# The Euclidean Algorithm for Generalized Minimum Distance Decoding of Reed-Solomon Codes

Sabine Kampf and Martin Bossert
Institute of Telecommunications and Applied Information Theory
University of Ulm, Germany
{sabine.kampf | martin.bossert}@uni-ulm.de

*Abstract*—**This paper presents a method to merge Generalized Minimum Distance decoding of Reed-Solomon codes with the extended Euclidean algorithm. By merge, we mean that the steps taken to perform the Generalized Minimum Distance decoding are similar to those performed by the extended Euclidean algorithm. The resulting algorithm has a complexity of $\mathcal{O}(n^2)$.**

## I. INTRODUCTION

In 1996, Ralf Kötter presented a fast algorithm for Generalized Minimum Distance (GMD) decoding of Reed-Solomon (RS) codes with a complexity of $\mathcal{O}(n^2)$ [1]. This algorithm is an extension of the well-known Berlekamp-Massey algorithm (BMA), that has been applied to decoding of RS codes up to half the minimum distance since the late 1960s [2]. Another algorithm that is often used to decode RS codes was first described by Sugiyama et. al. in 1975 [3], and is based on the extended Euclidean algorithm (EEA). Recently, there have been attempts to perform GMD decoding, also with a complexity of $\mathcal{O}(n^2)$, using polynomials that are obtained from the extended Euclidean algorithm [4], [5]. Since the BMA and EEA are known to be equivalent in the decoding of RS codes up to half the minimum distance, it is interesting to try to find an extension to the EEA that is equivalent to Kötters extension to the BMA. In this paper, we present such an extension.

The paper is organized as follows: In the next section, we shortly introduce RS codes as well as GMD decoding. In Section III, we first recall the EEA and how it is used for decoding RS codes up to half the minimum distance. Then we show, which of polynomials obtained are used as a basis for the GMD extension, and how to extend the EEA to GMD decoding. Finally, we shortly discuss how to modify our approach so that the selection of the best solution does not decrease the complexity. The conclusion follows in Section V.

## II. NOTATIONS AND DEFINITIONS

### A. RS Codes and Key Equation

To define an $\mathcal{RS}(n, k, d)$ code over $GF(q)$ with $R = \frac{k}{n}$ and minimum distance $d = n - k + 1$, let $\alpha \in GF(q)$ denote an element of order $n$, and $\alpha^j$ the $j$th power of this element. Later, the elements $\alpha_i$ will denote the $i$th element according

to a (implicitly) given indexing. A polynomial $c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$ with coefficients in $GF(q)$ is a valid codeword if the polynomial $C(x)$, whose coefficients $C_j$ are calculated by the discrete Fourier transform (DFT), i.e.

$$C_j = c(\alpha^j), \qquad j = 0, \ldots, n - 1, \qquad (1)$$

is zero at the first $n - k = d - 1$ coefficients, hence

$$C(x) = C_{d-1} x^{d-1} + \cdots + C_{n-1} x^{n-1}. \qquad (2)$$

We take the $C_i \in GF(q)$ to be the information symbols. The codeword $c(x)$ corresponding to the information word $C(x)$ is obtained through the inverse discrete Fourier transform (IDFT):

$$c_i = n^{-1} \cdot C(\alpha^{-i}), \quad i = 0, \ldots, n - 1. \qquad (3)$$

Throughout this paper, capital letters denote polynomials in the spectral domain, and small letters their correspondences in the time domain.

The transmitted codeword is corrupted by an additive error $e(x)$ of Hamming weight $t$, and the received word is $r(x) = c(x) + e(x)$. For decoding, calculate the syndrome $S(x)$:

$$S(x) = R(x) \bmod x^{d-1} = E(x) \bmod x^{d-1}. \qquad (4)$$

This syndrome is used in the *key equation* for decoding RS codes:

$$-\Omega(x) \equiv \Lambda(x) \cdot S(x) \bmod x^{d-1}, \qquad (5)$$

with the error locator polynomial $\Lambda(x)$ and the error evaluator polynomial $\Omega(x)$. These two polynomials satisfy the important degree relation:

$$\deg \Omega(x) < \deg \Lambda(x) = t. \qquad (6)$$

### B. GMD Decoding

First proposed by Forney in 1966 [6], the idea of GMD decoding is to allow soft-decision decoding of algebraically decodable codes by performing $m + 1$ decoding trials, each with a different number of positions being erased. In each trial, the decoder may either output an error locator polynomial as defined in (5), or declare a decoding failure. Consequently, a list of up to $m + 1$ candidate error locator polynomials is obtained, and the decoder should select one of these candidate words, that is best according to a certain criterion.

In order to determine the set $\mathcal{X}_j$ of erased positions in iteration $j$, the decoder needs to be supplied with reliability information regarding the decisions made. Reliability is defined intuitively, i.e. the less reliable a decision, the more probable it is that the received value is incorrect. Therefore, the least reliable positions are erased first. It is known, cf. e.g. [1], that decoding is possible if the number of errors $t$ and the number of erasures $\epsilon$ fulfill

$$2t + \epsilon < d. \tag{7}$$

We choose $|\mathcal{X}_0| = 0$ (i.e. we start with decoding errors only), $|\mathcal{X}_1| = 2$, $|\mathcal{X}_2| = 4$, ..., $|\mathcal{X}_m| = 2 \cdot \lfloor \frac{d-1}{2} \rfloor$, and further require $\mathcal{X}_j \subset \mathcal{X}_{j+1}$. Before, $\Lambda(x)$ was defined to be an error locator polynomial. For our algorithm, we slightly modify this and take $\Lambda(x)$ to be a joint error-erasure locator polynomial. In each iteration of the GMD decoding process, we have two additional erasures. On the other hand, according to (7) we can correct one error less than in the iteration before, and so we find that the degree of our candidate error locator polynomial should increase by 1 in each iteration. Due to this fact, we state our decoding problem as follows:

**Problem 1** *In each iteration of our GMD decoding, we want to find an error locator polynomial $\Lambda(x)$, that fulfills (5) and (6), of a given degree with certain prescribed zeros.*

This is different to the problem statement of Kötter, where the polynomial of *smallest* degree is to be found. It will be seen later that in some situations, it is not possible to fulfill the requirements given in our problem statement.

In "classical" GMD decoding, the iterations are independent of each other. In each iteration, one first determines the erasure locator polynomial. Due to the erasures, the minimum distance of the RS code is virtually decreased, and the decoder tries to find an error locator from the shorter syndrome. Because the complexity of decoding with the EEA is $\mathcal{O}(n^2)$, the overall decoding complexity of this approach is $\mathcal{O}(n^3)$. However, the decoding complexity can be decreased if erasing of positions is performed incrementally, i.e. the decoding result of iteration $j$ is used together with the additional erasures in $\mathcal{X}_{j+1} \setminus \mathcal{X}_j$ to yield the decoding result of iteration $j + 1$. The first such method was presented in [1], the complexity of his approach being $\mathcal{O}(n^2)$ (actually, Kötter claims the complexity to be $\mathcal{O}(nd)$, but since in general $d = \mathcal{O}(n)$, this is asymptotically the same). His algorithm is an extension of the BMA. However, it is known that decoding up to $\lfloor \frac{d-1}{2} \rfloor$ with the BMA and the EEA are equivalent. Therefore, we want to show that it is possible to modify the EEA, such that GMD decoding is merged into the decoding process.

## III. The Extended Euclidean Algorithm

The possibility of applying the EEA in the decoding of RS codes up to $\lfloor \frac{d-1}{2} \rfloor$ was first presented by Sugiyama et. al. in 1975 [3]. This decoding approach will be shortly reviewed in the first part of this section, as it is the basis for the extended decoding approach.

In the second and third part of this section, we will present an algorithm that integrates GMD decoding into the EEA. At the end of the third part, we show how to combine the formulas given to form the algorithm.

### A. Decoding up to $\lfloor \frac{d-1}{2} \rfloor$

The EEA uses input polynomials $A(x) = r^{(0)}(x)$ and $B(x) = r^{(-1)}(x)$ to recursively calculate a series of quotient polynomials $q^{(j+1)}(x)$ and remainders $r^{(j+1)}(x)$ that fulfill:

$$r^{(j+1)}(x) = r^{(j-1)}(x) - q^{(j+1)}(x) \cdot r^{(j)}(x), \tag{8}$$

with $\deg r^{(j+1)}(x) < \deg r^{(j)}(x)$. The algorithm stops when $r^{(j+1)}(x) = 0$. From the quotient polynomials, a series of auxiliary polynomials $u^{(j+1)}(x)$ is obtained recursively, namely

$$u^{(j+1)}(x) = u^{(j-1)}(x) - q^{(j+1)}(x) \cdot u^{(j)}(x), \tag{9}$$

where $u^{(-1)}(x) = 0$ and $u^{(0)}(x) = 1$. The degrees of these auxiliary polynomials are given by

$$\deg u^{(j)}(x) = \sum_{i=1}^{j} \deg q^{(i)}(x). \tag{10}$$

Further, these polynomials fulfill the relation

$$u^{(j)}(x) \cdot A(x) = r^{(j)}(x) \mod B(x), \tag{11}$$

which has a form similar to the key equation (5). This implies that the EEA can be used for solving (5). Hence by setting $A(x) = S(x)$ and $B(x) = x^{d-1}$, in some steps of the EEA, whenever

$$\deg u^{(j)}(x) > \deg r^{(j)}(x), \tag{12}$$

we obtain polynomials fulfilling both (5) and (6). If the number of errors $t$, i.e. the number of nonzero coefficients in $e(x)$, is limited by $t \le \lfloor \frac{d-1}{2} \rfloor$, then it is known [3] that $t = \deg u^{(j)}(x)$, $\Lambda(x) = u^{(j)}(x)$ and $\Omega(x) = -r^{(j)}(x)$ where $j$ is the smallest index for which (12) is fulfilled.

### B. From Classical Decoding to GMD Extension

In [5] it was shown that $c^{(j+1)}$, the number of coefficients of $q^{(j+1)}(x)$ that can be determined from the syndrome, is given by

$$c^{(j+1)} = \deg r^{(j)}(x) - \deg u^{(j)}(x) + 1. \tag{13}$$

Further, (8) yields that

$$\deg q^{(j+1)}(x) = \deg r^{(j-1)}(x) - \deg r^{(j)}(x). \tag{14}$$

As long as $\deg q^{(j+1)}(x) + 1 \le c^{(j+1)}$, the quotient polynomials are calculated as in the classical decoding procedure. This can be done as long as $\deg q^{(j+1)}(x) \le \lfloor \frac{d-1}{2} \rfloor$ [5]. If $\deg q^{(j+1)}(x) + 1 > c^{(j+1)}$, we switch to the GMD extension described in the next paragraph. In order to set the initial polynomials for the extension, we have to distinguish two cases: If $c^{(j+1)} \le 0$, then no coefficient of the next quotient polynomial can be determined. Hence, we use

$$u^{(j)}(x) =: \Delta^{(1,0)}(x) \text{ and } u^{(j-1)}(x) =: \Delta^{(2,0)}(x). \tag{15}$$

On the other hand, if $c^{(j+1)} > 0$, then it is possible to determine the upmost $c^{(j+1)}$ coefficients of $q^{(j+1)}(x)$ - we will call this part $\hat{q}^{(1)}(x)$, because it belongs to the "quotient" polynomial determined in the first iteration of the GMD extension - and it would be unwise to discard this information. Simulations have shown, that the best performance is achieved if we set $\Delta^{(1,0)}(x) := u^{(j)}(x)$ and $\Delta^{(2,0)}(x) := u^{(j-1)}(x) - \hat{q}^{(1)}(x)u^{(j)}(x)$. Unfortunately, we do not know yet why this performs better than using the definitions in (15) and just taking $\hat{q}^{(1)}(x)$ into account only in the first iteration.

### C. GMD Extension

In this section, we again use $j$ to index the iterations done in the GMD part of our algorithm. However, we count those iterations independently of the ones performed by the EEA before. Because the decoding up to $\lfloor \frac{d-1}{2} \rfloor$ is just decoding without erasures, $j$ now corresponds to the iterations defined in Section II-B. The basic idea is the following: We start with decoding up to half the minimum distance, as described before. Once the polynomials $u^{(j)}(x)$ can no longer be determined by the syndrome and we switch to the GMD extension, the decoder starts to determine the "quotient" polynomials $q^{(j)}(x)$ with the help of the reliability information given. Namely, two positions $\alpha_1$ and $\alpha_2$ are erased in each iteration, where

$$\{\alpha_1, \alpha_2\} = \mathcal{X}_{j+1} \setminus \mathcal{X}_j. \qquad (16)$$

Comparing (15) to (9), we set the equation to be solved by the GMD extension to be

$$\Delta^{(1,j+1)}(x) = \Delta^{(2,j)}(x) - q^{(j)}(x)\Delta^{(1,j)}(x). \qquad (17)$$

Consequently, $q^{(j)}(x)$ is determined in such a way that

$$\Delta^{(1,j+1)}(\alpha_1) = \Delta^{(1,j+1)}(\alpha_2) = 0. \qquad (18)$$

Now, the polynomials $\Delta^{(1)}(x)$ take the role of $u(x)$ in the classical decoding up to $\lfloor \frac{d-1}{2} \rfloor$, i.e. they form the list of candidate error locator polynomials from which the decoding result is chosen. Note that the erased positions in one iteration are always named $\alpha_1$ and $\alpha_2$, there is no separate indication of the iteration. It should always be clear from the context, to which pair of positions the two variables refer.

As mentioned above, we want the degree of the polynomial $\Delta^{(1)}(x)$ to increase by one in each iteration. In order to ensure this degree, we force $\alpha_1$ and $\alpha_2$ to be zeros of the polynomial

$$\Delta^{(1,j+1)}(x) = a\Delta^{(2,j)}(x) - (x+b) \cdot \Delta^{(1,j)}(x). \qquad (19)$$

Thus, we have a system of two linear equations and two unknowns, so we can give the general solution

$$a = \frac{\Delta^{(1,j)}(\alpha_1)\Delta^{(1,j)}(\alpha_2)(\alpha_1 - \alpha_2)}{\Delta^{(1,j)}(\alpha_1)\Delta^{(2,j)}(\alpha_2) - \Delta^{(1,j)}(\alpha_2)\Delta^{(2,j)}(\alpha_1)},$$
$$b = \frac{\Delta^{(1,j)}(\alpha_1)\Delta^{(2,j)}(\alpha_2)\alpha_1 - \Delta^{(1,j)}(\alpha_2)\Delta^{(2,j)}(\alpha_1)\alpha_2}{\Delta^{(1,j)}(\alpha_1)\Delta^{(2,j)}(\alpha_2) - \Delta^{(1,j)}(\alpha_2)\Delta^{(2,j)}(\alpha_1)}.$$
$$(20)$$

Because we do not require the polynomials to be monic, we can avoid the division in the actual implementation. However,

for the analysis of the algorithm we prefer to use the formulas given as they provide the easier insight regarding the cases when the intended updating is not possible.

Of course, $\Delta^{(2)}(x)$ also needs to be updated to enforce the required zeros, since otherwise it will not be guaranteed in further iterations that $\Delta^{(1)}(x)$ still has zeros at *all* positions in the corresponding erasure set $\mathcal{X}_j$. The updating of $\Delta^{(2)}(x)$ is performed by

$$\Delta^{(2,j+1)}(x) = \Delta^{(1,j)}(x) - (a \cdot x + b) \cdot \Delta^{(2,j)}(x). \qquad (21)$$

$\Delta^{(1,j)}(x)$ is multiplied by 1 because we want to have deg $\Delta^{(2,j+1)}(x) =$ deg $\Delta^{(1,j)}(x)$. This is derived from the fact that in (9), the same auxiliary polynomial is used twice, once in the role of $\Delta^{(1)}(x)$ and in the next iteration in that of $\Delta^{(2)}(x)$. The correct zeros are obtained if

$$a = \frac{\Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2) - \Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1)}{\Delta^{(2)}(\alpha_1)\Delta^{(2)}(\alpha_2)(\alpha_1 - \alpha_2)},$$
$$b = \frac{\Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1)\alpha_1 - \Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2)\alpha_2}{\Delta^{(2)}(\alpha_1)\Delta^{(2)}(\alpha_2)(\alpha_1 - \alpha_2)}.$$
$$(22)$$

Updating according to these two rules will be called regular updating. It is directly seen, that the solutions in (20) and (22) do not always exist: Regular updating of $\Delta^{(1)}(x)$ is not possible if $\Delta^{(1,j)}(\alpha_1)\Delta^{(2,j)}(\alpha_2) - \Delta^{(1,j)}(\alpha_2)\Delta^{(2,j)}(\alpha_1) = 0$. This happens if

$$\Delta^{(1,j)}(\alpha_1) = \Delta^{(1,j)}(\alpha_2) = 0 \qquad (23)$$

or

$$\Delta^{(2,j)}(\alpha_1) = \Delta^{(2,j)}(\alpha_2) = 0, \qquad (24)$$

and rarely also in other cases when the terms in the denominator of (20) are all not zero, but the denominator is. For these cases, we allow the algorithm to update $\Delta^{(1)}(x)$ in such a way that deg $\Delta^{(1,j+1)}(x) \neq$ deg $\Delta^{(1,j)}(x) + 1$. However, we keep track of this process, and perform a compensation step in some later iteration $j + j_0$ such that we get deg $\Delta^{(1,j+j_0)}(x) =$ deg $\Delta^{(1,j)}(x) + j_0$. We try to choose $j_0$ as small as possible, and in most situations it is actually possible to have $j_0 = 2$, hence we do not intensively study the case when compensation is not immediately possible. Additionally, if $\Delta^{(2)}(\alpha_1) = 0$ or $\Delta^{(2)}(\alpha_2) = 0$, the updating of $\Delta^{(2)}(x)$ has to be performed in a different way. Yet a closer look at the polynomials shows, that in this case it is sufficient to set $\Delta^{(2,j+1)}(x) = (x - \alpha_1)\Delta^{(2,j)}(x)$ if $\Delta^{(2)}(\alpha_2) = 0$ and vice versa. This might result in a polynomial of smaller degree than the intended one, but it is more important that the degree of the polynomial is not too large.

Before we study the updating procedures in the special cases indicated above, it should be noted that no further cases than the ones described before need to be distinguished. Because the auxiliary polynomials $u^{(j-1)}(x)$ and $u^{(j)}(x)$ fulfill the relation [3]

$$u^{(j)}(x)v^{(j-1)}(x) - u^{(j-1)}(x)v^{(j)}(x) = \pm 1, \qquad (25)$$

where the $v^{(j)}(x)$ can also be calculated recursively in the EEA, but are not needed for decoding RS codes. The greatest common divisor (gcd) of two polynomials calculated in consecutive iterations of the EEA is 1, and so this is true for $\Delta^{(1,0)}(x)$ and $\Delta^{(2,0)}(x)$. Of course, in the further iterations, the gcd of $\Delta^{(1,j)}(x)$ and $\Delta^{(2,j)}(x)$ will at least have roots at all positions in $\mathcal{X}_j$. Indeed, close examination shows that the gcd contains exactly these roots and no further common factor.

To illustrate this with an example, we will explicitly calculate the gcd for regular updating in the first GMD iteration. For the further iterations as well as the other cases described later on, this claim can be verified in a similar manner. First, we take a look at the proof for (25) as given in [3]. There, the calculation of the auxiliary polynomials is written in matrix form as

$$\begin{pmatrix} u^{(j)} & u^{(j-1)} \\ v^{(j)} & v^{(j-1)} \end{pmatrix} = \begin{pmatrix} q^{(1)} & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} q^{(2)} & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} q^{(j)} & 1 \\ 1 & 0 \end{pmatrix} \quad (26)$$

and taking the determinant on both sides immediately gives the relation (25). To extend this to our approach, we also use the matrix representation, namely

$$\begin{pmatrix} \Delta^{(1,j)} & \Delta^{(2,j)} \\ p^{(1)}(x) & p^{(2)}(x) \end{pmatrix} = \begin{pmatrix} \Delta^{(1,j-1)} & \Delta^{(2,j-1)} \\ p^{(3)}(x) & p^{(4)}(x) \end{pmatrix} \cdot \begin{pmatrix} q^{(j)} & 1 \\ a & \bar{q}^{(j)} \end{pmatrix}, \quad (27)$$

with polynomials $p^{(i)}(x)$ that are not used in our algorithm. For the first iteration, i.e. $j = 1$, we substitute (26) for the last step of the classical decoding procedure into (27), and by taking the determinant we obtain

$$u^{(j)}(x)p^{(2)}(x) - u^{(j-1)}(x)p^{(1)}(x) = q^{(1)}(x) \cdot \bar{q}^{(1)}(x) - a, \quad (28)$$

the right hand side possibly multiplied by $-1$. Since we know that the gcd includes the factors $(x - \alpha_1)$ and $(x - \alpha_2)$, the gcd cannot have degree less than 2. On the other hand, the right hand side of (28) is a polynomial of degree 2, and it is a multiple of the gcd. Hence the gcd has degree at most 2, so it is immediately clear that the gcd consists of exactly the two factors given, in particular no root at any value $\alpha_i$ that is to be erased in a later iteration, is contained in the gcd.

Now we will turn to the special cases, where the updating rule (19) cannot be used. First, we study the case given in (24). Due to the fact that $\Delta^{(1)}(x)$ and $\Delta^{(2)}(x)$ are coprime, we have $\Delta^{(1,j)}(\alpha_1) \neq 0, \Delta^{(1,j)}(\alpha_2) \neq 0$. In such a case, $(x + b)\Delta^{(1,j)}(x)$ can only have a root at one of the required zeros, and adding a multiple of $\Delta^{(2,j)}(x)$ cannot bring either position to zero. Therefore, we choose to set deg $\Delta^{(1,j+1)}(x) = $ deg $\Delta^{(1,j)}(x) + 2$. Then it is easy to find the updating rules

$$\Delta^{(1,j+1)}(x) = (x - \alpha_1)(x - \alpha_2)\Delta^{(1,j)}(x), \\ \Delta^{(2,j+1)}(x) = \Delta^{(2,j)}(x). \quad (29)$$

In the next iteration, the decoder should try to compensate for this decision. The thought leading to the result is the following: If we check how the polynomial $\Delta^{(1,j+2)}(x)$ is composed of $\Delta^{(1,j)}(x)$ and $\Delta^{(2,j)}(x)$ for regular updating, we find that the

first is multiplied by a polynomial of degree 2 and the latter by a polynomial of degree 1. We therefore set

$$\Delta^{(1,j+2)}(x) = (x + a)\Delta^{(2,j+1)}(x) + b\Delta^{(1,j+1)}(x), \\ \Delta^{(2,j+2)}(x) = (x - \alpha_1)(x - \alpha_2)\Delta^{(2,j+1)}(x), \quad (30)$$

which in combination with (29) gives the desired degrees for two updating steps. $\Delta^{(1,j+2)}(x)$ has the desired zeros if

$$a = \frac{\Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2)\alpha_1 - \Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1)\alpha_2}{\Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2) - \Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1)}, \\ b = \frac{\Delta^{(2)}(\alpha_1)\Delta^{(2)}(\alpha_2)(\alpha_1 - \alpha_2)}{\Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2) - \Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1)}; \quad (31)$$

we abbreviated $\Delta^{(1)} = \Delta^{(1,j+1)}$ and $\Delta^{(2)} = \Delta^{(2,j+1)}$.

If $\Delta^{(1,j)}(\alpha_1)\Delta^{(2,j)}(\alpha_2) - \Delta^{(1,j)}(\alpha_2)\Delta^{(2,j)}(\alpha_1) = 0$ without any of the involved terms being zero, we perform the update of $\Delta^{(1)}(x)$ as in (29). However, the updating of $\Delta^{(2)}(x)$ still is done according to (21), since (of course) $\Delta^{(2,j)}(x)$ does not yet have the required zeros. After the compensation step, $\Delta^{(2)}(x)$ will then have the same degree as $\Delta^{(1)}(x)$. However, recursive substitution, in order to get $\Delta^{(1)}(x)$ in dependence of $\Delta^{(1,0)}(x)$ and $\Delta^{(2,0)}(x)$ shows that these polynomials are multiplied by the intended degree in $\Delta^{(1)}(x)$, so this fact does not cause major problems, and simulations have shown that correct decoding is actually achieved with this setup.

The last special case that needs to be taken into account is $\Delta^{(1,j)}(\alpha_1) = \Delta^{(1,j)}(\alpha_2) = 0$. Here, there is no need to update $\Delta^{(1)}(x)$, but forcing the zeros in $\Delta^{(2,j+1)}(x)$ is not possible with the formula given in (21): The solution in (22) is valid, but the fact that $a = b = 0$ implies that $\Delta^{(2,j)}(x)$ is discarded, hence all further solutions would be multiples of $\Delta^{(1,j)}(x)$, which is not wanted. Therefore, we use the updating rules

$$\Delta^{(1,j+1)}(x) = \Delta^{(1,j)}(x), \\ \Delta^{(2,j+1)}(x) = (x - \alpha_1)(x - \alpha_2)\Delta^{(2,j)}(x). \quad (32)$$

The compensation step then is given as

$$\Delta^{(1,j+2)}(x) = (x - \alpha_1)(x - \alpha_2)\Delta^{(1,j+1)}(x), \\ \Delta^{(2,j+2)}(x) = a\Delta^{(2,j+1)}(x) - (x + b)\Delta^{(1,j+1)}(x), \quad (33)$$

where

$$a = \frac{\Delta^{(1)}(\alpha_1)\Delta^{(1)}(\alpha_2)(\alpha_1 - \alpha_2)}{\Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1) - \Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2)}, \\ b = \frac{\Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2)\alpha_1 - \Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1)\alpha_2}{\Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1) - \Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2)}. \quad (34)$$

Performing the compensation steps, both that in (30) and in (33) isn't always possible either. The situations in which compensation fails are actually the same as those where regular updating fails. Consequently, the same special updating rules are used again in this cases. If the same rule is used twice, two compensation steps are required later on. On the other hand, (29) and (32) serve as compensation steps for each other. Further, compensation is possible but should not be performed in (30) if $\Delta^{(2,j+1)}(\alpha_1)$ or $\Delta^{(2,j+1)}(\alpha_2)$ is zero: In this case, by performing the compensation step, one discards

the polynomial $\Delta^{(1,j+1)}(x)$, and all error locator polynomials obtained further are multiples of $\Delta^{(2,j+1)}(x)$ which is not wanted. Therefore, it is better to perform regular updating in this situation and try compensation in the next iteration. The same holds if the compensation in (33) is to be performed and $\Delta^{(1,j+1)}(\alpha_1) = 0$ or $\Delta^{(1,j+1)}(\alpha_2) = 0$.

The situation in these special cases is a little different if $\hat{q}^{(1)}(x)$ exists. As mentioned before, the updating can be written to avoid the division in the calculation of the coefficients - such a case is equivalent to multiplying both $\Delta^{(1,j)}(x)$ and $\Delta^{(2,j)}(x)$ by a constant. As a result, $\Delta^{(1,j+1)}(x) = c\Delta^{(2,j+1)}(x)$ with constant $c$ would be obtained, and in the next iteration, forcing more zeros would result in $\Delta^{(1,j+2)}(x) = 0$. This is still true for the changed definition of $\Delta^{(2,0)}(x)$, and so updating is best done as presented before. However, because we used the term $\hat{q}^{(1)}(x)u^{(j)}(x)$ in the definition of $\Delta^{(2,0)}(x)$, in these cases another solution may be obtained: As long as the constant multiplied to $\Delta^{(2,j)}(x)$ is not zero, the result includes a term $a_1 \cdot x^i \Delta^{(1,j)}(x)$ with $i > 1$. Therefore, in such cases an additional solution - of the form $\Delta^{(1,j+1)}(x) = a\Delta^{(2,j)}(x) + b\Delta^{(1,j)}(x)$ - is a valid solution and therefore it is stored, increasing the maximum list size at the decoder output.

We conclude this section by sketching how the formulas given for the GMD extension interact as an algorithm. We set $\bar{\Delta} := \Delta^{(1)}(\alpha_1)\Delta^{(2)}(\alpha_2) - \Delta^{(1)}(\alpha_2)\Delta^{(2)}(\alpha_1)$ to obtain a shorter notation. The variable $dd$ introduced in the algorithm is used to keep track of the special updatings performed.

---

**Algorithm 1**: GMD extension

**Input**: Polynomials $\Delta^{(1,0)}(x)$, $\Delta^{(2,0)}(x)$, erasure sets $\mathcal{X}_j$
**Output**: List $\mathcal{L}$ of candidate error locators
Initialization: $j = 0$, $dd = 0$, $\mathcal{L} = \{\Delta^{(1,0)}(x)\}$
**while** deg $\Delta^{(1,j)}(x) < d - 2$ **do**
    calculate $\bar{\Delta}$ from $\mathcal{X}_{j+1} \setminus \mathcal{X}_j$
    **if** $dd = 0$ *and* $\bar{\Delta} \neq 0$ **then**
       | update according to (19) and (21)
    **else if** $\bar{\Delta} = 0$ **then**
       | perform special updating and adjust $dd$:
       | $(29) \Rightarrow dd = dd + 1$
       | $(32) \Rightarrow dd = dd - 1$
    **else if** $dd > 0$ *(and $\bar{\Delta} \neq 0$)* **then**
       | perform compensation step (30), $dd = dd - 1$
    **else if** $dd < 0$ *(and $\bar{\Delta} \neq 0$)* **then**
       | perform compensation step (33), $dd = dd + 1$
    **end**
    store $\Delta^{(1,j+1)}(x)$ in $\mathcal{L}$
    $j = j + 1$
**end**

---

## IV. SELECTION OF THE BEST SOLUTION

In this section, we shortly discuss how to select the best solution from the list of candidate error locators. Although the distance criteria used in most cases are trivial, the straightforward approach requires to perform error evaluation for all

$\mathcal{O}(d)$ candidate error locators, with a complexity of $\mathcal{O}(n^2)$ each. Hence, in this straightforward approach the overall complexity is $\mathcal{O}(n^3)$ and determined by the evaluation step.

The method suggested by Kötter [1] is to use evaluation vectors instead of polynomials during the algorithm. Instead of using the polynomials, evaluation vectors are calculated in the initialization to the GMD extension. In these vectors, every component corresponds to the evaluation of the polynomial at a certain field value, i.e. we substitute

$$\Delta^{(1,0)}(x) \leftrightarrow \left[\Delta^{(1,0)}(1), \Delta^{(1,0)}(\alpha), \dots \Delta^{(1,0)}(\alpha^{n-1})\right]. \quad (35)$$

and so on. Consequently, polynomial multiplications are replaced by elementwise vector multiplications, e.g.

$$x \cdot p(x) \leftrightarrow \left[p(1), \alpha p(\alpha), \alpha^2 p(\alpha^2), \dots, \alpha^{n-1} p(\alpha^{n-1})\right]. \quad (36)$$

The evaluation of a polynomial is now simply the extraction of one component from a vector. It can be verified that the complexity of the main step, namely finding all candidate error locators, can still be performed with a complexity of $\mathcal{O}(n^2)$. Further, the selection of the best solution is now also possible with complexity $\mathcal{O}(n^2)$: Using a weighted hamming metric, it is only important which positions are in error, while the actual error value is not important. Since these positions can be easily extracted from the evaluation vectors of the candidate polynomials - the positions where the evaluation vectors are zero - the complexity of calculating the weighted Hamming weight of a single candidate is therefore only $\mathcal{O}(n)$, and so finding the best solution among $\mathcal{O}(d)$ candidates can be done with complexity $\mathcal{O}(n^2)$.

## V. SUMMARY AND CONCLUSION

We presented a method that is capable of performing Generalized Minimum Distance decoding of RS codes with an overall complexity of $\mathcal{O}(n^2)$. A method that exhibits the same performance had already been introduced by Kötter in 1996 [1]. This is not surprising, since Kötters algorithm extends the Berlekamp-Massey algorithm, and ours the extended Euclidean algorithm, and these two algorithms are known to be equivalent for decoding up to half the minimum distance. However, having a different problem formulation and erasing strategy, we do not always have the same intermediate results.

## REFERENCES

[1] R. Kötter, "Fast Generalized Minimum Distance Decoding of Algebraic-Geometry and Reed-Solomon Codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 721–737, May 1996.
[2] F. J. MacWilliams and N. J. A. Sloane, *The theory of error correcting codes*, 1977.
[3] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Decoding Goppa Codes," *Information and Control*, vol. 27, no. 1, pp. 87–99, 1975.
[4] S. Kampf, A. Wachter, and M. Bossert, "A Method for Soft-Decision Decoding of Reed-Solomon Codes Based on the Extended Euclidean Algorithm," in *8th International ITG Conference on Source and Channel Coding*, Siegen, Germany, January 2010.
[5] S. Kampf and M. Bossert, "A Fast Generalized Minimum Distance Decoder for Reed-Solomon Codes Based on the Extended Euclidean Algorithm," in *IEEE International Symposium on Information Theory*, June 2010.
[6] G. D. Forney, "Generalized Minimum Distance Decoding," *IEEE Transactions on Information Theory*, vol. 12, pp. 125–131, April 1966.