekamp-Massey Linear Feedback Shift-Register Synthesis Algorithm

Abstract: An analysis of the Berlekamp-Massey Linear Feedback Shift-Register (LFSR) Synthesis Algorithm is provided which shows that an input string of length n requires $\mathcal{O}(n^2)$ multiplication/addition operations in the underlying field of definition. We also derive the length distribution for digit strings of length n. Results show that, on the average, the encoded length is no greater than $n+1$. Furthermore, we exhibit a connection between step 1 of the Ling-Palermo algorithm and the LFSR Algorithm, and the LFSR Algorithm turns out to be computationally superior.

Introduction

The Block-Oriented Information Compression (BOIC) scheme of Ling and Palermo [1] is isomorphic to the Linear Feedback Shift-Register (LFSR) Synthesis Algorithm proposed by Massey [2] when one deals with bit strings. The LFSR algorithm, which is essentially the same as the Berlekamp Iterative Algorithm [3], was developed for encoding and is more general and computationally faster than the BOIC scheme. This work contains analysis of the LFSR scheme; in particular, we derive the distribution $D(n; l)$ of lengths l needed to generate strings of length n. In the binary case, $D(n; l)$ appears when the binary symmetric source is assumed.

The distribution $D(n; l)$ can be used to compute the expected value $\mathrm{E}(n)$ of a random bit string. It turns out that $n \leq \mathrm{E}(n) \leq n+1$; hence one gets, on the average, data expansion with this method. [The length of n ($\log_2 n$ bits) is not included in this estimate.] A fundamental result in information theory [4, p. 43, Theorem 3.11] states that any encoding must lead, on the average, to data expansion; the result here is quite good in view of that fact.

The following section of this paper can be considered a supplement to [2] and [3, Chapter 7]. In it we present Massey's algorithm and indicate which steps contribute to the computation cost. We then prove what the minimum, average, and maximum computation costs are in terms of the numbers of multiplications and additions. To first order, these three values are 0, $\frac{1}{4}n^2\,(2-p^{-1})$, and $\frac{1}{2}n^2$ when computations are done in a field with p elements. We determine that the average number of polynomial evaluations is $n(2-p^{-1})$. This section also contains a derivation of formulas for $D(n; l)$ and $\mathrm{E}(n)$.

The next section contains a comparison of step 1 of the BOIC scheme and the LFSR Algorithm. We show there that steps 2, 3, and 4 of the BOIC scheme compression stage are redundant; i.e., the result they seek is contained in step 1. Then we show that step 1 requires more work than the LFSR Algorithm; step 1 turns out to be $\mathcal{O}(n^3)$ for most sequences. We use the notation of [1] and refer to specific equations there. We do not give a parallel evaluation of the LFSR Algorithm and the BOIC scheme. However, assuming that we can count n bit operations as one operation, we find that the BOIC scheme can be done in $\mathcal{O}(n^2)$ operations whereas the LFSR Algorithm requires no more than $\mathcal{O}(n \log_2 n)$ operations.

Description and specification of the LFSR Algorithm

The problem: Given n elements $s_1, s_2, \cdots, s_n$ in a finite or an infinite field, find l constants $c_1, c_2, \cdots, c_l$ in the same field such that

$$s_{j+1} = -\sum_{\nu=1}^{l} c_\nu s_{j+1-\nu}, \qquad j = l, \cdots, n-1. \tag{1}$$

Let $L_n(s)$ be the smallest value of l, for which a shift-register exists, that generates $s_1, \cdots, s_n$; we wish to find $c_1, \cdots, c_l$ with $l = L_n(s)$. The following result from [2, p. 124] is most useful:

$$L_{n+1}(s) = \max\,[L_n(s),\, n+1-L_n(s)]. \tag{2}$$

It says, suppose (1) generates $s_1, \cdots, s_n$ with $L_n(s)$ constants. If (1) is satisfied for $j = n$ also, then $L_{n+1}(s) = L_n(s)$ and the same constants may be used. If (1) is not satisfied for $j = n$, then

$$L_{n+1}(s) = \begin{cases} L_n(s) & \text{if } 2L_n(s) > n, \\ n+1-L_n(s) & \text{if } 2L_n(s) \leq n, \end{cases}$$

and $L_{n+1}(s)$ new constants must be found. Let

$$C(x) = 1 + c_1x + \cdots + c_lx^l$$

denote a polynomial of degree at most l in the indeterminate x. We now state the LFSR Algorithm [2, p. 124]:

1. Initialization:

   $C(x) \leftarrow B(x) \leftarrow 1$,

   $j \leftarrow b \leftarrow 1 + l \leftarrow k \leftarrow 0$.

2. If $k = n$, stop; else compute

   $$d = s_{k+1} + \sum_{i=1}^{l} c_i s_{k+1-i}.$$

3. If $d = 0$, then GO TO 6; else GO TO 4.

4. If $2l > k$, then

   $C(x) \leftarrow C(x) - bd^{-1}x^jB(x)$,

   GO TO 6;

   else GO TO 5.

5. $T(x) \leftarrow C(x)$,

   $C(x) \leftarrow C(x) - db^{-1}x^jB(x)$,

   $B(x) \leftarrow T(x)$,

   $l \leftarrow k + 1 - l$,

   $b \leftarrow d$,

   $j \leftarrow 0$.

6. $j \leftarrow j + 1$,

   $k \leftarrow k + 1$,

   GO TO 2.

The computational effort occurs in steps 2, 4, and 5 and we endeavor to find minimum, average, and maximum computation costs of performing these steps. This computation is performed in a field containing p elements. By holding n fixed but letting $p \to \infty$, we can obtain results for the real number field. Another result we obtain is the distribution of lengths, $D(n; l)$, of the minimal LFSR's; our method of proof is induction.

• *Length distribution of minimal LFSR's*

Proposition 1 The distribution $D(l; n)$ of minimal LFSR's is given by

$$D(n; l) = \begin{cases} 1 & \text{if } l = 0; \\ p^{2l-1}\bar{p} & \text{if } l = 1, \cdots, \alpha; \\ p^{2(n-l)}\bar{p} & \text{if } l = \alpha + 1, \cdots, n, \end{cases} \qquad (3)$$

where $\bar{p} = p - 1$, $\alpha = \lfloor n/2 \rfloor$, and $\lfloor x \rfloor$, $\lceil x \rceil$ denote respectively the floor, ceiling of x.

Table 1 Length distribution $D(n; l)$ for the LFSR Synthesis Algorithm; $p = 2$.

	l										
n	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	1	1									
2	1	2	1								
3	1	2	4	1							
4	1	2	8	4	1						
5	1	2	8	16	4	1					
6	1	2	8	32	16	4	1				
7	1	2	8	32	64	16	4	1			
8	1	2	8	32	128	64	16	4	1		
9	1	2	8	32	128	256	64	16	4	1	
10	1	2	8	32	128	512	256	64	16	4	1

Proof Suppose (3) is true for $n = k$ and consider vectors $s_1, \cdots, s_k, s_{k+1}$. Here s_{k+1} can assume p values, and $\bar{p}$ of these values cause Eq. (1) to fail; for these cases l must be corrected via formula (2). On the other hand, s_{k+1} satisfies Eq. (1) for exactly one value and then $L_{k+1}(s) = L_k(s)$. It follows that p^k vectors retain the same length and the same set of c's. The remaining $p^k\bar{p}$ vectors get a new set of c's and a possible length change. We consider three cases:

Case 1: $0 \le l \le \alpha$
By induction, there are $p^{2l-1}\bar{p}$ vectors $s_1, \cdots, s_k$ whose minimum length generators have length l. Letting s_{k+1} be arbitrary, we have $p^{2l}\bar{p}$ vectors $s_1, \cdots, s_k, s_{k+1}$ to consider. Of these $p^{2l-1}\bar{p}$ remain of length l; $p^{2l-1}\bar{p}^2$ experience a length change to $k + 1 - l$.

Case 2: $k + 1 - \alpha \le l \le k + 1$
By induction, there are $p^{2(k-l)}\bar{p}$ vectors $s_1, \cdots, s_k$ of formula length l; there is no vector with formula of length $l = k + 1$. Adding component s_{k+1}, we have $p^{2(k-l)+1}\bar{p}$ vectors to consider. None of these experiences a length change because $l > k + 1 - l$. Summing these two components we obtain $p^{2(k+1-l)}\bar{p}$ formulas of length l.

Case 3: $\alpha < k - \alpha = l$
This case occurs only when k is odd. We consider formulas of length $\alpha + 1$, the term missing in Case 2. By induction, there are $p^{2(k-\alpha-1)}\bar{p}$ formulas of this length; adding component s_{k+1}, we obtain $p^{2(k-\alpha)}\bar{p}$ formulas to consider. All have length $l = \alpha + 1$ because $2(\alpha + 1) = k + 1$.

Combining Cases 1, 2, and 3, we see that Eq. (3) holds with $n = k$ replaced by $n = k + 1$. To start the induction, we hypothesize that there is one formula (of length zero) when $n = 0$. Table 1 displays the length distribution for the Boolean case ($p = 2$) for values of n between one and ten.

Next, we compute the length of an average formula, $E(n)$, assuming random sequences. By definition,

$$E(n) = \left[\sum_{\nu=0}^{n} 2\nu D(n;\nu)\right]/p^n.$$

The factor 2 is due to the fact that both $s_1, \cdots, s_l$ and $c_1, \cdots, c_l$ must be saved to compute $s_1, \cdots, s_n$. By grouping sum terms ν and $n-\nu$, we may write

$$2\nu D(n;\nu) + 2(n-\nu)D(n;n-\nu)$$
$$= n(D(n;\nu) + D(n;n-\nu)) + c(n;\nu)$$

where

$$c(n;\nu) = \begin{cases} n(\bar{p}-1) & \text{if } \nu = 0, \\ \bar{p}^2(n-2\nu)p^{2\nu-1} & \text{if } \nu \neq 0. \end{cases}$$

Hence,

$$E(n) = n + \left[\sum_{\nu=0}^{\beta-1} c(n;\nu)\right]/p^n,$$

where $\beta = \lceil n/2 \rceil$. Using formulas for the sum and derivative of a geometric series, we get $E(n) = n + N/D$, where

$$N = (n-2\beta+2)p^{2\beta+1} + (2\beta-n)p^{2\beta-1}$$
$$- 2(n+1)p - 2n$$

and

$$D = (p+1)^2 p^n.$$

We claim $0 \leq N < D$. That $0 \leq N$ follows from

$$N = \left[\sum_{\nu=0}^{\beta-1} c(n;\nu)\right](p+1)^2.$$

And,

$$N < p^{2\beta-1}[p^2(n+2-2\beta) + (2\beta-n)]$$
$$= \begin{cases} 2p^{n+1} & \text{if } n \text{ is even,} \\ p^n(1+p^2) & \text{if } n \text{ is odd.} \end{cases}$$

In either case, $N < D$. We summarize these facts in

$$n \leq E(n) < n+1,$$

which says that the expected value of the encoding length is always between n and $n+1$. Thus, on the average, one expects a slight data expansion. For example, when $p=2$ and n is large, we get $N/D = 4/9$ for n even and $5/9$ for n odd.

Computation cost of the LFSR Algorithm

We want to establish the minimum, average, and maximum "costs" (i.e., the numbers of multiplication operations) of the LFSR Algorithm in terms of executing steps 2, 4, and 5, and also the number of times these steps are entered. For the minimum and maximum costs we describe the types of input data strings that cause the LFSR Algorithm to do the least and the most multiplication. The average cost is handled by computing the total cost of processing each of the p^n possible input data strings one time and then dividing this total by p^n. A direct measure of the total cost of processing each of the p^n possible input strings one time is the sum of all multiplications done by the LFSR Algorithm during this processing. We call this the sum multiplication count and we determine it by deriving and solving recurrence relations. More specifically, we derive multiplication counts for steps 2, 4, and 5 at iteration step $\nu = 0, 1, \cdots, n-1$ and then sum over ν to obtain the total multiplication count of these steps. The average cost is then the sum of these three counts divided by p^n.

In the same manner we compute the number of times steps 2, 4, and 5 are executed. These counts refer to the average number of polynomial evaluations required by the LFSR Algorithm.

It should be noted that in the Boolean case no multiplication need be done in steps 4 and 5. This is because db^{-1} always equals one and thus the polynomial operations there become EXCLUSIVE OR and shift operations. However, the add count is *identical* to the multiplication count for LFSR's when $p > 2$; thus in the Boolean case the reader should substitute *add* for multiply in what follows. This is what we meant above when we said the multiplication count is a direct measure of the total cost of processing.

Before turning to an analysis of steps 2, 4, and 5 to determine the average cost, we state some ground rules: The computations in these steps require $l(k)$ multiplications and additions, where $l(k)$ is the value of l at the end of the particular step during the kth iteration. We neglect the multiplication required to compute bd^{-1}, which is justified by a remark at the end of the following section. We ignore operations on the constant term of $C(x)$ because it always equals one, and ignore savings due to the occasional cancellation of leading terms in $C(x)$.

- *Average cost*

Number of polynomial evaluations

Let u, v, and w be vectors of length n whose νth components are the number of times, respectively, that $d=0$ in step 3, that step 4 is done with $2l > k$, and that step 5 is done at iteration step ν. For any $s = s_1, \cdots, s_n$ and $1 \leq \nu \leq n$, we have

$$u(\nu) = 1,\ v(\nu) = w(\nu) = 0, \text{ or}$$

$$v(\nu) = 1,\ u(\nu) = w(\nu) = 0, \text{ or}$$

$$w(\nu) = 1,\ u(\nu) = v(\nu) = 0,$$

and one of these three possibilities must occur. Now consider vectors $U(n)$, $V(n)$ and $W(n)$ which are ob-

tained by summing u, v, and w over all possible p^n input strings s. For each integer n, $U(n)$ is a vector of length n; $U(n;\nu)$ is the νth component of $U(n)$, $\nu=0,\cdots,n-1$. The following recurrence relations hold:

$$U(n+1;n) = p^n;$$

$$V(n+1;n) = (p-1)\sum_{\nu=\alpha+1}^{n} D(n;\nu) = (p-1)\frac{p^{2\beta}-1}{p+1};$$

$$W(n+1;n) = (p-1)\sum_{\nu=0}^{\alpha} D(n;\nu) = (p-1)\frac{p^{2\alpha+1}+1}{p+1}, \quad (4)$$

where $\alpha=\lfloor n/2\rfloor$, $\beta=\lceil n/2\rceil$, and $n=0,1,\cdots$; also

$$U(n+1;\nu) = p\,U(n;\nu);$$

$$V(n+1;\nu) = p\,V(n;\nu);$$

$$W(n+1;\nu) = p\,W(n;\nu), \quad (5)$$

where $\nu=0,\cdots,n-1$, and $n=1,2,\cdots$.

Let $\bar{s}=s, s_{n+1}$ where s is any n-vector. Since s_{n+1} takes on p values, the LFSR Algorithm encounters p copies of s in processing $\bar{s}$. Hence, by summing over all possible $\bar{s}$, we verify the truth of Eqs. (5). Equation (4a) is true because, for each s, the associated LFSR computes an s_{n+1} for which Eq. (1) holds. The remaining $\bar{p}\,p^n$ cases, at step $\nu=n$, divide according to the length distribution for vectors s of length n; this is the meaning of Eqs. (4b, c).

Now define

$$SU(n) = \sum_{\nu=0}^{n-1} U(n;\nu)$$

and similarly for $SV(n)$ and $SW(n)$. Summing (5a, b, c) and then adding them to (4a, b, c), we get

$$SU(n+1) = p\,SU(n) + p^n;$$

$$SV(n+1) = p\,SV(n) + (p-1)\frac{p^{2\beta}-1}{p+1};$$

$$SW(n+1) = p\,SW(n) + (p-1)\frac{p^{2\alpha+1}+1}{p+1}, \quad (6)$$

for $n=1,2,\cdots$.

Equations (4), (5), and (6) can be solved either by induction or by using difference equation methods. We note that Eqs. (7) and (8), and also Eqs. (12) through (15) which follow, require a large amount of tedious, straightforward algebra on polynomials in the variable p. In doing the calculations we were aided immeasurably by the SCRATCHPAD system [5]. Each of the expressions contained in these equations represents a polynomial even though it is represented in part as the quotient of two polynomials, a compact form that allows one to easily see asymptotic values.

The solutions to equations (4), (5), and (6) are

$$U(n+1;\nu) = p^n;$$

$$V(n+1;\nu) = \frac{p^{n-\nu}(p-1)(p^{2\beta}-1)}{(p+1)};$$

$$W(n+1;\nu) = \frac{p^{n-\nu}(p-1)(p^{2\alpha+1}+1)}{(p+1)}, \quad (7)$$

where $\alpha=\lfloor\nu/2\rfloor$, $\beta=\lceil\nu/2\rceil$, $\nu=0,\cdots,n$, and $n=0,1,\cdots$; also

$$SU(n) = n\,p^{n-1};$$

$$SV(n) = \alpha\,p^{n-1}(p-1) - \frac{p^{2\alpha}-1}{p+1};$$

$$SW(n) = \beta\,p^{n+1}(p-1) + \frac{p^{2\alpha}-1}{p+1}, \quad (8)$$

where $\alpha=\lfloor n/2\rfloor$, $\beta=\lceil n/2\rceil$, and $n=1,2,\cdots$. Note that $[V(n+1;\nu)+W(n+1;\nu)]/U(n+1;\nu)=(p-1)$ so that, on the average, we correct the LFSR Algorithm $p-1$ out of p times at each iteration step. In the Boolean case ($p=2$) we get $d=0$ (in step 3) 50 percent of the time. The total polynomial count TPC is the sum of $n\,p^n$, SV, and SW. We get $TPC(n)=n(2p^n-p^{n-1})$. This indicates that an average input data string s requires $n(2-p^{-1})$ polynomial evaluations. The total cost of computing bd^{-1} in steps 4 and 5 is $SV(n)+SW(n)=p^n-p^{n-1}$ multiplications and the same number of divisions.

Number of multiplications required

Here we establish the multiplication count (which equals the addition count) for performing the polynomial operations in steps 2, 4, and 5. Analogous to Eqs. (4) and (5) we have recurrence relations

$$MC2(n+1;n) = p\sum_{\nu=0}^{n}\nu\,D(n;\nu);$$

$$MC4(n+1;n) = (p-1)\sum_{\nu=\alpha+1}^{n}\nu\,D(n;\nu);$$

$$MC5(n+1;n) = (p-1)\sum_{\nu=0}^{\alpha}(n+1-\nu)D(n;\nu), \quad (9)$$

where $\alpha=\lfloor n/2\rfloor$ and $n=0,1,2,\cdots$; $MC2$, $MC4$, and $MC5$ stand for multiplication count steps 2, 4, and 5 summed over all possible p^n formulas:

$$MC2(n+1;\nu) = p\,MC2(n;\nu);$$

$$MC4(n+1;\nu) = p\,MC4(n;\nu);$$

$$MC5(n+1;\nu) = p\,MC5(n;\nu), \quad (10)$$

where $\nu=0,\cdots,n-1$ and $n=1,2,\cdots$.

To verify Eqs. (9) and (10), let $\bar{s} = s, s_{n+1}$ where s is any input string of length n. Equations (10) are true in the same way as Eqs. (5). At iteration step $n+1$, input string s is processed p times. The multiplication count for a given s is $l(s)$ and, summing over all input strings s, we obtain $\Sigma_{\nu=0}^{n} \nu\, D(n;\nu)$ since there are $D(n,\nu)$ formulas of length ν. This verifies Eq. (9a).

We still have to correct the LFSR Algorithm for a given $\bar{s}$, with s fixed, $p-1$ times since the LFSR Algorithm for s correctly computes s_{n+1} only once. We choose step 4 or step 5 according to whether $2l(s) > n$ or $2l(s) \leq n$. Summing over all s, we get $\Sigma_{\nu=\alpha+1}^{n} \nu\, D(n;\nu)$ and $\Sigma_{\nu=0}^{\alpha} \nu\, D(n;\nu)$, where $\alpha = \lfloor n/2 \rfloor$, as the total multiplication counts for steps 4 and 5. This verifies Eqs. (9b, c).

We define

$$SM2(n) = \sum_{\nu=0}^{n-1} MC2(n;\nu)$$

and similarly for $SM4(n)$ and $SM5(n)$. Summing Eqs. (10) over ν and then adding them to Eq. (9), we get

$$SM2(n+1) = p\, SM2(n) + MC2(n+1;n);$$

$$SM4(n+1) = p\, SM4(n) + MC4(n+1;n);$$

$$SM5(n+1) = p\, SM5(n) + MC5(n+1;n), \qquad (11)$$

for $n = 1, 2, \cdots$. Equations (9), (10), and (11) can be solved in the same manner as Eqs. (4), (5), and (6); their solutions are

$$MC2(n+1;\nu) = p^{n-\nu}\left[\beta p^{\nu+1} - \frac{(-p)^{\nu+2} + (\nu+1)p^2 + \nu p}{(p+1)^2}\right];$$

$$MC4(n+1;\nu) = p^{n-\nu}\left[(\alpha+1)p^{2\beta} - \frac{2(\alpha+1)p^{2\beta+1} + (2\alpha+1)p^{2\beta} + (\nu+1)p^2 - \nu}{(p+1)^2}\right];$$

$$MC5(n+1;\nu) = p^{n-\nu}\left[(\beta+1)p^{2\alpha+1} - \frac{2(\beta+1)p^{2\alpha+2} + (2\beta+1)p^{2\alpha} - (\nu+1)p^2 + p + (\nu+1)}{(p+1)^2}\right], \qquad (12)$$

where $\alpha = \lfloor \nu/2 \rfloor$, $\beta = \lceil \nu/2 \rceil$, $\nu = 0, \cdots, n$, and $n = 0, 1, \cdots$; also

$$SM2(n) = \begin{cases} k\,p^n - p\left[\dfrac{p^{n+2} + p^n - (n+1)p^2 + n - 1}{(p^2-1)^2}\right]; \\ k(k+1)p^n - p\left[\dfrac{2p^{n+1} - (n+1)p + n - 1}{(p^2-1)^2}\right]; \end{cases}$$

$$SM4(n) = \begin{cases} (p-1)\left[\dfrac{k(k+1)}{2}\,p^{n-1} + \dfrac{-p^{n+2} + (k-1)p^{n+1} - p^n - kp^{n-1}}{(p^2-1)^2} + \dfrac{(n+1)p^2 + p + 1 - n}{(p^2-1)^2}\right]; \\ (p-1)\left[\dfrac{k(k+1)}{2}\,p^{n-1} + \dfrac{kp^{n+2} - 2p^{n+1} - (k+1)p^n}{(p^2-1)^2} + \dfrac{(n+1)p^2 + p + 1 - n}{(p^2-1)^2}\right]; \end{cases}$$

$$SM5(n) = \begin{cases} (p-1)\left[\dfrac{k(k+1)}{2}\,p^{n-1} + \dfrac{(k+1)p^{n+2} - kp^n - (n+1)p^2 + n}{(p^2-1)^2}\right]; \\ (p-1)\left[\dfrac{(k+1)(k+2)}{2}\,p^{n-1} + \dfrac{(k+2)p^{n+1} - (k+1)p^{n-1}}{(p^2-1)^2} - \dfrac{(n+1)p^2 - n}{(p^2-1)^2}\right], \end{cases} \qquad (13)$$

where $n = 2k$ is the condition for using the first expression in each pair and $n = 2k+1$, for the second.

Next, we sum Eqs. (12) to find the total multiplication count at iteration step ν, $\nu = 0, 1, \cdots, n$:

$$TMC(n+1;\nu) = \begin{cases} p^{n-\nu}\left[(\nu+1)p^{\nu+1} - \dfrac{kp^{\nu+2} + (\nu+1)p^{\nu+1} + kp^{\nu}}{(p+1)^2} - \dfrac{(\nu+1)p^2 + (\nu+1)p + 1}{(p+1)^2}\right], \nu = 2k; \\ p^{n-\nu}\left[(\nu+1)p^{\nu+1} - \dfrac{(k+1)p^{\nu+2} + \nu p^{\nu+1} + (k+1)p^{\nu}}{(p+1)^2} - \dfrac{(\nu+1)p^2 + (\nu+1)p + 1}{(p+1)^2}\right], \nu = 2k+1. \end{cases} \qquad (14)$$

Finally, the multiplication count of the LFSR Algorithm is the sum from $\nu = 0$ to $n-1$ of Eq. (14) or, equivalently, the sum of Eqs. (13):

$$MC(n) = \begin{cases} \dfrac{n(n+1)}{2}p^n - (k^2+1)p^{n-1} \\ \quad + \dfrac{-p^{n+2} - 3p^n + p^n + p^{n-1} + (n+1)p^3}{(p^2-1)^2} \\ \quad + \dfrac{p^2 - (n-1)p - 1}{(p^2-1)^2}, \quad n = 2k; \\ \dfrac{n(n+1)}{2}p^n - (k^2+k+1)p^{n-1} \\ \quad + \dfrac{-2p^{n+2} - p^{n+1} + p^{n-1} + (n+1)p^3}{(p^2-1)^2} \\ \quad + \dfrac{p^2 - (n-1)p - 1}{(p^2-1)^2}, \quad n = 2k+1. \end{cases} \tag{15}$$

From (15) we see that the average multiplication count for the LFSR Algorithm is approximately $\frac{1}{2}n(n+1) - \frac{1}{4}n^2p^{-1}$. Since one does about $n(2 - p^{-1})$ polynomial evaluations, the average polynomial evaluation requires $\frac{1}{4}n$ multiplications (and additions). Table 2 lists the multiplication counts for the Boolean case ($p = 2$) for input strings up to length ten.

• *Minimum cost*

The string $s = 0$ constitutes the only input data that minimize the computation cost of the LFSR Algorithm. Clearly, $L_n(s) = 0$ for all n when $s = 0$. Hence, steps 4 and 5 are never entered and there is no computation for these steps. Step 2 reduces to the value assignment $d = s_{k+1}$ for $k = 0, \cdots, n-1$ and, as we indicated previously, we do not count this operation.

• *Maximum cost*

There are several digit strings that produce a maximum multiplication count of $\frac{1}{2}n(n+1)$. To show this we give an induction proof that is more complicated than one might suspect because in going from strings of length n to length $n+1$, a maximal string s can increase its cost only by $n+1$ whereas other strings can increase their costs by as much as $2(n+1)$. It turns out that maxima for odd length input strings can occur at more than one value. To overcome this difficulty we need a slightly more complicated induction hypothesis, as follows.

Partition the p^n strings of length n into $n+1$ groups; members of group l require a minimum of $L_n(s) = l$ constants to generate them ($l = 0, 1, \cdots, n$). The maximum multiplication count for group l is

$$MMC(n; l) = \begin{cases} l(n+1), & l = 0, \cdots, \alpha; \\ l(2n+1-2l), & l = \alpha+1, \cdots, n, \end{cases} \tag{16}$$

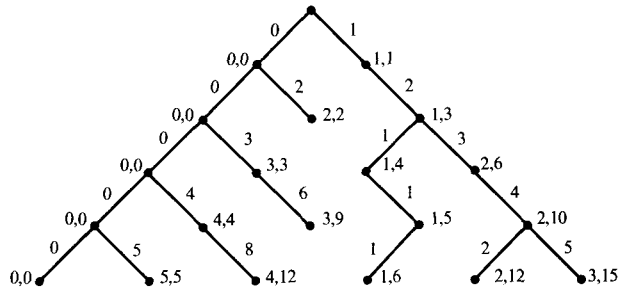

Figure 1 Tree diagram showing the maximum multiplication counts (MMC) for the LFSR Synthesis Algorithm; $p = 2$.

where $\alpha = \lfloor n/2 \rfloor$. When n is even (odd), the maximum occurs for $l = \alpha\ (\alpha+1)$; the maximum value is $\frac{1}{2}n(n+1)$ for both cases.

Figure 1 is a tree diagram that illustrates this induction hypothesis. Nodes are labeled by a pair of numbers m, n, where m equals the number of constants needed and n equals the maximum multiplication count. The edge label is the increase in multiplication count in going from iteration step ν to step $\nu+1$. Any path from the root node consisting of ν edges reaches iteration step ν. The set of nodes whose distance is n from the root constitutes the $n+1$ group maxima; we say these nodes are at level n. Node 2, 2 illustrates the difficulty: During iteration step 3, its multiplication count can increase by four (steps 2 and 4 are entered and each contributes a count of two) and thus it merges with node 2, 6. A similar remark holds for node 3, 9 at iteration step 4.

We now establish the induction result for MMC. Suppose formula (16) holds for $n = k$. Nodes at level k with $m = \nu$ and $2\nu \leq k$ experience no length change if d in step 3 is equal to zero. In this case, only step 2 is executed and it uses ν multiplications. Hence $MMC(k+1; \nu) = \nu(k+2)$ for $0 \leq 2\nu \leq k$. Nodes for which $2\nu > k$ experience no length change but experience a maximum change in multiplication count when $d \neq 0$ in step 3; i.e., steps 2 and 4 are executed at a cost of 2ν. However, nodes where $2\nu \leq k$ become members of the class $2\nu > k$ by experiencing a length change; step 3 then has $d \neq 0$ and the increase in count is $\nu + k + 1 - \nu$ as steps 2 and 5 are entered. Thus, $MMC(k+1; k+1-\nu) = \max[A(\nu), B(\nu)]$, $\nu = 0, \cdots, \alpha$, where $A(\nu) = (2\nu+1)(k+1-\nu)$ and $B(\nu) = (\nu+1)(k+1)$. [The domain of $A(\nu)$ is actually $1 \leq \nu \leq \alpha$. However, evaluate A at $\nu = 0$ and note that $k+1 = A(0) = B(0)$; hence this result is valid.] Since $A(\nu) - B(\nu) = \nu(k-2\nu)$ and is non-negative for $\nu = 0, \cdots, \alpha$, we conclude that $MMC(k+1; k+1-\nu) = A(\nu)$.

Replacing $k+1-\nu$ by ν, we get $MMC(k+1; \nu) = \nu(2k-2\nu+3)$, $\nu = k+1-\alpha, \cdots, k+1$. When k is odd, $1+\alpha < k+1-\alpha$ and we must consider the case $\nu = \alpha+1$. Here, $2(\alpha+1) > k$ and $MMC(k+1; \alpha+1) = A(\alpha+1) =$

Table 2 Multiplication counts for the LFSR Synthesis Algorithm; $p = 2$.

	ν										
n	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*Sum*
	$MC2(n; \nu)$										$SM2(n)$
1	0										0
2	0	2									2
3	0	4	8								12
4	0	8	16	26							50
5	0	16	32	52	68						168
6	0	32	64	104	136	174					510
7	0	64	128	208	272	348	408				1428
8	0	128	256	416	544	696	816	962			3818
9	0	256	512	832	1088	1392	1632	1924	2156		9792
10	0	512	1024	1664	2176	2784	3264	3848	4312	4886	24470
	$MC4(n; \nu)$										$SM4(n)$
1	0										0
2	0	1									1
3	0	2	2								4
4	0	4	4	11							19
5	0	8	8	22	16						54
6	0	16	16	44	32	69					177
7	0	32	32	88	64	138	90				444
8	0	64	64	176	128	276	180	367			1255
9	0	128	128	352	256	552	360	734	452		2962
10	0	256	256	704	512	1104	720	1468	904	1817	7741
	$MC5(n; \nu)$										$SM5(n)$
1	1										1
2	2	2									4
3	4	4	7								15
4	8	8	14	10							40
5	16	16	20	20	37						117
6	32	32	56	40	74	48					282
7	64	64	112	80	148	96	187				751
8	128	128	224	160	296	192	374	230			1732
9	256	256	448	320	592	384	748	460	913		4377
10	512	512	896	640	1184	768	1496	920	1826	1084	9838
	$TMC(n; \nu)$										$MC(n)$
1	1										1
2	2	5									7
3	4	10	17								31
4	8	20	34	47							109
5	16	40	68	94	121						339
6	32	80	136	188	242	291					969
7	64	160	272	376	484	582	685				2623
8	128	320	544	752	968	1164	1370	1559			6805
9	256	640	1088	1504	1936	2328	2740	3118	3521		17131
10	512	1280	2176	3008	3872	4656	5480	6236	7042	7787	42049

$(\alpha + 1)(2\alpha + 3) = \alpha'(k+2)$, where $\alpha' = \lfloor \frac{1}{2}(k+1) \rfloor$. Thus we have

$$MMC(k+1; \nu) = \begin{cases} \nu(k+2), & \nu = 0, 1, \cdots, \alpha'; \\ \nu(2k - 2\nu + 3), & \\ & \nu = \alpha' + 1, \cdots, k+1. \end{cases}$$

Our induction becomes complete by noting that $MMC(1; 0) = 0$ and $MMC(1; 1) = 1$; hence, the induction result holds for $n = 1$. The difference between the maximum and the average multiplication counts is approximately $\frac{1}{4}n^2\, p^{-1}$. As $p \to \infty$ (real case), the average and the maximum multiplication counts are the same.

When $p = 2$ (binary case) the average is about three-fourths of the maximum count. These cases are the extremes, and other finite fields yield ratios between three-fourths and one. Finally, note that Eqs. (3) and (16) complement each other; they indicate, respectively, the number of input strings and the maximum multiplication count for each of the $n + 1$ groups. Table 3 contains the maximum multiplication counts for each group for input strings up to length ten.

Table 3 Maximum multiplication counts for the LFSR Synthesis Algorithm; $p = 2$.

	l										
n	0	1	2	3	4	5	6	7	8	9	10
1	0	1									
2	0	3	2								
3	0	4	6	3							
4	0	5	10	9	4						
5	0	6	12	15	12	5					
6	0	7	14	21	20	15	6				
7	0	8	16	24	28	25	18	7			
8	0	9	18	27	36	35	30	21	8		
9	0	10	20	30	40	45	42	35	24	9	
10	0	11	22	33	44	55	54	49	40	27	10

Comments on the BOIC scheme

The compression stage BOIC scheme consists of four steps [1, p. 143]. We shall show that the computation in step 1 contains the answer, α, that they seek; this obviates the need for steps 2, 3, and 4. Step 1 is to be compared with the LFSR Algorithm. Since step 1 is $\mathcal{O}(n^3)$ (Gaussian elimination on l rows of length about $n - l$), we think the LFSR Algorithm should always be used. Berlekamp, in his book [3, Chapter 7, Section 5] discusses the relation between the LFSR Algorithm and matrices of the BOIC type. Because these matrices have a special structure ($m_{ij} = s_{i+j-1}$), it is possible to solve equations in less than $\mathcal{O}(n^3)$ operations; e.g., the LFSR Algorithm does the job in $\mathcal{O}(n^2)$. It is an open question as to whether another algorithm could find the $L_n(s)$ constants in less than $\mathcal{O}(n^2)$. Berlekamp also demonstrates a relation between $L_n(s)$ and the singularity of its associated matrix [3, Theorem 7.51, p. 189]; see the matrix A [1, p. 142. Eq. (7)]. The singularity of the matrix A for all $l \leq \lfloor n/2 \rfloor$ means the BOIC scheme causes data expansion for the given input. For $l > \lfloor n/2 \rfloor$ the matrix A contains x's (don't-care elements); nonetheless, matrix A can be singular regardless of the values the x's assume. An extreme case is given when the input string is $0, \cdots, 0, 1$; matrix A is then non-singular only if $l = n$.

Let $A_{\nu-1}$ be the n-component vector $a_\nu, a_{\nu+1}, \cdots, a_n, x, \cdots, x$. In what follows we use the notation of [1]. The reader can identify A_0 with the input string $s = s_1, \cdots, s_n$. The following result is true: $A_l = \sum_{\nu=1}^{l} \alpha_\nu A_{\nu-1}$; i.e., the linear combination of blocks that annihilate A_l are those blocks with $\alpha_\nu = 1$.

Proof It follows from Eq. (1) of [1], i.e.,

$$
\begin{aligned}
A_0 &= a_1a_2a_3a_4a_5a_6a_7\cdots a_n;\\
A_1 &= a_2a_3a_4a_5a_6a_7\cdots a_nx;\\
A_2 &= a_3a_4a_5a_6a_7\cdots a_nxx;\\
&\vdots\\
A_{n-1} &= a_nxxx\cdots xxx,
\end{aligned}
$$

that if $A_l = c_1A_{l-1} + \cdots + c_lA_0$, then $a_{l+\nu} = c_1a_{l+\nu-1} + \cdots + c_la_\nu$ for $\nu = 1, 2, \cdots, n - l$, as one sees immediately from the first $n - l$ columns of the first $l + 1$ rows of this equation. Hence [1, p. 142, Eq. (12)], $\alpha_\nu = c_{l+1-\nu}$, $\nu = 1, 2, \cdots, l$; this establishes the result and also displays the relation between the LFSR c's and the BOIC α's.

We now derive the number of additions needed to perform step 1. In doing so we include only the cost of adding rows to other rows to produce an upper triangular matrix form. At stage $\nu + 1$, which zeros out row $\nu + 1$ up to the diagonal, we do $n + 1 - \nu - \sigma$ additions with rows σ, $\sigma = 1, \cdots, \nu$. This uses $\frac{1}{2}\nu(2n + 1 - 3\nu)$ additions. The total cost (addition count) of processing rows 2 through $l + 1$ is

$$AC(n; \nu) = \tfrac{1}{2}l(l + 1)(n - l), \qquad l = 0, \cdots, \lfloor n/2 \rfloor. \quad (17)$$

An input string A_0, whose associated set of constants α is of length l, requires processing up to row $l + 1$ via the BOIC scheme. Hence, by subtracting MMC from AC we can determine a bound on the extra computation required by step 1. Remember that for the LFSR Algorithm, the addition count equals the multiplication count; also, in the Boolean case, multiplications can be avoided in steps 4 and 5. Thus we obtain

$$AC(n; \nu) - MMC(n; \nu) = l[\tfrac{1}{2}(l - 1)(n - l - 2) - 2], \qquad l = 0, \cdots, \lfloor n/2 \rfloor. \quad (18)$$

Inspection of (18) shows that the LFSR Algorithm is almost always superior; however, we have not given a complete analysis.

Summary

This paper supplements Massey's paper [2] and Chapter 7 of Berlekamp's book [3]. Our principal results are the derivation of the length distribution of the constants in the LFSR Algorithm and the analysis of its computation cost. They show that on the average, the LFSR Algorithm cannot be used as a data reduction method. However, it was found that the LFSR Algorithm is superior to the BOIC scheme [1], which does not exploit the

special character of its matrix in step 1. Furthermore, the BOIC scheme cannot, without additional information about the input strings, be used for information compression.

Acknowledgments

The author thanks L. R. Bahl, J. H. Mommens, and one of the referees for suggestions that improved the clarity of this presentation.

References

1. F. P. Palermo and H. Ling "Block-oriented Information Compression," *IBM J. Res. Develop.* **19**, 141 (1975).
2. J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Information Theory* **IT-15**, 122 (1969).
3. E. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Book Co., Inc., New York, 1968, Chapter 7.
4. R. G. Gallager, *Information Theory and Reliable Communication*, John Wiley & Sons, Inc., New York, 1968, Chapter 3, p. 43.
5. J. H. Griesmer and R. D. Jenks, "Experience with an Online Symbolic Mathematics System," *Proceedings of the ONLINE72 Conference*, Brunel University, Uxbridge, Middlesex, England, September 1972, Vol. 1, pp. 457-476; also available as "The SCRATCHPAD System," *Research Report RC 3925*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York.

Received August 22, 1975

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.