Simple Algorithms for BCH Decoding

Jonathan Hong and Martin Vetterli, *Senior Member, IEEE*

***Abstract*— In this paper we propose some simple algorithms for decoding BCH codes. We show that the pruned FFT is an effective method for evaluating syndromes and for finding the roots of error-locator polynomials. We show that a simple variation of the basic Gaussian elimination procedure can be adapted to compute the error-locator polynomial efficiently for codes with small designed distance. Finally, we give a procedure for computing the error values that has half the complexity of the Forney algorithm.**

I. Introduction

BCH CODES are a subclass of cyclic codes in which the generator polynomials of the codes have as roots $2t$ consecutive powers of an element of order n. BCH codes are attractive because a) a lower bound on their minimum distance is known ($2t + 1$, hence t-error-correcting) and b) efficient algorithms exist for their decoding. The decoding procedure for BCH codes consists of four steps.

1) *Evaluation of Syndromes:* If $r(z)$ is the received word and α^i, $i = 1, 2, \ldots, 2t$, are the $2t$ consecutive roots of the generator polynomial, then the syndromes are given by

$$S_i = r(\alpha^i) \quad i = 1, 2, \ldots, 2t. \tag{1}$$

The syndromes are calculated either directly by evaluating $r(z)$ at the roots using Horner's rule [15], or by first dividing $r(z)$ by the minimal polynomials of the roots and then evaluating the remainder polynomials at the roots. The complexity of this step is $\sim 2t(n-1)$.

2) *Determination of the Error-Locator Polynomial:* Suppose $e \leq t$ errors occurred at locations $j_1, j_2, \ldots, j_e$. Denote the ith *error location* α^{j_i} by X_i; then the *error-locator polynomial* is defined as

$$\sigma(z) = \prod_{i=1}^{e}(1 - X_i z) = 1 + \sigma_1 z + \sigma_2 z^2 + \cdots + \sigma_e z^e. \tag{2}$$

It is seen that the zeros of the error-locator polynomial are the inverse error locations. Knowledge of the polynomial thus serves to determine the error locations. Two popular methods are available for computing the error-locator polynomial: the shift-register synthesis algorithm of Berlekamp and Massey [2] and the Euclidean algorithm of Sugiyama *et al.* [3]. The Berlekamp-Massey algorithm makes use of the fact that the error-locator polynomial coefficients are elementary symmetric functions of $\{\alpha^{j_i}\}$ and are related to the syndromes $\{S_i\}$ by the recurrence relation

$$S_j = -\sum_{i=1}^{e} \sigma_i S_{j-i} \quad j = e+1, \ldots, 2t. \tag{3}$$

The algorithm interprets the equation as a linear feedback filter and computes the error-locator polynomial by finding the *minimum* length shift-register which will generate the requisite syndromes. The Berlekamp-Massey algorithm has (multiplicative) complexity [3]

$$C_\mu = 4t^2 + 2te - e^2 + 10t + e. \tag{4}$$

Alternatively, the error-locator polynomial can be computed using the Euclidean algorithm of Sugiyama *et al.* The Euclidean algorithm exploits another relationship between the error-locator polynomial and the syndromes, namely

$$\sigma(z) \equiv \omega(z)s(z) \bmod z^{2t}. \tag{5}$$

Here

$$s(z) = \sum_{i=1}^{2t} S_i z^{i-1} = S_1 + S_2 z + \cdots + S_{2t} z^{2t-1} \tag{6}$$

represents the *syndrome polynomial* and

$$\omega(z) = \sum_{j=1}^{e} Y_j X_j \prod_{i \neq j}(1 - X_i z) \tag{7}$$

is the *error-evaluator polynomial.* In $\omega(z)$, X_i denotes as before the ith error location and Y_i denotes the ith *error value.* Given the syndrome polynomial $s(z)$, the Euclidean algorithm produces both $\sigma(z)$ and $\omega(z)$. The Euclidean algorithm is somewhat less efficient than the Berlekamp–Massey algorithm; however, it has the advantage of being easier to understand and is in fact the algorithm of choice for a number of researchers. The Euclidean algorithm has (multiplicative) complexity [3]

$$C_\mu = 8te - \frac{1}{2}e^2 + \frac{13}{2}e. \tag{8}$$

3) *Determination of the Roots of the Error-Locator Polynomial:* Once the error-locator polynomial is known, the next task is to determine its roots. This is done using a procedure known as the Chien search. The search begins by summing the coefficients of the error-locator polynomial. The sum equals zero iff 1 ($\alpha^0 = \alpha^n$) is a root. After determining whether 1 is a root, the

Paper approved by S. G. Wilson, the Editor for Coding Theory and Applications of the IEEE Communications Society. Manuscript received March 16, 1993; received September 24, 1993.

J. J. Hong is with the Department of Electrical Engineering and Center for Telecommunications Research, Columbia University, NY 10027 USA.

M. Vetterli is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720 USA.

IEEE Log Number 9411088.

coefficients of the error-locator polynomial are modified according to the formula

$$\sigma_k \mapsto \alpha^k \sigma_k \quad k = 0, 1, \ldots, e. \tag{9}$$

The effect of the mapping is to cyclically shift $\{\alpha^i\}$. The new coefficients are again summed. Due to the cyclic shift, the sum now equals 0 iff α^{n-1} is a root. This process is repeated until all the roots of the error-locator polynomial are found. Since the Chien procedure searches, in order, $\alpha^n, \alpha^{n-1}, \ldots$, its complexity is determined by the root with the smallest index. Thus, if i, $1 \leq i \leq n$, is the least index such that α^i is a root of $\sigma(z)$, then the (multiplicative) complexity of the Chien algorithm is

$$C_\mu = e(n - i). \tag{10}$$

4) *Determination of Error Values:* The final step of the BCH decoding procedure involves computing the actual error values. This is done using the following algorithm due to Forney [15]

$$Y_i = -\frac{\omega(X_i^{-1})}{\sigma'(X_i^{-1})} \quad i = 1, 2, \ldots, e \tag{11}$$

where $\sigma'(z) = -\sum_{j=1}^{e} X_j \prod_{i \neq j}(1 - X_i z)$ is the formal derivative of the error-locator polynomial. Assuming that the two polynomials are evaluated using Horner's rule, it is easy to verify that Forney's algorithm has the following multiplicative (C_μ) and additive (C_α) complexities

$$C_\mu = 2e^2 - e \tag{12}$$

$$C_\alpha = 2(e^2 - e). \tag{13}$$

In this paper we will propose alternative algorithms for each step of the BCH decoding procedure. The algorithms are presented as tools from which the designer may pick and choose along with existing algorithms as deemed appropriate. In Section II, we examine steps 1) and 3) of the decoding procedure. We cast the problems in the spectral domain and propose pruned FFT's for their computation. While it is difficult to quantify the merits of this approach in general, we demonstrate the feasibility of the technique with two important examples: one involving a Fermat number transform and the other a space communication code. In Section III, we examine the problem of determining the error-locator polynomial. We propose a simple algorithm that is based on Gaussian elimination and LU decomposition. Though elementary, the procedure has lower complexity than the Euclidean algorithm and the Berlekamp–Massey algorithm for codes of practical interest ($t \leq 15$). Finally, in Section IV, we consider the problem of error evaluation. For this step, we propose an algorithm that is based on the fast Vandermonde solver of Björck and Pereyra [5]. It will be shown that the proposed algorithm has a complexity approximately half of that of the Forney algorithm.

II. Syndrome Evaluation and Root Finding

Steps 1) and 3) of the BCH decoding algorithm are similar in that both entail the evaluation of polynomials at select elements of the extension field. The technique most often employed to carry out these two steps is Horner's method of polynomial evaluation (or variants of it, e.g., the Chien search). Though conceptually simple, these two steps are potentially the most computationally intensive steps in the BCH decoding procedure [7]. The reason for this is that the complexity of these two steps depends on n, the block length of the code. By contrast, the complexity of determining the error-locator polynomial and the error values is a function of t, the error-correction capability of the code. In general, $n \gg t$.

The high cost of syndrome evaluation and root finding can be traced to the iterative nature of their computation procedures. The same set of computations is carried out for each syndrome or each root totally independent of any other syndromes or roots. If all the syndromes or all the roots can be computed together, a reduction in complexity is perhaps possible. To that end, it is fruitful to examine syndrome evaluation and root finding from the spectral point of view.

Consider the syndromes

$$S_i = r(\alpha^i) = \sum_{j=0}^{n-1} r_j \alpha^{ij} \quad i = 1, 2, \ldots, 2t. \tag{14}$$

Since α is an element of order n, these equations may be interpreted as a Fourier transform with the syndromes representing $2t$ contiguous components of the spectrum of the received word $r(z)$. Thus an alternative way to compute the syndromes is to take the Fourier transform of $r(z)$ and discard the unwanted spectral components.

Root finding can similarly be cast in the spectral domain. Given the error-locator polynomial $\sigma(z)$, the roots of $\sigma(z)$ are those powers of α that satisfy

$$\sigma(\alpha^i) = \sum_{j=0}^{e} \sigma_j \alpha^{ij} = 0. \tag{15}$$

Note that the sum does not run from 0 to $n - 1$ as required in a Fourier transform. However, that is easily fixed by defining $\sigma_{e+1} = \sigma_{e+2} = \cdots = \sigma_{n-1} = 0$. Doing so allows us to rewrite the above equation as

$$\Lambda_i \stackrel{\text{def}}{=} \mathcal{F}\{\sigma_i\} = \sum_{j=0}^{n-1} \sigma_j \alpha^{ij} = \sigma(\alpha^i) = 0. \tag{16}$$

Thus α^i is a zero of $\sigma(z)$ iff Λ_i is zero. This provides another way to compute the roots of the error-locator polynomial: pad the error-locator polynomial with zeros and take its n-point Fourier transform. Where the spectrum is zero, $\sigma(z)$ has a root there.

As an aside, let us mention that the celebrated Chien search has a very simple interpretation in the Fourier domain. Recall that the Chien search works by repeatedly summing the error-locator polynomial coefficients and checking whether that sum is zero followed by a modification of the coefficients according to $\sigma_k \mapsto \alpha^k \sigma_k \ \forall k$. In the Fourier domain, this is equivalent

to repeatedly checking whether the spectral component Λ_0 is zero followed by circularly shifting the spectrum of $\sigma(z)$ "up" by one. The latter interpretation follows from the modulation property of the Fourier transform [15]

$$\{\sigma_i\} \leftrightarrow \{\Lambda_j\} \Longleftrightarrow \{\alpha^i \sigma_i\} \leftrightarrow \{\Lambda_{((j+1))}\}. \tag{17}$$

In other words, the Chien search is equivalent to computing, in order, $\Lambda_0 = \Lambda_n, \Lambda_{n-1}, \ldots, \Lambda_1$.

The idea of using FFT's to evaluate syndromes and find roots of the error-locator polynomial has been suggested by a number of authors [15], [7]. However, [7] suggests that FFT's cannot be profitably employed unless $\log n$ is on the order of e or t, which is seldom the case. Indeed, by doing a few practical examples, one finds that the FFT approach is, by and large, less efficient than the polynomial-based approaches. This seems surprising at first since polynomial evaluation is equivalent to computing the DFT point by point which is known to be inefficient. The apparent contradiction is easily resolved once one realizes that in syndrome evaluation and root finding either not all inputs are present or not all output are required. Indeed, for syndrome evaluation, only $2t$ points of the n-point spectrum are necessary and for root finding, only $e+1$ points of the n-point input are nonzero. Thus the appropriate algorithms to use are not FFT algorithms, but rather, *pruned* FFT algorithms. Clearly, syndrome evaluation requires *output pruning* and root finding requires *input pruning*.

It is difficult to give a general comparison of the pruned FFT approach versus the polynomial approach. Much depends on the block length of the code, the error-correction capability of the code, the actual number of errors that occurred, the locations of the errors, and the actual FFT algorithms chosen. However, in every case known to us, pruning substantially reduces the amount of computation required. The reduction in complexity is attributable to the fact that practical codes have small error-correction capabilities relative to their code lengths. From the discussion above, this means that much of the complexity of a full-scale FFT is unnecessary. By eliminating the unwanted computations, a significant reduction in complexity can be achieved. Since it is not our purpose to develop pruned FFT algorithms but rather to demonstrate their usefulness, we will illustrate the technique with two examples.

Example 1) Fermat Number Transforms: A Fermat number transform is a Fourier transform over GF(2^m+1), where 2^m+1 is a prime called a Fermat prime. The first four Fermat primes are 5, 17, 257, and 65537 corresponding to $m = 2, 4, 8$, and 16. A Fermat number transform is unusual (for finite fields) in that it is a transform of size 2^m (or divisors thereof). Hence, the ubiquitous radix-2 FFT algorithm can be used to compute the transform. For BCH decoding, a Fermat number transform has the added advantage that there exist well-developed algorithms for both input and output pruning [8]–[12]. Moreover, the complexities of the pruning algorithms are well-studied. For input pruning, if only the first 2^l of the 2^m input points are nonzero, then by deleting the redundant butterflies from the radix-2 algorithm, a factor of l/m reduction in computation time is possible [9]. Similarly, if only the first 2^k of the 2^m output points are desired, pruning the unnecessary butterflies results in a factor of $(k+2(1-2^{k-m}))/m$ reduction in computation time [8], [10].

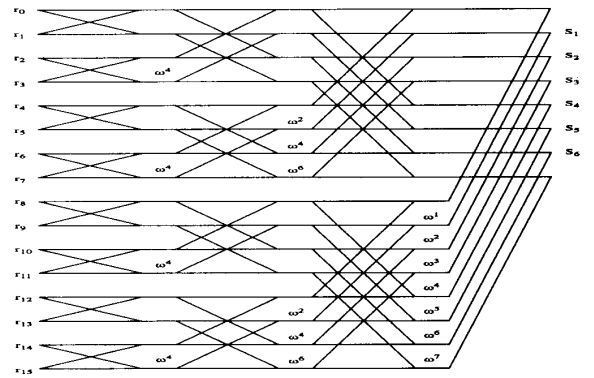


Fig. 1. 16-point FFT with output pruning.

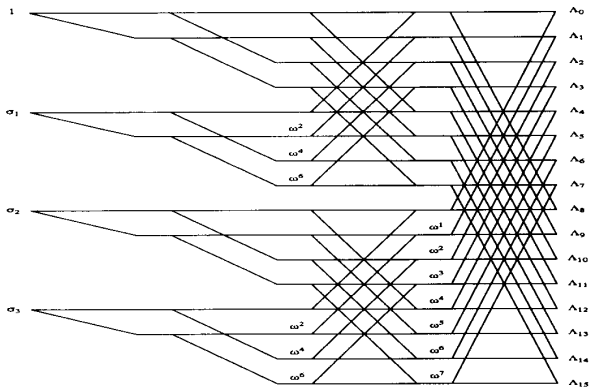


Fig. 2. 16-point FFT with input pruning.

Consider the Reed-Solomon code RS(16, 10) over GF(17). This is a distance 7 code capable of correcting 3 errors. If one were to compute the 6 syndromes $S_1, S_2, \ldots, S_6$ using Horner's method of polynomial evaluation, it would require a total of $6 \times 15 = 90$ multiplications. By contrast, if one computes the syndromes using a pruned DIT FFT as in Fig. 1, then the total number of multiplications reduces to 17.

Consider now the problem of finding the roots of the error-locator polynomial. Assume that 3 errors occurred. Then the error-locator polynomial is given by $\sigma(z) = 1+\sigma_1 z+\sigma_2 z^2+\sigma_3 z^3$. The pruned FFT algorithm thus has only 4 nonzero inputs 5. As we can see from Fig. 2, it can compute the entire 16-point spectrum with only 13 multiplications.

How does the method compare with Chien? Since Chien searches the roots in the order $\alpha^0 = \alpha^n, \alpha^{n-1}, \ldots, \alpha^1$, its complexity is determined by the root with the smallest index. That being the case, the comparison of the two methods must necessarily be probabilistic. Since all but the first iteration of Chien requires 3 multiplications ($\sigma_k \mapsto \alpha^k \sigma_k, k = 1, 2, 3$), Chien can make no more than 5 iterations ($4 \times 3 = 12 < 13$) before its complexity exceeds that of the pruned FFT. In other words, the Chien approach has a lower complexity when (and only when) the 3 errors are confined

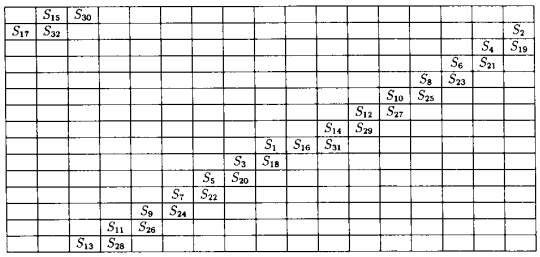


Fig. 3. Good-Thomas mapping for evaluating the syndromes of the RS(255, 223) code.

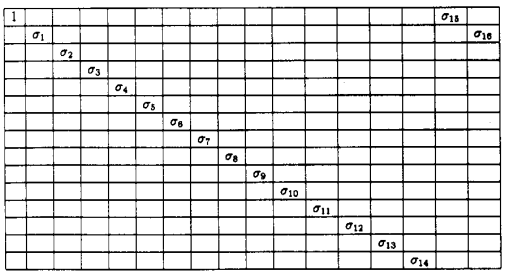


Fig. 4. Good-Thomas mapping for finding the roots of the RS(255, 223) code.

to $\{\alpha^{16}, \alpha^{15}, \alpha^{14}, \alpha^{13} \quad ^{12}\}$. How often does this happen? Assuming that all triple error patterns are equally likely, the probability of such an occurrence is

$$\frac{\binom{5}{3}}{\binom{16}{3}} = \frac{10}{560} = 0.0178.$$

Thus with probability greater than 98%, the pruned FFT algorithm will surpass Chien in performance.

Example 2) Space Communication BCH Code: Let us now look at a practical example: the (255, 223) Reed–Solomon code over GF(2^8). This is a 16-error-correcting code which is used as the outer code of an interleaved code recently standardized by the Consultative Committee for Space Data Systems (CCSDS) for use in space communication [16]. Consider the task of evaluating syndromes for this code. There are a total of $2t = 32$ syndromes. It will require $32 \times 254 = 8{,}128$ multiplications to compute all the syndromes using Horner's method. By contrast, the pruned FFT approach requires substantially less. Since $255 = 15 \times 17$ and 15 and 17 are relatively prime, we can map the one-dimensional FFT into a twiddle-factor-free two-dimensional FFT via the Good–Thomas mapping [6, 15]. Fig. 3 shows the output array obtained with the mapping $i = 15i_1 + 17i_2 \pmod{255}$. Only the syndromes $S_1, S_2, \ldots, S_{32}$ are displayed. The other values are of no consequence and need not be computed.

Clearly, the way to compute the transform is to perform the 17 15-point column transforms followed by pointwise evaluation of the 32 points along the rows. It can be shown that each 15-point FFT requires 39 multiplications. Therefore the syndromes can be computed with $(39 \times 17) + (16 \times 32) = 1{,}175$ multiplications or a saving by a factor of 7.

Consider now the task determining the roots of the error-locator polynomial. We will work out the result for $e = 16$ errors and state the results for other error values. For $e = 16$, the error-locator polynomial is given by

$$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z^2 + \cdots + \sigma_{16} z^{16}.$$

The pruned FFT thus has 17 nonzero and $255 - 17 = 238$ zero inputs. Using the Good-Thomas mapping

$$i_1 = i \pmod{15}$$
$$i_2 = i \pmod{17},$$

the one-dimensional FFT can be mapped into the two-dimensional FFT in Fig. 4.

To compute the two-dimensional transform, we will compute the 15 17-point row transforms followed by the 17 15-point column transforms. Because of the sparsity of the input, the 17-point transforms are best carried out the obvious way, using straight-forward multiplications. For the 15-point column transforms, we will use the fast 39 multiplication algorithm. The pruned FFT approach thus requires a total of $(16 \times 16) + (39 \times 17) = 919$ multiplications to find all the roots of the error-locator polynomial. The Chien approach, on the other hand, requires a minimum of $16 \times 15 = 240$ multiplications in the optimum scenario in which the 16 errors are in the first 16 positions of the search path, i.e. when the errors are in locations $\alpha^0 = \alpha^{255}, \alpha^{254}, \ldots, \alpha^{240}$. Its complexity then increases at the rate of 16 multiplications per position that the "minimum-index" zero is beyond α^{240}. When this error is at location α^{197} or beyond, the complexity of the Chien search would have exceeded that of the pruned FFT. Thus in order for Chien search to have a lower complexity than the pruned FFT algorithm, the 16 errors must be confined to positions $\{\alpha^{255}, \alpha^{254}, \ldots, \alpha^{198}\}$. How often does this happen? Assuming that all 16-error patterns are equally likely, the probability of such an occurrence is

$$\frac{\binom{58}{16}}{\binom{255}{16}} = 8.46 \times 10^{-12}.$$

Thus, with near total certainty the pruned FFT algorithm will surpass Chien in performance.

In Table I, we compare the performance of the Chien and the pruned FFT algorithms. We have listed the probability (in percent) that the Chien procedure surpasses the pruned FFT in performance versus e, the actual number of errors that occurred. We have omitted the case $e = 1$ since in that case $\sigma(z) = 1 + \sigma_1 z$ and the root can be found directly.

As is evident from the table, the Chien search procedure is a better method when the number of errors is 2 or 3. Beyond that, the pruned FFT offers better performance. It should be noted that the number of errors is known from the degree of the error-locator polynomial, thus the designer has a choice of which algorithm to use.

III. Error-Locator Polynomial Evaluation

The following algorithm is motivated by the fact that a) most BCH codes in use correct in the range from 1 to 10 errors [16], and b) the occurrence of a small number of errors is more

TABLE I
A COMPARISON OF THE CHIEN AND THE PRUNED FFT ALGORITHMS FOR THE RS(255, 223) CODE

e	probability (%) C_e(Chien) < C_e(FFT)
2	100.00 %
3	80.21 %
4	25.70 %
5	6.62 %
6	1.44 %
7	0.27 %
8	4.31×10^{-2} %
9	6.48×10^{-3} %
10	9.08×10^{-4} %
11	1.12×10^{-4} %
12	1.27×10^{-5} %
13	1.12×10^{-6} %
14	1.22×10^{-7} %
15	1.12×10^{-8} %
16	8.46×10^{-10} %

probable than the occurrence of a large number of errors. For an (n, k, d) BCH code with per symbol error probability p, the probability of e errors is

$$P_e = \binom{n}{e} p^e (1-p)^{n-e} \tag{18}$$

if we make the usual assumption that the errors occur independently. In a typical communication system, $p \ll 1$, consequently P_e decays rapidly as a function of e. As an example, for the Reed–Solomon code RS(255, 223, 33) mentioned in the previous section, assuming a value of $p = 10^{-2}$ leads to the conclusion that $P_1/P_{16} = 2.32 \times 10^7$, i.e. a single error is more probable than 16 errors by a factor of 10 million. The figure increases to $P_1/P_{16} = 2.66 \times 10^{22}$ when $p = 10^{-3}$. Since many binary communication systems aim for a bit error rate of 10^{-5} [18], translating to a symbol error rate of $\sim k10^{-5}$ when there are k bits per symbol, the probability of a large number of errors is even more remote in an actual system. For this reason, it is desirable to have an algorithm for computing the error-locator polynomial which will be efficient when the number of errors is small even at the expense of greater complexity at higher error values. In this way, the *expected complexity* is minimized.

Consider the $t \times (t+1)$ *augmented syndrome matrix*

$$S_A \stackrel{\text{def}}{=} \begin{pmatrix} S_1 & S_2 & \cdots & S_t & S_{t+1} \\ S_2 & S_3 & \cdots & S_{t+1} & S_{t+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ S_t & S_{t+1} & \cdots & S_{2t-1} & S_{2t} \end{pmatrix}. \tag{19}$$

Claim: If $e \le t$ errors occurred, then S_A is row equivalent to

$$\begin{pmatrix} 1 & & & -\sigma_e & * & \cdots & \cdots & * \\ & \ddots & & \vdots & \vdots & & & \vdots \\ & & 1 & -\sigma_1 & * & \cdots & \cdots & * \\ 0 & \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & & \vdots \\ \vdots & & \vdots & \vdots & \vdots & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{20}$$

where $*$ =don't care.

Proof: Assume that $e \le t$ errors occurred, then the syndromes satisfy the linear recurrence (3)

$$S_j = -\sum_{i=1}^{e} \sigma_i S_{j-i} \quad j = e+1, \ldots, 2t. \tag{21}$$

The equation implies that, except for the first e rows, every row of the matrix S_A is a linear combination of the previous e rows. By induction, the last $t-e$ rows of S_A are linearly dependent on the first e rows. It follows that S_A is row equivalent to a matrix in which the last $t-e$ rows are zero.

Consider now the leading principal minors of S_A

$$M_k = \begin{pmatrix} S_1 & S_2 & \cdots & S_k \\ S_2 & S_3 & \cdots & S_{k+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_k & S_{k+1} & \cdots & S_{2k-1} \end{pmatrix} \quad k = 1, 2, \ldots, t. \tag{22}$$

Gorenstein and Zieler [1] have shown that the matrices $M_t, M_{t-1}, \ldots, M_{e+1}$ are singular, whereas the matrix M_e is invertible. It is therefore possible to find a sequence of elementary row operations to reduce M_e to I_e. In the reduction process, if it becomes necessary to exchange rows (for the purpose of selecting a nonzero pivot, say), by choosing the row with the first nonzero component below the "diagonal" we will ensure that the exchange takes place within the confines of M_e. The net effect of this sequence of row operations is to apply M_e^{-1} to M_e. It follows from (21) that the application of the same sequence of operations to the first e components of column $e+1$ would yield

$$M_e^{-1} \begin{pmatrix} S_{e+1} \\ S_{e+2} \\ \vdots \\ S_{2e} \end{pmatrix} = \begin{pmatrix} -\sigma_e \\ -\sigma_{e+1} \\ \vdots \\ -\sigma_1 \end{pmatrix}. \tag{23}$$

This establishes the claim. □

In fields of characteristic 2, $\beta = -\beta$ for all β, we thus have the following simple but important corollary to the claim

Corollary: In fields of characteristic 2, S_A is row equivalent to

$$\begin{pmatrix} 1 & & & \sigma_e & * & \cdots & \cdots & * \\ & \ddots & & \vdots & \vdots & & & \vdots \\ & & 1 & \sigma_1 & * & \cdots & \cdots & * \\ 0 & \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & & \vdots \\ \vdots & & \vdots & \vdots & \vdots & & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & \cdots & 0 \end{pmatrix}. \tag{24}$$

Example 3: Consider the space communication code RS(255, 223, 33) of the previous section. This is a 16-error correcting code over GF(2^8). The Galois field GF(2^8) is constructed from the primitive polynomial $p(z) = z^8 + z^7 + z^2 + z + 1$ and the code itself is generated from the polynomial $g(z) = \prod_{j=112}^{143}(z - \alpha^{11j})$ [16]. For simplicity, assume that the zero codeword is sent and that the received word is

$$r(z) = e(z) = \alpha + \alpha^2 z.$$

It is easy to verify that the augmented syndrome matrix is given by $(*)$ shown at the bottom of the page. Performing

row reduction on the first column of the augmented syndrome martrix yields the matix in (**) shown at the bottom of the page. Performing row reduction on the second column of (**) yields the matrix in (* * *) at the top of the next page.

The procedure terminates at this point since no further reduction can be done. Since the syndromes belong to GF(2^8), a field of characteristic 2, the third column contains the error-locator polynomial coefficients σ_2 and σ_1. The error-locator polynomial is therefore

$$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z^2 = 1 + \alpha^{67} z + \alpha^{11} z^2.$$

The claim will serve only as the theoretical basis of our computation. The actual calculations will be carried out differently in order to eliminate unnecessary operations. To that end we note that the row reduction operations terminates after exactly e columns have been reduced (zeroed). Since column $e+1$ contains the desired coefficients, operations that were performed on columns $e+2$ through $t+1$ were in fact unnecessary and can be eliminated. When $e \ll t$, the saving can be substantial. A priori, we do not know the value e, so we proceed as follows: when working to reduce column i, we will apply all row operations associated with columns 1 through i to column $i+1$ and only column $i+1$. Columns $i+2$ through $t+1$ will be left untouched as they are in the augmented syndrome matrix S_A. When the reduction on the ith column is complete, we examine column $i+1$. If all elements on and below the "diagonal" are zero, the reduction cannot be continued. The process terminates with e equal to i and the negative error-locator polynomial coefficients in column $i+1$. If, however, at least one element on or below the "diagonal" is nonzero, then we know that the reduction process can be carried one step further. We move on to column $i+1$ and repeat the above process.

At this point, the elimination method is of the Gauss-Jordan type. That is to say, the reduction of the columns seeks to place zeros both above and below the diagonal. However, it is well known that Gaussian elimination (which seeks to place zeros only below the diagonal), followed by back substitution is a more efficient way to solve a linear system of equations. Therefore, instead of reducing M_e to I_e, we will transform it

$$\begin{pmatrix}
\alpha^{66} & \alpha^{212} & \alpha^{74} & \alpha^{136} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} \\
\alpha^{212} & \alpha^{74} & \alpha^{136} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} \\
\alpha^{74} & \alpha^{136} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} \\
\alpha^{136} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} \\
\alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} \\
\alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} \\
\alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} \\
\alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} \\
\alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} \\
\alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} \\
\alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} \\
\alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 \\
\alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} \\
\alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} \\
\alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} & \alpha^{19} \\
\alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} & \alpha^{19} & \alpha^{14}
\end{pmatrix}. \quad (*)$$

$$\begin{pmatrix}
1 & \alpha^{146} & \alpha^{8} & \alpha^{70} & \alpha^{133} & \alpha^{234} & \alpha^{45} & \alpha^{18} & \alpha^{10} & \alpha^{249} & \alpha^{207} & \alpha^{66} & \alpha^{48} & \alpha^{61} & \alpha^{168} & \alpha^{130} & \alpha^{212} \\
0 & \alpha^{28} & \alpha^{95} & \alpha^{234} & \alpha^{229} & \alpha^{228} & \alpha^{252} & \alpha^{101} & \alpha^{242} & \alpha^{217} & \alpha^{240} & \alpha^{104} & \alpha^{33} & \alpha^{82} & \alpha^{241} & \alpha^{239} & \alpha^{13} \\
0 & \alpha^{95} & \alpha^{162} & \alpha^{46} & \alpha^{41} & \alpha^{40} & \alpha^{64} & \alpha^{168} & \alpha^{54} & \alpha^{29} & \alpha^{52} & \alpha^{171} & \alpha^{100} & \alpha^{149} & \alpha^{53} & \alpha^{51} & \alpha^{80} \\
0 & \alpha^{234} & \alpha^{46} & \alpha^{185} & \alpha^{180} & \alpha^{179} & \alpha^{203} & \alpha^{52} & \alpha^{193} & \alpha^{168} & \alpha^{191} & \alpha^{55} & \alpha^{239} & \alpha^{33} & \alpha^{192} & \alpha^{190} & \alpha^{219} \\
0 & \alpha^{229} & \alpha^{41} & \alpha^{180} & \alpha^{175} & \alpha^{174} & \alpha^{198} & \alpha^{47} & \alpha^{188} & \alpha^{163} & \alpha^{186} & \alpha^{50} & \alpha^{234} & \alpha^{28} & \alpha^{187} & \alpha^{185} & \alpha^{214} \\
0 & \alpha^{228} & \alpha^{40} & \alpha^{179} & \alpha^{174} & \alpha^{173} & \alpha^{197} & \alpha^{46} & \alpha^{187} & \alpha^{162} & \alpha^{185} & \alpha^{49} & \alpha^{233} & \alpha^{27} & \alpha^{186} & \alpha^{184} & \alpha^{213} \\
0 & \alpha^{252} & \alpha^{64} & \alpha^{203} & \alpha^{198} & \alpha^{197} & \alpha^{221} & \alpha^{70} & \alpha^{211} & \alpha^{186} & \alpha^{209} & \alpha^{73} & \alpha^{2} & \alpha^{51} & \alpha^{210} & \alpha^{208} & \alpha^{237} \\
0 & \alpha^{101} & \alpha^{168} & \alpha^{52} & \alpha^{47} & \alpha^{46} & \alpha^{70} & \alpha^{174} & \alpha^{60} & \alpha^{35} & \alpha^{58} & \alpha^{177} & \alpha^{106} & \alpha^{155} & \alpha^{59} & \alpha^{57} & \alpha^{86} \\
0 & \alpha^{242} & \alpha^{54} & \alpha^{193} & \alpha^{188} & \alpha^{187} & \alpha^{211} & \alpha^{60} & \alpha^{201} & \alpha^{176} & \alpha^{199} & \alpha^{63} & \alpha^{247} & \alpha^{41} & \alpha^{200} & \alpha^{198} & \alpha^{227} \\
0 & \alpha^{217} & \alpha^{29} & \alpha^{168} & \alpha^{163} & \alpha^{162} & \alpha^{186} & \alpha^{35} & \alpha^{176} & \alpha^{151} & \alpha^{174} & \alpha^{38} & \alpha^{222} & \alpha^{16} & \alpha^{175} & \alpha^{173} & \alpha^{202} \\
0 & \alpha^{240} & \alpha^{52} & \alpha^{191} & \alpha^{186} & \alpha^{185} & \alpha^{209} & \alpha^{58} & \alpha^{199} & \alpha^{174} & \alpha^{197} & \alpha^{61} & \alpha^{245} & \alpha^{39} & \alpha^{198} & \alpha^{196} & \alpha^{225} \\
0 & \alpha^{104} & \alpha^{171} & \alpha^{55} & \alpha^{50} & \alpha^{49} & \alpha^{73} & \alpha^{177} & \alpha^{63} & \alpha^{38} & \alpha^{61} & \alpha^{180} & \alpha^{109} & \alpha^{158} & \alpha^{62} & \alpha^{60} & \alpha^{89} \\
0 & \alpha^{33} & \alpha^{100} & \alpha^{239} & \alpha^{234} & \alpha^{233} & \alpha^{2} & \alpha^{106} & \alpha^{247} & \alpha^{222} & \alpha^{245} & \alpha^{109} & \alpha^{38} & \alpha^{87} & \alpha^{246} & \alpha^{244} & \alpha^{18} \\
0 & \alpha^{82} & \alpha^{149} & \alpha^{33} & \alpha^{28} & \alpha^{27} & \alpha^{51} & \alpha^{155} & \alpha^{41} & \alpha^{16} & \alpha^{39} & \alpha^{158} & \alpha^{87} & \alpha^{136} & \alpha^{40} & \alpha^{38} & \alpha^{67} \\
0 & \alpha^{241} & \alpha^{53} & \alpha^{192} & \alpha^{187} & \alpha^{186} & \alpha^{210} & \alpha^{59} & \alpha^{200} & \alpha^{175} & \alpha^{198} & \alpha^{62} & \alpha^{246} & \alpha^{40} & \alpha^{199} & \alpha^{197} & \alpha^{226} \\
0 & \alpha^{239} & \alpha^{51} & \alpha^{190} & \alpha^{185} & \alpha^{184} & \alpha^{208} & \alpha^{57} & \alpha^{198} & \alpha^{173} & \alpha^{196} & \alpha^{60} & \alpha^{244} & \alpha^{38} & \alpha^{197} & \alpha^{195} & \alpha^{224}
\end{pmatrix}. \quad (**)$$

$$\begin{pmatrix}
1 & 0 & \alpha^{11} & \alpha^{78} & \alpha^{217} & \alpha^{212} & \alpha^{211} & \alpha^{235} & \alpha^{84} & \alpha^{225} & \alpha^{200} & \alpha^{223} & \alpha^{87} & \alpha^{16} & \alpha^{65} & \alpha^{224} & \alpha^{222} \\
0 & 1 & \alpha^{67} & \alpha^{206} & \alpha^{201} & \alpha^{200} & \alpha^{224} & \alpha^{73} & \alpha^{214} & \alpha^{189} & \alpha^{212} & \alpha^{76} & \alpha^{5} & \alpha^{54} & \alpha^{213} & \alpha^{211} & \alpha^{240} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}. \qquad (***)$$

to a unit upper triangular matrix.

$$S_A \longmapsto \left(\begin{array}{ccc|c|c|ccc}
1 & + & + & | & \beta_e & | & * & \cdots & * \\
 & \ddots & + & | & \vdots & | & \vdots & \ddots & \vdots \\
 & & 1 & | & \beta_1 & | & * & \cdots & * \\
\hline
- & - & - & - & - & - & - & - & - \\
\hline
0 & \cdots & 0 & | & 0 & | & * & \cdots & * \\
\vdots & \ddots & \vdots & | & \vdots & | & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & | & 0 & | & * & \cdots & *
\end{array}\right). \qquad (25)$$

Here the *'s represent *don't care* values and the +'s denote unspecified values which may or may not be zero. A final step of back substitution yields the error-locator polynomial coefficients

$$\left(\begin{array}{ccc|c}
1 & + \cdots & + & \beta_e \\
 & \ddots & \vdots & \vdots \\
 & 1 & + & \beta_2 \\
 & & 1 & \beta_1
\end{array}\right) \longmapsto \left(\begin{array}{ccc|c}
1 & & & \sigma_e \\
 & \ddots & & \vdots \\
 & & 1 & \sigma_2 \\
 & & & \sigma_1
\end{array}\right). \qquad (26)$$

Example 4: Had we performed the previous example using the technique just described, the first row reduction process would have yielded the matrix at the top of the next page. The second row reduction would have yielded the second matrix shown on the next page. The process terminates at this point since all elements on and below the third diagonal are zero. Note that columns 4 through 17 are as they were in the augmented syndrome matrix S_A. The reduction process was not applied to them, resulting in substantial savings in computation. A final step of back substitution gives the desired result

$$\left(\begin{array}{cc|c} 1 & \alpha^{146} & \alpha^{8} \\ 0 & 1 & \alpha^{67} \end{array}\right) \longmapsto \left(\begin{array}{cc|c} 1 & 0 & \alpha^{11} \\ 0 & 1 & \alpha^{67} \end{array}\right).$$

We now consider the complexity of this algorithm. With e errors, there will be te multiplications associated with the first column reduction, $(t-1)(e-1)$ multiplications associated with the second column reduction, and so on. Thus the total number of multiplications necessary to put S_A into upper triangular form (25) is

$$C_\mu(\text{row}) = \sum_{k=0}^{e-1}(t-k)(e-k) = \frac{1}{2}te^2 + \frac{1}{2}te + \frac{1}{6}(e-e^3). \qquad (27)$$

The back substitution step, on the other hand, requires

$$C_\mu(\text{back}) = 1 + 2 + \cdots + (e-1) = \frac{1}{2}(e^2 - e) \qquad (28)$$

multiplications. Thus, the multiplicative complexity of this algorithm is

$$C_\mu(\text{Gaussian}) = \frac{1}{2}t(e^2+e) - \frac{1}{6}(e^3 - 3e^2 + 2e). \qquad (29)$$

Table II compares the complexity of the modified Gaussian algorithm (29) with those of the Euclidean (8) and the Berlekamp–Massey (4) algorithms. The comparison is carried out for values of e less than or equal to 20. The value t, though unspecified, is assumed to be at least as large as e for each e.

TABLE II
A COMPARISON OF THE COMPLEXITIES OF THE BERLEKAMP-MASSEY, THE EUCLIDEAN AND THE MODIFIED GAUSSIAN ALGORITHMS

e	Gaussian	Euclidean	Berlekamp-Massey
1	t	8t+1	$4t^2$+12t
2	3t	16t+11	$4t^2$+14t-2
3	6t-1	24t+15	$4t^2$+16t-6
4	10t-4	32t+18	$4t^2$+18t-12
5	15t-10	40t+20	$4t^2$+20t-20
6	21t-20	48t+21	$4t^2$+22t-30
7	28t-35	56t+21	$4t^2$+24t-42
8	36t-56	64t+20	$4t^2$+26t-56
9	45t-84	72t+18	$4t^2$+28t-72
10	55t-120	80t+15	$4t^2$+30t-90
11	66t-165	88t+11	$4t^2$+32t-110
12	78t-220	96t+6	$4t^2$+34t-132
13	91t-286	104t	$4t^2$+36t-156
14	105t-364	112t-7	$4t^2$+38t-182
15	120t-455	120t-15	$4t^2$+40t-210
16	136t-560	128t-24	$4t^2$+42t-240
17	153t-680	136t-34	$4t^2$+44t-272
18	171t-816	144t-45	$4t^2$+46t-306
19	190t-969	152t-57	$4t^2$+48t-342
20	210t-1140	160t-70	$4t^2$+50t-380

$$\left(\begin{array}{cc|ccccccccccccccc}
1 & \alpha^{146} & \alpha^{74} & \alpha^{136} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} \\
0 & \alpha^{28} & \alpha^{136} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} \\
0 & \alpha^{95} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} \\
0 & \alpha^{234} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} \\
0 & \alpha^{229} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} \\
0 & \alpha^{228} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} \\
0 & \alpha^{252} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} \\
0 & \alpha^{101} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} \\
0 & \alpha^{242} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} \\
0 & \alpha^{217} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} \\
0 & \alpha^{240} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} \\
0 & \alpha^{104} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 \\
0 & \alpha^{33} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} \\
0 & \alpha^{82} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} \\
0 & \alpha^{241} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} & \alpha^{19} \\
0 & \alpha^{239} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} & \alpha^{19} & \alpha^{14}
\end{array}\right)$$

$$\left(\begin{array}{ccc|ccccccccccccc}
1 & \alpha^{146} & \alpha^{8} & \alpha^{136} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} \\
0 & 1 & \alpha^{67} & \alpha^{199} & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} \\
0 & 0 & 0 & \alpha^{45} & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} \\
0 & 0 & 0 & \alpha^{111} & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} \\
0 & 0 & 0 & \alpha^{84} & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} \\
0 & 0 & 0 & \alpha^{76} & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} \\
0 & 0 & 0 & \alpha^{60} & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} \\
0 & 0 & 0 & \alpha^{18} & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} \\
0 & 0 & 0 & \alpha^{132} & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} \\
0 & 0 & 0 & \alpha^{114} & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} \\
0 & 0 & 0 & \alpha^{127} & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} \\
0 & 0 & 0 & \alpha^{234} & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 \\
0 & 0 & 0 & \alpha^{196} & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} \\
0 & 0 & 0 & \alpha^{23} & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} \\
0 & 0 & 0 & \alpha^{170} & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} & \alpha^{19} \\
0 & 0 & 0 & \alpha^{158} & \alpha^{194} & \alpha^{64} & \alpha^{226} & \alpha^{213} & \alpha^{225} & \alpha^{241} & \alpha^{113} & \alpha^{57} & 0 & \alpha^{68} & \alpha^{135} & \alpha^{19} & \alpha^{14}
\end{array}\right)$$

An examination of Table II reveals that the Gaussian algorithm is uniformly more efficient than the Euclidean algorithm for error values less than or equal to 15 for any value of t. For error values greater than 15, the Gaussian approach may have higher or lower complexity than the Euclidean algorithm depending on the error-correction capability of the code. For example, if $e = t = 20$, then $C_\mu(\text{Gaussian}) = 3060$ and $C_\mu(\text{Euclid}) = 3130$.

A comparison of the Gaussian algorithm with the Berlekamp–Massey algorithm is difficult without actually specifying t. However, for any t, it is easy to find values of e for which the Gaussian approach is more efficient.

Although the Gaussian algorithm is efficient for small error values, it must be pointed out that the approach is not asymptotic. For large error values, the classical algorithms are clearly superior and should be used.

Fig. 5 is a graphical comparison of all three algorithms for the cases $1 \leq e \leq t \leq 40$. In the figure, the white region represents the inadmissible set $\{(e,t) : e > t\}$. The admissible set $\{(e,t) : e \leq t\}$ is represented by the shaded regions. The shadings reflect the regions where each of the algorithms is dominant. The light, medium and dark shadings in the figure represent, respectively, those values of t and e for which the complexities of the Euclidean, the Gaussian and the Berlekamp-Massey algorithms are minimal. In accordance with previous discussions, the Gaussian algorithm is seen to have lower complexity for small error values while the traditional algorithms are more efficient for large error values.

IV. Error Evaluation

We now consider the problem of error evaluation. As mentioned previously, this step is generally carried out using the Forney algorithm (11). In this section we will present an alternate solution that is based on the fast Vandermonde-system solver of Björck and Pereyra [5]. The algorithm will be seen to have about one half the complexity of the Forney algorithm.

TABLE III

Decoding Steps	traditional	decoder	proposed	decoder
Syndrome Evaluation	Horner:	8,128	pruned FFT:	1,175
Error-Locator Polynomial	R^n/BM:	1,044/1,384	Gauss:	520
Root Finding	Chien:	1,784	pruned FFT:	791
Error Evaluation	Forney:	120	Vandermonde:	64
Total		11,076/11,416		2,550

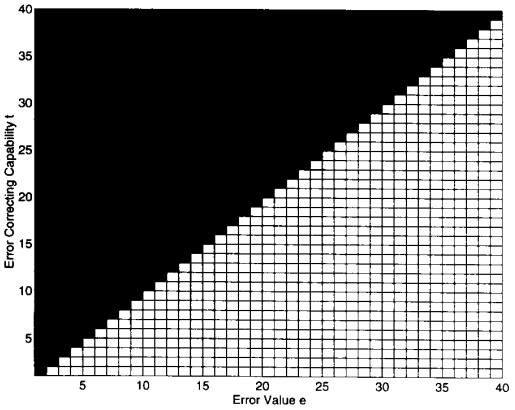


Fig. 5. A comparison of the Berlekamp–Massey, the Euclidean, and the modified Gaussian algorithms for $1 \leq e \leq t \leq 40$.

Recall that the syndromes are given by

$$S_i = r(\alpha^i) \quad i = 1, 2, \ldots, 2t \tag{30}$$

where $r(z)$ is the received word. Using the fact that the received word is the sum of a codeword and an error polynomial and the fact that $\alpha^1, \alpha^2, \ldots, \alpha^{2t}$ are zeros of the given codeword, one arrives at the following equivalent formulation of the syndromes [15]

$$S_i = \sum_{j=1}^{e} Y_j X_j^i \quad i = 1, 2, \ldots, 2t. \tag{31}$$

Here X_i and Y_i represent, respectively, the ith error location and the ith error value. This is a set of nonlinear equations which has exactly one solution when the number of errors that occurred is no more than the error-correction capability of the code [15]. The values $e, X_1, X_2, \ldots, X_e$ are determined in the first 3 steps of the BCH decoding procedure. With these values, (31) reduces to an overdetermined system of linear equations. The first e of the $2t$ equations are sufficient to determine the the e error values $Y_1, Y_2, \ldots, Y_e$.

The e equations can be written in matrix form as

$$\begin{pmatrix} X_1 & X_2 & \cdots & X_e \\ X_1^2 & X_2^2 & \cdots & X_e^2 \\ \vdots & \vdots & & \vdots \\ X_1^e & X_2^e & \cdots & X_e^e \end{pmatrix} \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_e \end{pmatrix} = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_e \end{pmatrix}. \tag{32}$$

Factoring the matrix of error locations, we have

$$\underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ X_1 & X_2 & \cdots & X_e \\ \vdots & \vdots & & \vdots \\ X_1^{e-1} & X_2^{e-1} & \cdots & X_e^{e-1} \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} X_1 & & & \\ & X_2 & & \\ & & \ddots & \\ & & & X_e \end{pmatrix}}_{D} \times \underbrace{\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_e \end{pmatrix}}_{Y} = \underbrace{\begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_e \end{pmatrix}}_{S}. \tag{33}$$

Since the X_i's are distinct and nonzero by definition, both the Vandermonde matrix V and the diagonal matrix D are invertible. Thus,

$$Y = D^{-1}V^{-1}S. \tag{34}$$

Since V is a Vandermonde matrix, the step $V^{-1}S$ is most efficiently carried out using the algorithm of Björck and Pereyra [5]. The step $D^{-1}(V^{-1}S)$ consists of pointwise multiplications carried out in the obvious manner. We will replicate the Björck-Pereyra algorithm without details. Good expositions of the algorithm may be found in [5], [17].

Algorithm for Computing Error Values:

for $k = 1$ to $e - 1$
  for $i = e$ downto $k + 1$
    $S_i = S_i - X_k S_{i-1}$
for $k = e - 1$ downto 1
  for $i = k + 1$ to e
    $S_i = S_i / (X_i - X_{i-k})$
  for $i = k$ to $e - 1$
    $S_i = S_i - S_{i+1}$
for $k = 1$ to e
  $S_i = X_i^{-1} S_i$

The algorithm above is an *in-place* algorithm. The array $S[\cdot]$ which contains the syndromes $\{S_i\}$ on entry is overwritten so that upon exit it contains the error values $\{Y_i\}$. By direct counting, it is easy to verify that this algorithm has the following complexity

$$C_\mu = e^2 \tag{35}$$

$$C_\alpha = \frac{3}{2}(e^2 - e). \tag{36}$$

Compared with the Forney algorithm (12) (13), we see that this represents ~50% reduction in the number of multiplications and ~25% reduction in the number of additions.

Example 5: As a final example, we will compare the complexities of the traditional BCH decoder and that of a decoder using the algorithms detailed in this chapter. We will compare the performance of the decoders on the 16 error-correcting RS(255, 223) code over GF(2^8). We will assume that $e = t/2 = 8$ errors occurred and that the errors are uniformly spaced. Table III shows the number of multiplications required by each decoder.

V. Conclusion

In this paper, we have introduced alternative algorithms for the various parts of the BCH decoder. We have shown that the pruned FFT can be used effectively to compute the syndromes and to evaluate the roots of the error-locator polynomial. We have also shown that a simple variation of the Gaussian elimination procedure can be used to compute the error-locator polynomial of practical BCH codes. Finally, we presented an algorithm for evaluating error values that has half the complexity of the Forney algorithm.

References

[1] D. Gorenstein and N. Zierler, "A class of error-correcting codes in p^m symbols," *J. Soc. Indian Appl. Math.*, vol. 9, pp. 207–214, June 1961.

[2] J. L. Massey, "Shift-register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122–127, Jan. 1969.

[3] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Inform. Contr.*, vol. 27, pp. 87–99, Jan. 1975.

[4] R. T. Chien, "Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. IT-10, pp. 357–363, Oct. 1964.

[5] A. Björck and V. Pereyra, "Solution of Vandermonde systems of equations," *Math. Computation*, vol. 24, pp. 893–903, Oct. 1970.

[6] I. J. Good, "The relationship between two fast fourier transforms," *IEEE Trans. Comput.*, vol. C-20, pp. 310–317, 1971.

[7] J. Ganz, "Evaluation of polynomials using the structure of the coeffi-cients," *SIAM J. Comput.*, Jan. 1992.

[8] J. D. Markel, "FFT Pruning," *IEEE Trans. Audio Electroacoust.*, vol. AU-19, pp. 305–311, Dec. 1971.

[9] D. P. Skinner, "Pruning the decimation-in-time FFT algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-24, pp. 193–194, Apr. 1976.

[10] T. V. Sreenivas and P. V. S. Rao, "FFT algorithm for both input and output pruning," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-27, pp. 291–292, June 1979.

[11] ——, "High-resolution narrow-band spectra by FFT pruning," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, pp. 254–257, Apr. 1980.

[12] K. Nagai, "Pruning the decimation-in-time FFT algorithm with frequency shift," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 1008–1010, Aug. 1986.

[13] R. Lidl and H. Niederreiter, *Finite Fields-Encyclopedia of Mathematics and Its Applications*. Reading, MA: Addison-Wesley, 1983, vol. 20.

[14] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1977.

[15] R. E. Blahut, *Theory and Practice of Error Control Codes*. Reading, MA: Addison-Wesley, 1983.

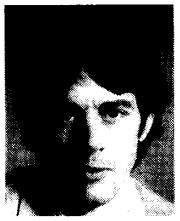
[16] H. Imai, *Essentials of Error-Control Coding Techniques*. New York: Academic, 1990.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Baltimore, MD: The Johns Hopkins University Press, 1983.

[18] M. Schwartz, *Information Transmission, Modulation, and Noise*. New York: McGraw-Hill, 1980, 3rd ed.

Jonathan Hong received the B.A. degree in mathematics from the University of California, Berkeley, in 1985, the M.S. degrees in mathematics and computer science in 1988, both from the University of Illinois, Urbana-Champaign, and the Ph.D. degree in electrical engineering from Columbia University, New York, NY, in 1994.

His research interests include finite field transforms, computational complexity and coding theory.



Martin Vetterli (M'86–SM'90) received the Dipl. El.-Ing. degree from ETH Zurich, Switzerland, in 1981, the M.S. degree from Stanford University in 1982, and the Doctorat ès Science degree from EPF Lausanne, Switzerland, in 1986.

He was a Research Assistant at Stanford and EPFL, and has worked for Siemens and AT&T Bell Laboratories. In 1986, he joined Columbia University, New York, where he is currently Associate Professor of Electrical Engineering. Since July 1993, he is also on the faculty of the Department of EECS at the University of California, Berkeley. His research interests include wavelets, multirate signal processing, computational complexity, signal processing for telecommunications and digital video processing and compression.

Dr. Vetterli is a member of the SIAM and the ACM, and of the editorial boards of *Signal Processing, Image Communication, Annals of Telecommunications Applied and Computational Harmonic Analysis* and *The Journal of Fourier Analysis and Applications*. He received the Best Paper Award of EURASIP in 1984 for his paper on multidimensional subband coding, the Research Prize of the Brown Bovery Corporation, Switzerland, in 1986 for his thesis, and the IEEE Signal Processing Society's 1991 Senior Award for a 1989 transactions paper with D. LeGall. He was a plenary speaker at the 1992 IEEE ICASSP, and is the co-author with J. Kovacevic of the forthcoming book, *Wavelets and Subband Coding* (Englewood Cliffs, NJ: Prentice-Hall, 1994).