Fast Decoding Of Alternant Codes Using A Divison-Free Analog Of An Accelerated Berlekamp-Massey Algorithm

MARC A. ARMAND WEE SIEW YEN
Department of Electrical & Computer Engineering
National University of Singapore, 10 Kent Ridge Crescent,
SINGAPORE 119260.

Abstract: – We present division-free analogs of the accelerated Berlekamp-Massey (BM) algorithms of R. Blahut based on an algorithm by G. Norton. Due to their similar structures, the computational complexity of Blahut's algorithms and ours are essentially the same. However, the division-free operation of our algorithms make them attractive, particularly for hardware implementations. We further show how our algorithms may be used for fast decoding of alternant codes. In particular, we show that our algorithms simultaneously compute both an error locator and error evaluator polynomial. Although this is achievable with the accelerated BM algorithms as well, explicit details have not been given. Finally, due to the frequent occurence of minimal polynomials in other areas besides coding theory, such as cryptography and systems theory, we expect our algorithms to have many useful applications in related areas.

Key-Words: – Minimal Realization, Alternant Codes, Key Equation, Algebraic Decoding.

1 Introduction

Alternant codes represent a large class of codes comprising important subclasses of codes, including BCH, Reed-Solomon and Goppa codes. It is well-known that in decoding alternant codes based on the conventional key equation over a formal power series ring, the error locator polynomial σ has the form $\prod(1 - \alpha_{j_i} X)$ where the j_i are the error locations in a received word. This means that the roots of σ, once found, have to be inverted to obtain the error locations. In the case of non-binary codes, having found σ, the so-called error evaluator polynomial ω is then computed to enable the error magnitudes to be found. Forney's procedure [3] which is commonly used to compute the error magnitudes may be viewed as a means of simultaneously computing and evaluating the error evaluator polynomial at the respective error locations. An algebraic decoding procedure for alternant codes based on the conventional key equation can be summarized as follows. For a received word, (i) compute the syndrome sequence, (ii) compute σ, (iii) invert the roots of σ to obtain the error locations, (iv) compute ω, and (v) compute the error magnitudes.

We focus on an analogous key equation over a Laurent series ring where the corresponding error locator polynomial has the form $\prod(X - \alpha_{j_i})$. Consequently, step (iii) is no longer necessary. Moreover, a division-free analog of the Berlekamp-Massey (BM) algorithm, namely, Algorithm MR of [5], is available for solving this alternative key equation. To significantly accelerate the decoding process, we focus on accelerating Algorithm MR. As with the BM algorithm, the computational load in any given iteration of Algorithm MR is proportional to the degree of the polynomial being updated in that iteration, and this degree increases with the number of iterations. Using a divide-and-conquer approach as in [2, Sections 11.6 & 11.7], we obtain accelerated versions of Algorithm MR which exploit the computational simplicity of the early iterations. It turns out that our accelerated algorithms compute the error locator and error evaluator polynomials simultaneous, thus fusing steps (ii) and (iv) above.

We begin by reviewing Algorithm MR. We then show how a doubling strategy may be used to accelerate the algorithm. Subsequently, by repeated application of the doubling strategy, we arrive at a fast recursive version of Algorithm MR. Finally, we show how our accelerated algorithms can be incorporated into an algebraic decoding strategy for alternant codes.

2 Algorithm MR

Let R be a domain. By $s|L$, $L \geq 1$, we denote the R-sequence $s_0, s_{-1}, \ldots, s_{-L+1}$ with generating polynomial $\Gamma(s|L) = \sum_{i=-L+1}^{0} s_i X^i$. Let the degree of $f \in R[X]$ be denoted by δf. By f_i we denote the i-th coefficient of $f \in R[X^{-1}, X]$, the ring of Laurent polynomials over R that contains $R[X]$ as a subring. The pair $(f, g) \in R[X] \times XR[X]$ is said to be a realization of $s|L$ if it satisfies the congruence relation

$$f\Gamma(s|L) \equiv g \bmod X^{-L+\delta f}$$

and $f \neq 0$ and $1 \leq \delta g \leq \delta f$. Moreover, (f, g) is said to be a minimal realization of $s|L$ if $\delta f = \min\{\delta f' : (f', g') \text{ realizes } s|L\}$ [5], i.e. f is a minimal polynomial of $s|L$. The following algorithm computes a minimal realization of a finite sequence. We write $\overline{\mu}^{(i)}$ for $(\mu^{(i)}, \beta^{(i)}) \in R[X] \times XR[X]$ for short. Addition and polynomial multiplication of minimal realizations will be by component.

Algorithm 1 *([5, Algorithm MR])*
$\alpha_0 := -1$; $\overline{\mu}^{(\alpha_0)} := (0, -X)$; $\overline{\mu}^{(0)} := (1, 0)$; $\Delta^{(\alpha_0)} := 1$; $d_0 := -1$;

for i:=0 to L-1 do
  $\Delta^{(i)} := \sum_{j=0}^{\delta\mu^{(i)}} \mu_j^{(i)} s_{-i+\delta\mu^{(i)}-j}$;
  if $\Delta^{(i)} \neq 0$ *then*
    $d_i := -d_i$; *swap*$(\overline{\mu}^{(i)}, \overline{\mu}^{(\alpha_i)})$; *swap*$(\Delta^{(i)}, \Delta^{(\alpha_i)})$;
    $\overline{\mu}^{(i+1)} := \Delta^{(\alpha_i)}\overline{\mu}^{(i)} - \Delta^{(i)} X^{d_i} \overline{\mu}^{(\alpha_i)}$;
  $d_{i+1} := d_i - 1$;

return $\overline{\mu}^{(L)}$.

The quantity $\Delta^{(i)} = \sum_{j=0}^{\delta\mu^{(i)}} \mu_j^{(i)} s_{-i+\delta\mu^{(i)}-j}$ is referred to as the discrepancy of $\mu^{(i)}$, which is the obstruction to $\overline{\mu}^{(i)}$ also realizing $s|i+1$. It is readily seen to be the $(-i + \delta\mu^{(i)})$-th coefficient of the product $\mu^{(i)}\Gamma(s|L)$.

The first step to accelerating Algorithm 1 is to recast it as follows.

Algorithm 2
Step 0: Set $\alpha_0 := -1$, $\overline{\mu}^{(\alpha_0)} := (0, -X)$, $\overline{\mu}^{(0)} := (1, 0)$, $\Delta^{(\alpha_0)} := 1$, $d_0 := -1$ *and* $i := 0$. *Then go to Step 1.*

Step 1: Set $\Delta^{(i)} := \sum_{j=0}^{\delta\mu^{(i)}} \mu_j^{(i)} s_{i+\delta\mu^{(i)}-j}$. *If* $\Delta^{(i)} \neq 0$ *then set* $z_i := 0$ *if* $d_i < 0$ *and set* $z_i := 1$ *otherwise and go to Step 2; else, set* $z_i := 1$ *and go to Step 2.*

Step 2: Set

$$\begin{bmatrix} \overline{\mu}^{(i+1)} \\ \overline{\mu}^{(\alpha_{i+1})} \end{bmatrix} := \begin{bmatrix} \Delta^{(\alpha_i)} X^{d_i(z_i-1)} & \Delta^{(i)} X^{d_i z_i} \\ 1 - z_i & z_i \end{bmatrix} \begin{bmatrix} \overline{\mu}^{(i)} \\ \overline{\mu}^{(\alpha_i)} \end{bmatrix}$$

$$\Delta^{(\alpha_{i+1})} := (1 - z_i)\Delta^{(i)} + z_i \Delta^{(\alpha_i)}$$

$$d_{i+1} := (2z_i - 1)d_i - 1.$$

Then go to Step 3.

Step 3: If $i < L - 1$ *then set* $i := i + 1$ *and go to Step 1; otherwise, exit.*

Clearly, the main computations of Algorithm 2 revolve around the equations

$$\Delta^{(i)} = \sum_{j=0}^{\delta\mu^{(i)}} \mu_j^{(i)} s_{i+\delta\mu^{(i)}-j}$$

$$\begin{bmatrix} \overline{\mu}^{(i+1)} \\ \overline{\mu}^{(\alpha_{i+1})} \end{bmatrix} = \begin{bmatrix} \Delta^{(\alpha_i)} X^{d_i(z_i-1)} & \Delta^{(i)} X^{d_i z_i} \\ 1 - z_i & z_i \end{bmatrix} \begin{bmatrix} \overline{\mu}^{(i)} \\ \overline{\mu}^{(\alpha_i)} \end{bmatrix}. \quad (1)$$

Define the matrix $\Lambda^{(i)}$ by

$$\Lambda^{(i)} = \begin{bmatrix} \Delta^{(\alpha_i)} X^{d_i(z_i-1)} & \Delta^{(i)} X^{d_i z_i} \\ 1 - z_i & z_i \end{bmatrix}$$

and the matrix $M^{(i)}$ by

$$M^{(i)} = \prod_{j=0}^{i} \Lambda^{(j)}$$

so that $M^{(i)} = \Lambda^{(i)} M^{(i-1)}$ and $M^{(-1)} = \mathbf{I}$, the identity matrix.

Then (1) may be rewritten as

$$\begin{bmatrix} \overline{\mu}^{(i+1)} \\ \overline{\mu}^{(\alpha_{i+1})} \end{bmatrix} = \left[\prod_{j=0}^{i} \Lambda^{(j)}\right] \begin{bmatrix} \overline{\mu}^{(0)} \\ \overline{\mu}^{(\alpha_0)} \end{bmatrix} = M^{(i)} \begin{bmatrix} \overline{\mu}^{(0)} \\ \overline{\mu}^{(\alpha_0)} \end{bmatrix}$$

where $\overline{\mu}^{(0)} = (1,\ 0)$ and $\overline{\mu}^{(\alpha_0)} = (0,\ -X)$.

Thus,

$$\left[\begin{array}{c} \overline{\mu}^{(i+1)} \\ \overline{\mu}^{(\alpha_{i+1})} \end{array}\right] = \left[\begin{array}{c} (M_{11}^{(i)},\ -XM_{12}^{(i)}) \\ (M_{21}^{(i)},\ -XM_{22}^{(i)}) \end{array}\right]$$

where A_{jk} denotes the element of matrix A at row j, column k.

Clearly, updating $M^{(i)}$ is equivalent to updating $\overline{\mu}^{(i+1)}$ and $\overline{\mu}^{(\alpha_{i+1})}$, although this incurs a penalty of having approximately twice as many multiplications since $M^{(i)}$ has four elements. We proceed to show how this penalty will eventually be overcome.

3 A doubling strategy

In this section, we assume that L is divisible by 2. For $i > i' \geq -1$, define the matrix

$$M^{(i,i')} = \prod_{j=i'+1}^{i} \Lambda^{(j)}.$$

Therefore,

$$M^{(i)} = M^{(i,-1)} = M^{(i,i')}M^{(i')}. \tag{2}$$

Further define the matrix $S^{(i)}$ by

$$S^{(i)} = M^{(i)}\left[\begin{array}{c} \Gamma(s|L) \\ 0 \end{array}\right] = M^{(i,i')}S^{(i')}.$$

Thus, $S^{(i)} = \Lambda^{(i)}S^{(i-1)}$ and $S^{(-1)} = [\Gamma(s|L) \quad 0]^T$. Next, we expand the product $M_{11}^{(i-1)}\Gamma(s|L)$ as follows.

$$\begin{aligned} & M_{11}^{(i-1)}\Gamma(s|L) \\ &= [M^{(i-1,i')}M^{(i')}]_{11}\Gamma(s|L) \\ &= \left[M_{11}^{(i-1,i')}M_{11}^{(i')} + M_{12}^{(i-1,i')}M_{21}^{(i')}\right]\Gamma(s|L) \\ &= M_{11}^{(i-1,i')}S_1^{(i')} + M_{12}^{(i-1,i')}S_2^{(i')} \end{aligned}$$

where $S^{(i')} = [S_1^{(i')} \quad S_2^{(i')}]^T$.

The discrepancy $\Delta^{(i)}$ of $\mu^{(i)}$ which we recall to be the $(-i + \delta\mu^{(i)})$-th coefficient of the product $\mu^{(i)}\Gamma(s|L) = M_{11}^{(i-1)}\Gamma(s|L)$ can therefore be expressed as

$$\Delta^{(i)} = \sum_{j=0}^{\delta M_{11}^{(i-1,i')}} M_{11,j}^{(i-1,i')} S^{(i')}_{1,-i+\delta M_{11}^{(i-1,i')}+\delta M_{11}^{(i')}-j} + \sum_{j=0}^{\delta M_{12}^{(i-1,i')}} M_{12,j}^{(i-1,i')} S^{(i')}_{2,-i+\delta M_{11}^{(i-1,i')}+\delta M_{11}^{(i')}-j} \tag{3}$$

Since L is assumed to be even, we can split Algorithm 2 into two halves by computing $M^{((L-2)/2)}$ and then modifying the original sequence $s|L$ to compute $M^{(L-1,(L-2)/2)}$. Then, taking the product $M^{(L-1,(L-2)/2)}M^{((L-2)/2)}$ yields $M^{(L-1)}$.

Specifically, for the first half, i.e. for $0 \leq i < L/2$, from (2) and (3), the discrepancy $\Delta^{(i)}$ may be computed using the equation

$$\begin{aligned} \Delta^{(i)} &= \sum_{j=0}^{\delta M_{11}^{(i-1)}} M_{11,j}^{(i-1)} S^{(-1)}_{1,-i+\delta M_{11}^{(i-1)}-j} + \sum_{j=0}^{\delta M_{12}^{(i-1)}} M_{12,j}^{(i-1)} S^{(-1)}_{2,-i+\delta M_{11}^{(i-1)}-j} \\ &= \sum_{j=0}^{\delta M_{11}^{(i-1)}} M_{11,j}^{(i-1)} S^{(-1)}_{1,-i+\delta M_{11}^{(i-1)}-j} \end{aligned} \tag{4}$$

since $S_2^{(-1)} = 0$. At the end of the first half, we then modify $S^{(-1)}$ as $S^{((L-2)/2)} = M^{((L-2)/2)}S^{(-1)}$ and store a copy of $M^{((L-2)/2)}$ as well as $\delta M_{11}^{((L-2)/2)}$. Let $\delta = \delta M_{11}^{((L-2)/2)}$.

For the second half, i.e. for $L/2 \leq i < L$, from (3), $\Delta^{(i)}$ may be computed using the equation

$$\begin{aligned} \Delta^{(i)} &= \sum_{j=0}^{\delta M_{11}^{(i-1)}} M_{11,j}^{(i-1)} S^{((L-2)/2)}_{1,-i+\delta M_{11}^{(i-1)}+\delta-j} + \sum_{j=0}^{\delta M_{12}^{(i-1)}} M_{12,j}^{(i-1)} S^{((L-2)/2)}_{2,-i+\delta M_{11}^{(i-1)}+\delta-j} \end{aligned} \tag{5}$$

with $M^{((L-2)/2)}$ re-initialized as $M^{((L-2)/2)} = \mathbf{I}$. In this way, $M_{11}^{(i-1)}$ and $M_{12}^{(i-1)}$ in (5) are the first row elements of $M^{(i-1,(L-2)/2)}M^{((L-2)/2)}$ (prior to the re-initialization), as required.

This completes the justification of the following algorithm. Note that indices of the various quantities involved have been suppressed since only their current values are needed. In particular, we write $\Delta^{(\alpha)}$ and $\overline{\mu}^{(\alpha)}$ as Δ' and $\overline{\mu}$ in order to differentiate them from $\Delta = \Delta^{(i)}$ and $\overline{\mu} = \overline{\mu}^{(i)}$ respectively.

Algorithm 3 *(cf. [2, Figure 11.7])*
Step 0a: Set $S = [\Gamma(s|L) \quad 0]^T$, $M := \mathbf{I}$, $d := -1$, $\Delta' := 1$ *and* $i := 0$. *Then go to Step 0b.*

Step 0b: If $i = L/2$ *then set* $\delta := \delta M_{11}$, $S := M \cdot S$, $M' := M$, $M := \mathbf{I}$, $d := -1$. *Go to Step 1.*

Step 1: If $i < L/2$ *then compute* Δ *as in (4); otherwise, compute* Δ *as in (5). If* $\Delta \neq 0$ *then set* $z := 0$ *if* $d < 0$ *and set* $z := 1$ *otherwise and go to Step 2; else, set* $z := 1$ *and go to Step 2.*

Step 2: Set

$$M := \begin{bmatrix} \Delta' X^{d(z-1)} & \Delta X^{dz} \\ 1-z & z \end{bmatrix} \cdot M$$

$$\Delta' := (1-z)\Delta + z\Delta'$$

$$d := (2z-1)d - 1.$$

Then go to Step 3.

Step 3: If $i < L-1$ *then set* $i := i+1$ *and go to Step 0b; otherwise, set* $M := M \cdot M'$, $\overline{\mu} := (M_{11}, -XM_{12})$ *and exit.*

As with the algorithm in [2, Figure 11.7], it is important that fast convolutional techniques (see e.g. [2, Sections 11.1 & 11.3] for details) be used to carry out the polynomial multiplications pertaining to the matrix products $M \cdot S$ and $M \cdot M'$ in Step 0b and Step 3 respectively. If not, there will be no net computational savings.

4 A recursive version

Algorithm 3 splits a problem of length L into two halves. If $L/2$ is even, then we can further split the two halves. If L is a power of 2, this splitting can continue until we are left with problems of length 1, each requiring only a single iteration of Algorithm MR. This idea leads to Algorithm 4 below which is a recursive version of Algorithm MR and is even more computationally efficient than Algorithm 3. Its formulation is similar to that described in [2, Section 11.7] for the recursive BM algorithm and so we state our recursive algorithm without further elaboration. For simplicity, we assume that L is a power of 2. The various methods for modifying the recursive BM algorithm to accomodate the case when L is not a power of 2, as suggested in [2, Section 11.7], equally apply to Algorithm 4 and is not repeated here.

Algorithm 4 *(cf. [2, Figure 11.8])*
Step 0: Set $S = [\Gamma(s|L) \quad 0]^T$, $\Delta' := 1$, $\delta := 0$ *and* $i := 0$.

Step 1: Call procedure **RecursiveMR**.

Step 2. Set $\overline{\mu} := (M'_{11}, -XM'_{12})$ *and exit.*

Procedure **RecursiveMR**:–
Step 0: Store a copy of S *and* M *and go to Step 1.*

Step 1: If $L = 1$ *then go to Step 2a; otherwise, go to Step 3a.*

Step 2a: Set $\Delta := S_{1,-i+\delta}$. *If* $\Delta \neq 0$ *then set* $z := 0$ *and go to Step 2b; otherwise, set* $z := 1$ *and go to Step 2b.*

Step 2b: Set

$$M' := \begin{bmatrix} \Delta' X^{1-z} & \Delta \\ 1-z & z \end{bmatrix}$$

$$\Delta' := (1-z)\Delta + z\Delta'$$

$$\delta := \delta + 1 - z$$

and $i := i+1$ *and go to Step 4.*

Step 3a: Call **RecursiveMR**. *When control returns to Step 3a, go to Step 3b.*

Step 3b: Set $S := M' \cdot S$ *and* $M := M'$. *Then call* **RecursiveMR**. *When control returns to Step 3b, go to Step 3c.*

Step 3c: Set $M' := M' \cdot M$, $L := 2L$ *and go to Step 4.*

Step 4: Retrieve a copy of S *and* M *and exit.*

As before, in order for computational savings to be achieved, the availability of fast convolutional techniques is necessary for efficiently executing the polynomial multiplications pertaining to the matrix products $M' \cdot S$ and $M' \cdot M$ in Steps 3b and 3c, respectively, of **RecursiveMR**.

In the next section, we show how our accelerated algorithms may be used for fast algebraic decoding of alternant codes.

5 Fast decoding of alternant codes

We begin by briefly recalling the definition of alternant codes over finite fields. For details, see e.g. [4, Chapter 12]. For a more general definition of alternant codes over commutative rings with identity, see [7, Section 3.1].

Let $\alpha_1, \ldots, \alpha_n$ be distinct elements of $F = GF(q^m)$ and let $h_1, \ldots, h_n$ be non-zero elements of F. An alternant code over $GF(q)$ of length n and minimum distance at least $d \geq 2$ is defined by the matrix

$$H = \begin{bmatrix} h_1 & h_2 & h_3 & \ldots & h_n \\ h_1\alpha_1 & h_2\alpha_2 & h_3\alpha_3 & \ldots & h_n\alpha_n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_1\alpha_1^{d-2} & h_2\alpha_2^{d-2} & h_3\alpha_3^{d-2} & \ldots & h_n\alpha_n^{d-2} \end{bmatrix}$$

known as the parity-check matrix.

Suppose a transmitted codeword c is received as $r = c + e$. The syndrome sequence $s|2t$ is given by $r \cdot H^T = e \cdot H^T$ where t is the error-correctability of the code satisfying $t = \lfloor (d-1)/2 \rfloor$ (note: $\lfloor x \rfloor$ denotes the largest integer less than or equal to x). We shall assume that the numbers of errors in the error vector e does not exceed t. Let the support of e be denoted by $\mathrm{Supp}(e)$. Define the error locator and error evaluator polynomials by

$$\sigma = \prod_{j \in \mathrm{Supp}(e)} (X - \alpha_j)$$

and

$$\omega = \sum_{j \in \mathrm{Supp}(e)} e_j h_j \prod_{i \in \mathrm{Supp}(e), i \neq j} (X - \alpha_i)$$

respectively. From [7, Lemma 3.13], σ and ω satisfy the congruence relation

$$\Gamma(s|2t) \equiv X\omega/\sigma \bmod X^{-2t}.$$

That is to say, $(\sigma, X\omega)$ realizes $s|2t$. It can be further shown that it is the unique minimal realization of $s|2t$.

Thus, in using Algorithms 3 or 4 to compute the minimal realization $(M_{11}^{(L-1)}, -XM_{12}^{(L-1)})$ of $s|2t$, we have that $\sigma = M_{11}^{(L-1)}$ and $\omega = -M_{12}^{(L-1)}$. Rather than performing a literal evaluation of $-M_{12}^{(L-1)}$ to obtain ω, it would be easier to compute the quotient $M_{12}^{(L-1)}(\alpha_j)/(h_j M_{11}^{(L-1)\prime}(\alpha_j))$ and add the result to the j-th coefficient r_j of r to get c_j, the j-th coefficient of the nearest codeword. Here, $M_{11}^{(L-1)\prime}$ is the formal derivative of $M_{11}^{(L-1)}$.

We thus have the following decoding procedure.

Algorithm 5
Input: The received vector $r = (r_1, \ldots, r_n)$.
Output: The nearest codeword $c = (c_1, \ldots, c_n)$.

Step 1: Compute the syndrome sequence $s|2t = r \cdot H^T$. *If* $s_i = 0$ *for* $-2t + 1 \leq i \leq 0$, *then stop; otherwise go to Step 2.*

Step 2: Compute the matrix $M^{(L-1)}$ *using Algorithm 3 or 4 with* $s|2t$ *as input.*

Step 3: Evaluate $M_{11}^{(L-1)}(\alpha_j)$ *for* $1 \leq j \leq n$. *If* $M_{11}^{(L-1)}(\alpha_j) = 0$, *then* $j \in \mathrm{Supp}(e)$.

Step 4: Set $(g_1, \ldots, g_n) = (0, \ldots, 0)$. *For* $j \in \mathrm{Supp}(e)$, *set* $g_j = M_{12}^{(L-1)}(\alpha_j)/(h_j M_{11}^{(L-1)\prime}(\alpha_j))$.

Step 5: Return $c = r + g$.

Note that in contrast to decoding based on the conventional key equation in $F[[X]]$, the roots of our error locator polynomial, rather than their inverses, correspond directly to the error locations.

6 Conclusion

We have presented division-free analogues of the accelerated BM algorithms of [2]. Due to their similar structures, the computational complexity of our algorithms and those of [2] are essentially the same. However, the division-free operation of our algorithms make them attractive, particularly for hardware implementations. When used in a decoding application, Blahut stated that his accelerated BM algorithms can compute an error evaluator polynomial as well but did not give details on how to do it. We have clarified this point for our algorithms, showing that our error evaluator polynomial ω is simply $-M_{12}^{(L-1)}$. Finally, minimal polynomials frequently occur in other areas besides coding theory, such as cryptography and systems theory. Thus, it is reasonable to assert that our algorithms could have many important applications in related areas.

References:

[1] Berlekamp, E., Algebraic coding theory, *New York: McGraw-Hill*, 1968.

[2] Blahut, R.E., Fast algorithms for digital signal processing. *Reading, MA: Addison-Wesley*, 1983.

[3] Forney, G.D. Jr., On decoding BCH codes, *IEEE Trans. Inform. Theory*, Vol. 11, (1965), pp. 549–557.

[4] MacWilliams, F.J., Sloane, N.J.A., The theory of error-correcting codes. *North Holland, Amsterdam*, 1977.

[5] Norton, G.H., On the minimal realizations of a finite sequence, *J. Symbolic Computation*, Vol. 20, 1995, pp. 93–115.

[6] Norton, G.H., Some decoding applications of minimal realization, *(C. Boyd Ed.) Cryptography and Coding, LNCS, Springer*, Vol. 1025, 1995, pp. 53–62.

[7] Norton, G.H., Salagean-Mandache A., On the key equation over a commutative ring, *Designs, Codes and Cryptography*, Vol. 20, No. 2, 2000, pp. 125–141.