H. Ling
F. P. Palermo

Block-oriented Information Compression

Abstract: Data base statistics play an important role in conventional information compression. For a large data base, the acquisition of data base statistics becomes a very difficult task. This paper presents a new scheme for information compression that does not use information statistics. Each information block is represented by two sub-blocks called the *alphabet* and the *generator*. The *alphabet* contains the linearly independent elements; the *generator* is computed through the linear combination of the linearly dependent elements. The total length of these two sub-blocks is generally shorter (never greater) than the original block.

Introduction

The underutilization of CPUs and the existence of redundant data are well-known phenomena that will become more profound when the computer shares more non-computing loads such as data management and non-coded information handling.

In 1952 Huffman presented a "two-pass" scheme to minimize the average number of coding digits per message. [1]. In the first pass the statistics of the message occurrence frequency are collected through scanning. During the second pass, the shortest code is used to represent the information with the highest occurrence frequency. A minimum redundant code is introduced. Unfortunately, in a large data base, the time required to collect the data base statistics could limit the usefulness of this approach.

Recently Raviv [2] has proposed a sampling technique for obtaining statistics of the message occurrence frequency. For a large information collection, the statistics vary from file to file; sampling is time consuming. To compress the information in a large data base of on-line operation, a one-pass algorithm appears to be more efficient. This method is used to compress the information without knowing the complete data base statistics or any characteristics of future incoming information.

The theory is presented first, followed by a consideration of the implementation procedures, the compression and reconstruction of binary numbers, and lastly some logical descriptions. To show the step-by-step operation, an example is included. The possible application of the method is also discussed.

Theory

In this paper we present a technique to achieve information compression by using a shift register. Let A_0 be the information block $a_1a_2\cdots a_n$ to be compressed. A new block of information is generated each time A_0 is successively shifted to the left. After the $(n-1)$th shift we have obtained n blocks of information:

$$\begin{aligned} A_0 &= a_1a_2a_3a_4a_5a_6a_7\cdots a_n \\ A_1 &= a_2a_3a_4a_5a_6a_7\cdots a_n\mathrm{X} \\ A_2 &= a_3a_4a_5a_6a_7\cdots a_n\mathrm{XX} \\ &\vdots \\ A_{n-1} &= a_n\mathrm{XXXX}\cdots\mathrm{XXX}, \end{aligned} \tag{1}$$

where X of Eq. (1) represents the *don't care* case.

When the first linearly dependent block is obtained, the remaining blocks are also linearly dependent.

The linearly dependent block of Eq. (1) can be eliminated by using the row-reducing method [3]. For example, if block l is equal to the linear combination of blocks i and j, block l is eliminated. Blocks $l+1$ through A_{n-1} are obtained through the shifting of block l. Therefore, when block l is eliminated, the successive blocks are also eliminated. After filtering out the linearly dependent blocks, the linearly independent blocks are as shown below:

$$\begin{aligned} A_0 &= a_1a_2a_3a_4a_5a_6a_7\cdots a_n \\ A_1 &= a_2a_3a_4a_5a_6a_7\cdots a_n\mathrm{X} \\ A_2 &= a_3a_4a_5a_6a_7\cdots a_n\mathrm{XX} \\ &\vdots \\ A_{l-1} &= a_la_{l+1}\cdots a_n\mathrm{X}\cdots\mathrm{XX}. \end{aligned} \tag{2}$$

By extracting the leading $l+1$ digit from each block of Eq. (2), we obtain a standard array:

$$\begin{aligned} &a_1a_2a_3a_4a_5a_6a_7\cdots a_{l+1} \\ &a_2a_3a_4a_5a_6a_7a_8\cdots a_{l+2} \\ &a_3a_4a_5a_6a_7a_8a_9\cdots a_{l+3} \\ &\vdots \\ &a_{l+1}a_{l+2}\cdots\quad a_{l+l+1}. \end{aligned} \tag{3}$$

We consider first the lth component of each row of Eq. (3) as a member of the matrix **M**:

$$\mathbf{M} = \begin{vmatrix} a_1 a_2 a_3 \cdots a_l \\ a_2 a_3 a_4 \cdots a_{l+1} \\ a_3 a_4 a_5 \cdots a_{l+2} \\ \vdots \\ a_l a_{l+1} \cdots a_{l+l-1} \end{vmatrix}. \tag{4}$$

M may contain more elements than A_0. To condense these elements into finite terms, we introduce a new matrix **F** defined by

$$\mathbf{F} \times \mathbf{M} = \mathbf{I}, \tag{5}$$

where **I** is the identity matrix. **M** can be written as

$$\mathbf{M} = \mathbf{F}^{-1}\mathbf{I}. \tag{6}$$

Equation (6) shows that if **M** is a symmetric matrix, **F** must be another symmetric matrix. Therefore, if f_{ij} represents the element of the ith row and jth column, from (6) we have $f_{ij} = f_{ji}$.

Let us consider every element of the standard array as a member of matrix **A**. We then have

$$\mathbf{A} = \begin{vmatrix} a_1 a_2 a_3 \cdots a_l & a_{l+1} \\ a_2 a_3 a_4 \cdots a_{l+1} & a_{l+2} \\ a_3 a_4 a_5 \cdots a_{l+2} & a_{l+3} \\ \vdots \\ a_l a_{l+1} \cdots a_{l+l-1} a_{l+l} \end{vmatrix}. \tag{7}$$

By multiplying **F** and **A**, we obtain

$$\mathbf{F} \times \mathbf{A} = \begin{vmatrix} 1\ 0\ 0\ 0\ 0 \cdots 0 & \alpha_1 \\ 0\ 1\ 0\ 0\ 0 \cdots 0 & \alpha_2 \\ 0\ 0\ 1\ 0\ 0 \cdots 0 & \alpha_3 \\ 0\ 0\ 0\ 1\ 0 \cdots 0 & \alpha_4 \\ \vdots \\ 0\ 0\ 0\ 0\ 0 \cdots 1 & \alpha_l \end{vmatrix}. \tag{8}$$

Equation (8) shows that **M** is condensed into an identity matrix and all the elements beyond a_l are compressed into elements $\alpha_1\alpha_2 \cdots \alpha_l$. Therefore, an information block of n digits can be represented by two sub-blocks $a_1a_2 \cdots a_l$ and $\alpha_1\alpha_2 \cdots \alpha_l$. We call the first block $a_1a_2 \cdots a_l$ the *alphabet*, and the second block $\alpha_1\alpha_2 \cdots \alpha_l$ the *generator*. When A_0 is compressible, l is less than $n/2$.

To compute the α's, each component of **F** has to be calculated first. From (8) we have

$$\begin{aligned}
&f_{11}a_1 + f_{12}a_2 + f_{13}a_3 + \cdots + f_{1l}a_l = 1 \\
&f_{11}a_2 + f_{12}a_3 + f_{13}a_4 + \cdots + f_{1l}a_{l+1} = 0 \\
&\quad\vdots \\
&f_{11}a_l + f_{12}a_{l+1} \qquad\quad \cdots + f_{1l}a_{2l-1} = 0 \\
&\quad\vdots \\
&f_{21}a_1 + f_{22}a_2 + f_{23}a_3 + \cdots + f_{2l}a_l = 0 \\
&f_{21}a_2 + f_{22}a_3 + f_{23}a_4 + \cdots + f_{2l}a_{l+1} = 1 \\
&\quad\vdots \\
&f_{21}a_l + f_{22}a_{l+1} \qquad\quad + \cdots + f_{2l}a_{2l-1} = 0 \\
&\quad\vdots \\
&f_{l1}a_1 + f_{l2}a_2 + f_{l3}a_3 + \cdots + f_{ll}a_l = 0 \\
&f_{l1}a_2 + f_{l2}a_3 + f_{l3}a_4 + \cdots + f_{ll}a_{l+1} = 0 \\
&\quad\vdots \\
&f_{l1}a_l + f_{l2}a_{l+1} \qquad\quad + \cdots + f_{ll}a_{2l-1} = 1.
\end{aligned} \tag{9}$$

Because of the symmetry of **F**, the total number of computations is reduced. The values of α can be written as

$$f_{11}a_{l+1} + f_{12}a_{l+2} + \cdots + f_{1l}a_{2l} = \alpha_1 \tag{10-1}$$

$$f_{21}a_{l+1} + f_{22}a_{l+2} + \cdots + f_{2l}a_{2l} = \alpha_2 \tag{10-2}$$

$$\vdots \qquad\qquad \vdots$$

$$f_{l1}a_{l+1} + f_{l2}a_{l+2} + \cdots + f_{ll}a_{2l} = \alpha_l. \tag{10-l}$$

Equations (10) show that the linearly dependent elements can be represented by $\alpha_1\alpha_2 \cdots \alpha_l$. Hence an information compression is obtained.

The reconstruction process can be divided into the following steps:

Step 1 Multiply Eq. (10-1) by a_1, (10-2) by a_2, $\cdots$, (10-l) by a_l; then the summation of these products gives

$$a_{l+1} = a_1\alpha_1 + a_2\alpha_2 + \cdots + a_l\alpha_l. \tag{11-1}$$

Step 2 Multiply Eq. (10-1) by a_2, (10-2) by a_3, $\cdots$, (10-l) by a_{l+1}; then the summation of products, as in step 1, gives

$$a_{l+2} = a_2\alpha_1 + a_3\alpha_2 + \cdots + a_{l+1}\alpha_l. \tag{11-2}$$

In a similar way we obtain all other steps so that the final step l is obtained.

Step l Multiply Eq. (10-1) by a_l, (10-2) by a_{l+1}, $\cdots$, (10-l) by a_{2l-1}; then the summation of these products gives

$$a_{2l} = a_l\alpha_1 + a_{l+1}\alpha_2 + \cdots + a_{2l-1}\alpha_l. \tag{11-l}$$

The general term for Eqs. (11-1) through (11-l) can be written as

$$a_{l+p} = a_p\alpha_1 + a_{p+1}\alpha_2 + \cdots + a_{l+p-1}\alpha_l, \tag{12}$$

and it is applicable when generating the digit beyond a_{2l}. By letting $p = l + 1$, we have

$$a_{2l+1} = a_{l+1}\alpha_1 + a_{l+2}\alpha_2 + \cdots + a_{2l}\alpha_l. \tag{13}$$

Using Eqs. (11), (12), and (13), we can reconstruct the original information block $a_1a_2 \cdots a_n$.

Implementation

We have presented an algorithm that enables us to obtain the information compression via computation. The idle CPU time can be used to compress the linearly dependent elements. The reconstruction process is rather simple; Eq. (12) shows that a shift register with feedback is sufficient for reconstruction. The additions in Eqs. (1) through (13) can be further simplified by replacing them with modulo two addition.

• *Compression*

The process of compression can be described in four steps as shown in Fig. 1.

Step 1 The search for l can be accomplished by the EXCLUSIVE OR operation, adding block i to block k until the null block is obtained. The value of l is equal to the value of k.

Step 2 Equation (9) shows that f_{ij} can be written as

$$f_{ij} = \Delta_{ij}/\Delta, \tag{14}$$

where Δ is the determinant of (9) and Δ_{ij} is the cofactor. The determinant Δ contains all the elements of the linearly dependent blocks. Using the EXCLUSIVE OR operation on both row and column, we can transform Δ into a new determinant with all the elements along the main diagonal equal to one and all other elements equal to zero. Therefore, Δ has the value 1, and the computation of Δ is eliminated. We have

$$f_{ij} = \Delta_{ij}; \tag{15}$$

Δ_{ij} can be a symmetric or a non-symmetric determinant. A non-symmetric Δ_{ij} is equal to one, except for those usual cases for which a determinant is zero (a row or column equals zero, or two rows or two columns are identical). When Δ_{ij} is symmetric, its value V_e can be written as

$$V_e = a_1'a_2 + a_1(a_2 \oplus a_3), \tag{16}$$

if there are three distinct elements in the determinant, and as

$$V_e = a_1'(a_3 \oplus a_2a_5) + a_1[a_2'(a_4 \oplus a_3a_5') + a_2(a_4' \oplus a_3'a_5')], \tag{17}$$

if there are five distinct elements. The $\oplus$ symbol represents the EXCLUSIVE OR operation, and the primed factors indicate the complements of their respective values. A general expression can be derived in a similar manner for more distinct elements.

Step 3 The implementation of α_j is shown in Fig. 2, which is a general representation for Eqs. (10-1) through (10-l). When $f_{j1} \cdots f_{jl}$ are available, α_j is implemented through a two-level logic. When the number of inputs to the EXCLUSIVE OR gate increases, the number of input levels may increase due to the *fan-in* limitation.

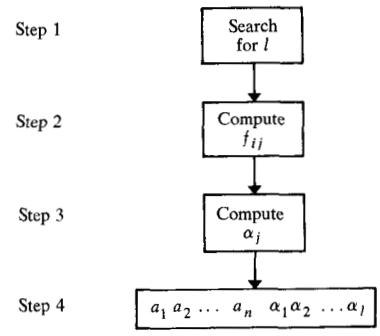

Figure 1 Compression procedure.

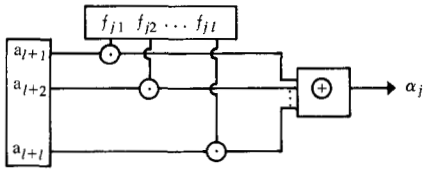

Figure 2 The logical implementation of α_j.

Step 4 The final compressed information is shown in the box below. The first row of **M** of Eq. (4) is concatenated with the α_j computed from Eqs. (10) or logically implemented as shown in Fig. 2.

$$a_1a_2 \cdots a_l \ \alpha_1\alpha_2 \cdots \alpha_l$$

• *Reconstruction*

Equation (12) shows that we can reconstruct the original information by using a simple "shift and add." The reconstruction procedure is shown in Fig. 3.

The blocks $a_1a_2 \cdots a_l$ and $\alpha_1\alpha_2 \cdots \alpha_l$ can be pre-recorded or received from the on-line operation. The output is the next digit to be generated. This particular digit a_{l+p} is fed back to the shift register R_S for generating the suceeding digit. The original information block is obtained from the output of the shift register R_S.

Detailed description

We now describe the operational procedure step-by-step, by means of an example. Let us consider

$$A_0 = 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0. \tag{18}$$

• *Compression*

Step 1 Find the linearly independent element of

1 0 1 1 1 0 1 0 0 1 1 1 0 1 0 0 1.

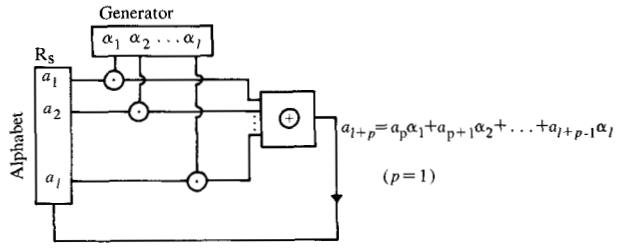

Figure 3 The reconstruction procedure.

From Eq. (1) we obtain the following information block by successive shifting of A_0 [Eqs. (19-1)-(19-15)]:

$A_1 = 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X$

$A_2 = 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X$

$A_3 = 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X$

$A_4 = 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X$

$A_5 = 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X$

$A_6 = 1\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X$

$A_7 = 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X$

$A_8 = 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X$

$A_9 = 1\ 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X\ X$

$A_{10} = 1\ 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X$

$A_{11} = 1\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X$

$A_{12} = 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X$

$A_{13} = 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X$

$A_{14} = 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X$

$A_{15} = 0\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X\ X$

As described in the previous section we obtain a null block by adding (19-1) and (19-2) to (19-4). Therefore, there are four linearly independent elements in A_0. We have $l=4$.

Step 2 By extracting the four leading bits from A_0 and Eqs. (19-1)-(19-3), we have

$$\mathbf{M} = \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{vmatrix}. \qquad (20)$$

From equation (15) we obtain

$f_{11} = \Delta_{11}$, a symmetric matrix; using (17), $\Delta_{11} = 1.$

$f_{12} = \Delta_{12}$ a symmetric matrix; using (17), $\Delta_{12} = 0.$

$f_{13} = \Delta_{13}$ not a symmetric matrix; $\Delta_{13} = 1.$

$f_{14} = \Delta_{14}$ a symmetric matrix; using (17), $\Delta_{14} = 1.$

$f_{21} = f_{12}.$

$f_{22} = \Delta_{12}$ a symmetric matrix; using (18), $\Delta_{22} = 1.$

$f_{23} = \Delta_{23}$ not a symmetric matrix; $\Delta_{23} = 1.$

$f_{24} = \Delta_{24}$ not a symmetric matrix; $\Delta_{24} = 1.$

$f_{31} = f_{13}.$

$f_{32} = f_{23},$

$f_{33} = \Delta_{33}$, a symmetrix matrix; using (18), $\Delta_{33} = 1.$

$f_{34} = \Delta_{34}$, a symmetrix matrix; using (18), $\Delta_{34} = 1.$

$f_{41} = f_{14}.$

$f_{42} = f_{24}.$

$f_{43} = f_{34}.$

$f_{44} = \Delta_{44}$, a symmetric matrix; using (18), $\Delta_{44} = 1.$

Step 3 From Fig. 2, we have the implementation shown in Fig. 4. By replacing f_{11}, f_{12}, f_{13}, and f_{14} with f_{21}, f_{22}, f_{23}, and f_{24}, respectively, we obtain $\alpha_2 = 1$. This process continues until all the α values are generated. Therefore,

$\alpha_1\alpha_2\alpha_3\alpha_4 \rightarrow 0\ 1\ 0\ 1.$

This result can also be obtained by using Eqs. (10-1)-(10-4).

Step 4 The compressed information is shown as

$a_1a_2a_3a_4\ \alpha_1\alpha_2\alpha_3\alpha_4$
1 0 1 1 0 1 0 1

;

We have compressed a 16-bit information block into an 8-bit block.

• *Reconstruction*

From Fig. 3, the first linearly dependent element a_5 is generated as shown in Fig. 5. After generation of a_5, a_1 is shifted out of register R_s; a_2, a_3, and a_4 are each shifted upward; and the newly determined a_5 is shifted into the position where a_4 had been. From this configuration a_6 can be determined. This process continues until all 16 digits are generated. The final digit, a_{16}, is obtained when a_{12}, a_{13}, a_{14} and a_{15} are located in R_s. This same result can also be obtained by using Eq. (12):

$a_5 = a_1\alpha_1 \oplus a_2\alpha_2 \oplus a_3\alpha_3 \oplus a_4\alpha_4 = 1$

$a_6 = a_2\alpha_1 \oplus a_3\alpha_2 \oplus a_4\alpha_3 \oplus a_5\alpha_4 = 0$

$a_7 = a_3\alpha_1 \oplus a_4\alpha_2 \oplus a_5\alpha_3 \oplus a_6\alpha_4 = 1$

$\vdots$

$a_{16} = a_{12}\alpha_1 \oplus a_{13}\alpha_2 \oplus a_{14}\alpha_3 \oplus a_{15}\alpha_4 = 0.$

The original information block 1 0 1 1 1 0 1 0 0 1 1 1 0 1 0 0 is reconstructed. In this example, we have shown the step-by-step procedure used in applying this method to compress and reconstruct the information

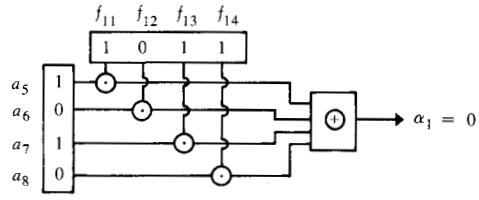

Figure 4 The logical implementation of α_j.

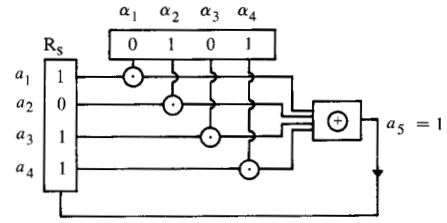

Figure 5 The reconstruction of the original information block.

block. In the compression stage, additional CPU time is required for the computation of l. The f_{ij} and α_j can be logically implemented in hardware or in software. In the reconstruction phase, a simple shift register with some logic as shown in Fig. 3 is sufficient to reconstruct the original information block.

Applications

It is commonly known that CPU utilization is less than 100 percent and that it can vary over wide ranges for different applications. This method can make use of some of the idle CPU time for information compression.

For non-coded data transmission, such as pictures, there exists much redundant information. The method described above can be used to compress this kind of redundant information on-line, and the bandwidth requirement for the transmission line can thereby be reduced. This method can also be used to reduce both storage space requirements and bandwidth requirements in computer networks.

In error correction, there are various codes used to correct multiple errors. Unfortunately, multiple-error correction codes are inefficient. For example, to transmit a 16-bit information block with single-error correcting capability [4], the total required code length is 24 bits. By using our method, the 16-bit information block can be compressed into an 8-bit block plus 4 bits to indicate that $l = 8$ for this block; for single-error correction capability, the total code length is thus reduced from 24 bits to 18 bits. This, of course, is true when the message contains the same degree of redundancy as used in the example.

Discussion

The technique we have used for information compression is based on mathematical system realization theory [5]. Massey, writing on the application of coding theory to shift-register synthesis and BCH decoding [6], has suggested that an additional application of coding theory would be in the area of data compression. By comparing Massey's work with our own, we see that there exists a link between coding theory and mathematical system theory in this area.

The method presented in this paper takes the input string and operates in a parallel fashion to obtain the alphabet and the generator. The total process requires l steps. Massey's method is to take the input string and iterate serially through n steps to obtain the feedback connection of the shift registers.

The coefficient of the connection polynomial in [6] is similar to Eq. (10) in our paper. Massey's coefficients are obtained by using the Berlekamp iterative algorithm [7], whereas our α's are obtained directly through matrix multiplication. The reader should be aware that the ratio of compression is a function of the degree of data redundancy. The 2:1 compression ratio obtained by compressing the data in Eq. (18) is only an illustrative example.

Summary

Unlike the conventional method, information statistics play no role in the information compression technique described. Because no statistical information on the data is available, the current or on-line information is considered to be relevant to information compression. This paper has presented an algorithm for such information compression and considered its implementation. The simple EXCLUSIVE OR operation and *shifting* are used in both compression and reconstruction. These instructions require very few machine cycles. In many cases, a simple logic array can be used in data base management to save storage space and, in on-line operation, to reduce the bandwidth requirements.

References

1. D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. I.R.E.* **40,** 1098 (1952).
2. J. Raviv, private communication.
3. G. Birkhoff and S. S. MacLane, *A Survey of Modern Algebra*, The Macmillian Co., New York, 1963, p. 176.
4. W. W. Peterson, *Error Correcting Codes*, John Wiley and Sons, Inc., New York, 1961, p. 71.
5. R. G. Kalman, T. L. Falb, and M. A. Arbib, *Topics in Mathematical System Theory*, McGraw-Hill Book Co., Inc., New York, 1969.
6. J. L. Massey, "Shift Register Synthesis and BCH Decoding," *IEEE Trans. Information Theory* **IT-15,** 122 (1969).
7. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Book Co., Inc., New York, 1968.

Received March 6, 1974

H. Ling is located at the IBM Data Processing Product Group Headquarters, White Plains, New York 10604. F. P. Palermo is located at the IBM Research Laboratory, Monterey and Cottle Roads, San Jose, California 95193.