Correspondence

A Hensel Lifting to Replace Factorization in List-Decoding of Algebraic–Geometric and Reed–Solomon Codes

Daniel Augot and Lancelot Pecquet

***Abstract*—This correspondence presents an algorithmic improvement to Sudan's list-decoding algorithm for Reed–Solomon codes and its generalization to algebraic–geometric codes from Shokrollahi and Wasserman. Instead of completely factoring the interpolation polynomial over the function field of the curve, we compute sufficiently many coefficients of a Hensel development to reconstruct the functions that correspond to codewords. We prove that these Hensel developments can be found efficiently using Newton's method. We also describe the algorithm in the special case of Reed–Solomon codes.**

***Index Terms*—Algebraic–geometric codes, Hensel lifting, list decoding, Newton's method, polynomials over algebraic function fields, Reed–Solomon codes.**

I. Introduction

In [10], [11], Sudan proposed an algorithm of polynomial-time complexity to decode low-rate Reed–Solomon codes [4, p. 294 ff.] beyond the usual correction capability. This problem may have several solutions, thus Sudan's algorithm is a *list-decoding* algorithm. The ideas of this algorithm have been adapted [8] by Shokrollahi and H. Wasserman to algebraic–geometric (AG) codes. More recently, Guruswami and Sudan described an enhanced version of the method for all transmission rates [2]. The structure of these algorithms can be sketched as follows:

1) find an interpolation polynomial G under constraints;
2) find some factors of degree 1 of G and retrieve the codewords from them.

In Step 2), both algorithms involve factorization or finding the roots of the polynomial G over the function field of an algebraic curve, (which turns to be $\boldsymbol{F}_q(x)$ in the case of Reed–Solomon codes).

Our contribution is a speedup of this second step, that avoids factorization. We notice that the solutions can be characterized by a finite number of coefficients of their Hensel development and we prove that this development can be computed fast using Newton's method.

For instance, in the case of Reed–Solomon codes, a bivariate polynomial $G(x,T)$ has to be factored over $\boldsymbol{F}_q$. The method is roughly as follows [13, p. 434]:

1) specialize the variable T in a well-chosen value $y_0 \in \boldsymbol{F}_q$ in $G(x,T)$;
2) factor the univariate polynomial $G(x,y_0)$;
3) lift the results to get the factorization of $G(x,T) \bmod (T-y_0)^l$ for large enough l.

As noted by a referee, both Steps 1) and 2) are eliminated in our procedure. The lifting process can be launched immediately by inspection of the symbols of the received word. This is an improvement since Step 1) may require algebraic extensions of the base field [13, p. 433], and since there is no known deterministic algorithm to perform Step 2). We use Newton's method to do the lifting, and our algorithm is completely deterministic. The same idea can be adapted in the case of AG codes.

Note that the interpolation step has been investigated by T. Høholdt and R. R. Nielsen [3] and by M. A. Shokrollahi and V. Olshevsky [6].

In Section II, we recall the list-decoding algorithm of Shokrollahi and Wasserman for AG codes, and the original version of Sudan's algorithm in the special case of Reed–Solomon codes. In Section III, we recall the main known results concerning Hensel developments of functions in a discrete valuation ring and a method to retrieve such a development using Newton's method when the function is a root of some polynomial. In Section IV, we show how this method can be used to replace the factorization step in Shokrollahi and Wasserman's algorithm. Then in Section V, we describe in detail a fast implementation of our method, for which we give some complexity estimates in Section VI. The paper contains three Appendixes: the first describes an algorithm to build a basis of functions of the vector space associated to a divisor with increasing valuations at a given place as we use such bases in our implementation. The next two Appendixes are examples of a step-by-step application of our method to a Reed–Solomon code and to a Hermitian code.

II. Outline of Shokrollahi–Wasserman Algorithm

A. General Idea

We recall the Shokrollahi and Wasserman generalization of Sudan's algorithm.

Let $\mathcal{X}$ be a projective absolutely irreducible curve defined over $K = \boldsymbol{F}_q$ and let $K(\mathcal{X})$ be its function field. A *place* of $K(\mathcal{X})$ is a maximal ideal of a discrete valuation ring of $K(\mathcal{X})$.

Let $(P_1,\ldots,P_n)$ be an n-tuple of pairwise distinct places of degree 1 of $K(\mathcal{X})$, and D be a divisor of $K(\mathcal{X})$ such that none of the P_i is in $\operatorname{Supp} D$, we denote by $\mathcal{L}(D)$ the vector space associated to D and define the evaluation mapping $\operatorname{ev} : \mathcal{L}(D) \to \boldsymbol{F}_q^n$ by

$$\operatorname{ev}(f) = (f(P_1),\ldots,f(P_n)).$$

If $2g-2 < \deg D < n$, following the terminology of [7], the $[n, k = \ell(D) = \deg D - g + 1, d \geq n-k+1-g]_q$-code $C = \operatorname{Im}(f)$ is called a *strongly algebraic–geometric* (SAG) code. The reader may refer to [9], [12] for more complete reference on AG codes. We henceforth suppose that all these parameters are fixed, and for $1 \leq i \leq n$, we also will denote by l_{P_i} the maximal valuation at place P_i of a function $f \in \mathcal{L}(D)$.

For a given $y \in \boldsymbol{F}_q^n$, the problem of finding the set $B_\tau(y)$ of all words of C at distance at most τ of y is equivalent to the problem of finding the set $B_\tau^*(y)$ of all functions $f \in \mathcal{L}(D)$ such that $f(P_i) = y_i$ for at least $n-\tau$ values of i. We suppose that y is fixed as well, and rephrase the theorem given in [8].

Theorem 1 (Shokrollahi and Wasserman, 1999): Let Δ be a divisor of degree less than $n-\tau$ such that none of the P_i is in $\operatorname{Supp} \Delta$, for all polynomials $G(T) = a_0 + \cdots + a_b T^b \in K(\mathcal{X})[T]$ such that

1) G is nonzero;
2) $G(y_i)$ is a function of $K(\mathcal{X})$ vanishing on all P_i, $i \in \{1,\ldots,n\}$;
3) a_j is in the space $\mathcal{L}(\Delta - jD)$ for all $j \in \{0,\ldots,b\}$.

Then for all $f \in B_\tau^*(y)$, f is a root of $G(T)$.

Manuscript received January 14, 2000; revised May 22, 2000.

The authors are with INRIA, Domaine de Voluceau, F 78153 Le Chesnay Cedex, France (e-mail: Daniel.Augot@inria.fr; Lancelot.Pecquet@inria.fr).

Communicated by I. F. Blake, Associate Editor for Coding Theory.
Publisher Item Identifier S 0018-9448(00)09673-5.

Proof: At any place P of $K(\mathcal{X})$, for all

$$f \in K(\mathcal{X}), v_P(G(f)) \geq \min_{0\leq j\leq b}(v_P(a_j) + j v_P(f)).$$

Thus for all j, we have

$$v_P(a_j) + j v_P(f) \geq -v_P(\Delta) + j v_P(D) - j v_P(D) = -v_P(\Delta).$$

Hence $G(f) \in \mathcal{L}(\Delta)$.

Consider the set I of indexes for which $f(P_i) = y_i$. For $i \in I$ we have $G(f) = a_0 + \cdots + a_b f^b$. Since $P_i \notin \operatorname{Supp} D \cup \operatorname{Supp} \Delta$, we can evaluate $G(f)$ at P_i. Then, by hypothesis

$$\begin{aligned}(G(f))(P_i) &= \sum_{j=0}^{b} a_j(P_i) f(P_i)^j \\ &= \sum_{j=0}^{b} a_j(P_i) y_i^j = (G(y_i))(P_i) = 0.\end{aligned}$$

Consequently, $v_{P_i}(G(f)) \geq 1$ and $(G(f)) \geq -\Gamma_I$ where

$$\Gamma_I = \Delta - \sum_{i\in I} P_i.$$

The degree of Γ_I is $\deg(\Gamma_I) = \deg(\Delta) - |I|$ and for all $f \in B^*_\tau(y)$, $|I| \geq n - \tau$. Therefore, the hypothesis $\deg \Delta < n - \tau$ implies $\deg \Gamma_I < 0$ hence $\mathcal{L}(\Gamma_I) = \{0\}$ and $G(f) = 0$. □

A polynomial G satisfying Conditions 1)–3) of Theorem 1 exists for all $y \in \boldsymbol{F}_q^n$ if τ is not too large. From [8], a necessary condition, based on the count of the number of unkowns and linear equations satisfied by G, can be stated as follows.

Theorem 2: Let C be an $[n, k]$-stronlgy AG code, for any

$$\tau \leq n + g - \left\lceil \sqrt{2n(k + g - 1)} \right\rceil$$

a polynomial satisfying hypothesis of Theorem 1 exists of degree at most $\lceil \sqrt{2n/(k+g-1)} \rceil$ for all $y \in \boldsymbol{F}_q^n$.

The algorithm is therefore as follows.

1) Find a polynomial $G(T)$ satisfying Conditions 1)–3) of Theorem 1.
2) Compute the roots f of $G(T)$ in $\mathcal{L}(D)$ such that $f(P_i) = y_i$ for at least $n - \tau$ values of i.

B. Case of Reed–Solomon Codes

The $[n, k]_q$-Reed–Solomon codes can be described as a special case of a SAG code. The corresponding curve is the projective line, which is of genus 0, and the function field $K(\mathcal{X})$ is $\boldsymbol{F}_q(x)$. All places of degree 1 of $\boldsymbol{F}_q(x)$ are the $P_i = (x - p_i)$ with $p_i \in \boldsymbol{F}_q$, plus the place at infinity $P_\infty = (1/x)$. The divisor D is $(k-1)\cdot P_\infty$, $\mathcal{L}(D)$ is the set of polynomials of degree less than k over $\boldsymbol{F}_q$.

Using a divisor $\Delta = \Big(n - \tau - 1 - b(k-1)\Big)\cdot P_\infty$, Theorem 1 can be reformulated as in Sudan's original articles [10], [11]:

Theorem 3 (Sudan, 1997): For all bivariate polynomials

$$G(x, T) = a_0(x) + \cdots + a_b(x) T^b \in \boldsymbol{F}_q[x, T]$$

such that

1) G is nonzero;
2) $G(p_i, y_i) = 0$ for all $i \in \{1, \ldots, n\}$;
3) $\deg a_j(x) < n - \tau - (k-1)j$ for all $j \in \{0, \ldots, b\}$;

then for all $f \in B^*_\tau(y), G(x, f(x)) = 0$.

By applying Theorem 2, if $\tau \leq n - \lceil \sqrt{2nk} \rceil$, a polynomial $G(x, T)$ satisfying Conditions 1)–3) of Theorem 3 exists of degree at most $\lceil \sqrt{2n/k} \rceil$. The algorithm becomes as follows.

1) Find a bivariate polynomial $G(x, T)$ as in Theorem 3.
2) Find factors of the form $(T - f(x))$ where f is a polynomial of $\boldsymbol{F}_q[x]$ of degree less than k such that $f(p_i) = y_i$ for at least $n - \tau$ values of i.

III. Hensel Lifting

In this section we recall some results on Hensel developments and their computation by Newton's method. We will use the following notation until the end of the correspondence. For a given place P of the function field $K(\mathcal{X})$, we denote by $\mathcal{O}_P$ its discrete valuation ring, by t_P a fixed uniformizer of P, and by v_P the valuation of $\mathcal{O}_P$. Any nonzero element f of $\mathcal{O}_P$ can be written in a unique way as a converging (cf. [1, p. 432]) power series in t_P with coefficients in K, called its Hensel *development at place* P or P*-adic development*

$$f = \sum_{j=0}^{\infty} \alpha_j t_P^j.$$

The Hensel development up to order l of f will be denoted by

$$\operatorname{Hens}_P(f, l) = \sum_{j=0}^{l} \alpha_j t_P^j = f \operatorname{rem} t_P^{l+1}.$$

(To emphasize the algorithmics, we distinguish $f \operatorname{rem} t_P^{l+1}$, which is a polynomial in t_P, from $f \bmod t_P^{l+1}$, which is an element of a quotient ring.) We will also denote by $\operatorname{coef}_P(f, m)$ and $\operatorname{init}_P(f)$, respectively, the mth and the first nonzero coefficient of the development of f.

We recall the following well-known results (see [5, pp. 126 ff.] and [13, pp. 243–263]).

Theorem 4 (Newton's Approximation Theorem): Let $G \in \mathcal{O}_P[T]$ and $\varphi_j \in K[t_P]$ such that

$$G(\varphi_j) = 0 \bmod t_P^{2^j}$$

and

$$G'(\varphi_j) \neq 0 \bmod t_P.$$

Then $\varphi_{j+1} = \varphi_j - G(\varphi_j)\cdot G'(\varphi_j)^{-1} \operatorname{rem} t_P^{2^{j+1}}$ is defined in $K[t_P]$ and

1) $\varphi_{j+1} = \varphi_j \bmod t_P^{2^j}$;
2) $G(\varphi_{j+1}) = 0 \bmod t_P^{2^{j+1}}$;
3) $G'(\varphi_{j+1}) \neq 0 \bmod t_P$.

Moreover, if ψ_{j+1} is a polynomial in $K[t_P]$ such that $G(\psi_{j+1}) = 0 \bmod t_P^{2^{j+1}}$ and $\psi_{j+1} = \varphi_{j+1} \bmod t_P^{2^j}$, then $\psi_{j+1} = \varphi_{j+1} \bmod t_P^{2^{j+1}}$ (unicity of Hensel development).

Proof: As $G'(\varphi_j) \neq 0 \bmod t_P$, $G'(\varphi_j) \neq 0 \bmod t_P^{2^j}$. In addition, $G(\varphi_j) = 0 \bmod t_P^{2^j}$ implies $\varphi_{j+1} - \varphi_j = 0 \bmod t_P^{2^j}$, which proves Assertion 1). Taylor's development at order two of $G(T)$ in φ_j is

$$G(T) = G(\varphi_j) + (T - \varphi_j)G'(\varphi_j) + (T - \varphi_j)^2 R(T - \varphi_j)^2$$

where R is some polynomial in $\mathcal{O}_P[T]$. If we specialize T in φ_{j+1}, we have

$$\begin{aligned}G(\varphi_{j+1}) &= G(\varphi_j) + (\varphi_{j+1} - \varphi_j)G'(\varphi_j) \\ &\quad + (\varphi_{j+1} - \varphi_j)^2 R(\varphi_{j+1} - \varphi_j) \\ &= G(\varphi_j) + (\varphi_{j+1} - \varphi_j)G'(\varphi_j) \bmod t_P^{2^{j+1}} \\ &= G(\varphi_j) + (-G(\varphi_j)\cdot G'(\varphi_j)^{-1})G'(\varphi_j) \bmod t_P^{2^{j+1}} \\ &= 0 \bmod t_P^{2^{j+1}}.\end{aligned}$$

This proves Assertion 2). Since $\varphi_{j+1} = \varphi_j \bmod t_P^{2^j}$, we have $\varphi_{j+1} = \varphi_j \bmod t_P$. Consequently, Assertion 3) follows since $G'(\varphi_j) \neq 0 \bmod t_P$ implies $G'(\varphi_{j+1}) \neq 0 \bmod t_P$.

Finally, let ψ_{j+1} be a polynomial in $K[t_P]$ such that $G(\psi_{j+1}) = 0 \bmod t_P^{2^{j+1}}$ and $\psi_{j+1} = \varphi_{j+1} \bmod t_P^{2^j}$. The second-order Taylor's development of $G(T)$ in φ_{j+1}, when specialized in ψ_{j+1} is

$$G(\psi_{j+1}) - G(\varphi_{j+1}) = (\psi_{j+1} - \varphi_{j+1})(G'(\varphi_{j+1}) + c \cdot (\psi_{j+1} - \varphi_{j+1})^2)$$

where $c \in \mathcal{O}_P$. Thus

$$(\psi_{j+1} - \varphi_{j+1})G'(\varphi_{j+1}) = 0 \bmod t_P^{2^{j+1}}.$$

Since $G'(\varphi_{j+1}) \neq 0 \bmod t_P^{2^{j+1}}$, $\psi_{j+1} - \varphi_{j+1} = 0 \bmod t_P^{2^{j+1}}$. □

Newton's *method* builds iteratively the Hensel development of the roots of a given polynomial by repeating the iterations

$$\varphi_{j+1} \leftarrow \varphi_j - G(\varphi_j) \cdot G'(\varphi_j)^{-1} \operatorname{rem} t_P^{2^{j+1}}$$

as stated in the following corollary.

Corollary 1: Let $G(T) \in \mathcal{O}_P[T]$ and $\alpha \in K$ such that $(G'(\alpha))(P) \neq 0$. For all $f \in \mathcal{O}_P$ such that $G(f) = 0$ and $f(P) = \alpha$, let $(\varphi_j)_{j \in \boldsymbol{N}}$ be the sequence defined by $\varphi_0 = \alpha$ and for $j \geq 0$

$$\varphi_{j+1} = \varphi_j - G(\varphi_j) \cdot G'(\varphi_j)^{-1} \operatorname{rem} t_P^{2^{j+1}}.$$

Then for all $j \in \boldsymbol{N}$, $\varphi_{j+1} = f \bmod t_P^{2^{j+1}}$, and $(\varphi_j)_{j \in \boldsymbol{N}}$ converges to f in $\mathcal{O}_P$.

IV. Application to Shokrollahi–Wasserman Algorithm

Let $G(T)$ ge a polynomial satisfying the conditions of Theorem 1. For $f \in B_\tau^*(y)$, we know that $G(f) = 0$ and in order to use Corollary 1 to build a sequence that converges to f, we need a place P and $\alpha \in K$ such that $f(P) = \alpha$, and $(G'(\alpha))(P) \neq 0$. We now prove that if $G(T)$ is chosen of minimal degree, then for any $f \in B_\tau^*(y)$ there is a position i such that $f(P_i) = y_i$ and $(G'(y_i))(P) \neq 0$. Thus by choosing $\alpha = y_i$, we will retrieve f using Newton's method.

Theorem 5: If $G(T)$ is a polynomial of minimal degree satisfying the conditions of Theorem 1, then for each root $f \in \mathcal{L}(D)$, there is an index $i \in \{1, \ldots, n\}$ such that $f(P_i) = y_i$ and $(G'(y_i))(P_i) \neq 0$.

Proof: Let $a_0 + a_1 T + \cdots + a_b T^b$ be a polynomial of minimal degree, satisfying the conditions of Theorem 1. For any $f \in \mathcal{L}(D)$ such that $G(f) = 0$, let $R(T) = r_0 + \cdots + r_{b-1} T^{b-1} \in K(\mathcal{X})[T]$ be such that $G(T) = (T - f)R(T)$. By identification, we have

$$G(T) = \underbrace{-fr_0}_{a_0} + \underbrace{(r_0 - fr_1)}_{a_1} T + \cdots + \underbrace{(r_{b-2} - fr_{b-1})}_{a_{b-1}} T^{b-1} + \underbrace{r_{b-1}}_{a_b} T^b.$$

Let us prove recursively that $r_j \in \mathcal{L}(D - (j+1)D)$ for $0 \leq j < b$.

It is true for $r_{b-1} = a_b \in \mathcal{L}(D - bD)$. Suppose that for some $j \leq b-1$, $r_j \in \mathcal{L}(D - (j+1)D)$. Then we have $a_j = r_{j-1} - fr_j$, hence at any place P

$$v_P(r_{j-1}) \geq \min(v_P(a_j), v_P(f) + v_P(r_j)).$$

On the one hand, we know from Condition 3) of Theorem 1 that $v_P(a_j) \geq -v_P(\Delta - jD)$; on the other hand, the recursion hypothesis tells that $v_P(r_j) \geq -v_P(\Delta - (j+1)D)$. We also know that $f \in \mathcal{L}(D)$, hence

$$v_P(f) + v_P(r_j) \geq -v_P(\Delta - jD).$$

So in all cases, $v_P(r_{j-1}) \geq -v_P(\Delta - jD)$.

We, therefore, have proved that for any $j \leq b-1$

$$r_j \in \mathcal{L}(\Delta - (j+1)D) \subseteq \mathcal{L}(\Delta - jD).$$

In particular, $R(T)$ also satisfies Condition 3) of Theorem 1.

Consider the set I of indices i such that $f(P_i) = y_i$, then for $i \notin I$, $(R(y_i))(P_i) = 0$. Suppose now that for all $i \in I$, $(R(y_i))(P_i) = 0$, then $R(T)$ would clearly satisfy Conditions 1) and 2) of Theorem 1. As we have shown that $R(T)$ satisfies Condition 3), it would contradict the fact that we have chosen $G(T)$ of minimal degree satisfying these conditions. Consequently, there exists an index i for which $f(P_i) = y_i$ and $(R(y_i))(P_i) \neq 0$. The derivative is $G'(T) = (T - f)R'(T) + R(T)$, hence $(G'(y_i))(P_i) = 0 + (R(y_i))(P_i) \neq 0$. Theorem 1 tells us that any $f \in B_\tau^*(y)$ is a root of $G(T)$. □

As we will see in Section V-B, any function in $\mathcal{L}(D)$ can be characterized by its Hensel development up to order l_{P_i}, for any $i \in \{1, \ldots, n\}$. Thus the method to perform Step 2) of Shokrollahi and Wasserman's algorithm is the following. Iterate through the set S of all positions $i \in \{1, \ldots, n\}$ such that $(G'(y_i))(P_i) \neq 0$. For any such i, compute the l_{P_i}th term of the sequence $(\varphi_j)_{j \in \boldsymbol{N}}$ as built in Corollary 1 with $\varphi_0 = y_i$ for large enough l_{P_i} (see Section V-A), then convert $\varphi_{l_{P_i}}$ to a function $f \in \mathcal{L}(D)$, and test if $f \in B_\tau(y)$.

The next section describes all details of this method and the whole algorithm for list-decoding is given in Section V-C.

V. Implementation

A. Newton's Method

In Newton's method, each iteration

$$\varphi_{j+1} \leftarrow \varphi_j - G(\varphi_j)/G'(\varphi_j) \operatorname{rem} t_P^{2^{j+1}}$$

involves a division of series, which can be replaced by a multiplication: $\varphi_{j+1} \leftarrow \varphi_j - G(\varphi_j) \cdot \eta_j \operatorname{rem} t_P^{2^{j+1}}$ where η_j is an auxiliary sequence which approximates $G'(\varphi_j)^{-1}$. More precisely, we define

Proposition 1 (Newton Inversion): Let P be a place of degree 1 of $K(\mathcal{X})$, and $h \in K[[t_P]]$ such that $h \neq 0 \bmod t_P$. Let $(\eta_j)_{j \in \boldsymbol{N}}$ be the sequence defined by $\eta_0 = h(P)^{-1}$ and for all $j \in \boldsymbol{N}$, $\eta_{j+1} = 2\eta_j + h\eta_j^2 \operatorname{rem} 2^{j+1}$. Then $(\eta_j)_{j \in \boldsymbol{N}}$ satisfies $\eta_j h = 1 \bmod t_P^{2^j}$ for all $j \in \boldsymbol{N}$.

Proof: For $j = 0$, we have $h\eta_0 = h(P)\eta_0 = 1 \bmod t_P^{2^0}$. Suppose that the result is true up to j, then

$$1 - h\eta_{j+1} = 1 - h(2\eta_j - h\eta_j^2) = 1 - 2h\eta_j + h^2\eta_j^2 = (1 - h\eta_j)^2 = 0 \bmod t_P^{2^{j+1}}$$

by the recurrence hypothesis. □

For computational purpose, the polynomial

$$G(T) = a_0 + \cdots + a_b T^b \in \mathcal{O}_P[T]$$

is replaced by the polynomial $\widetilde{G}_{P,l}$ whose coefficients are the Hensel developments at P of the coefficients of G up to order l, i.e.,

$$\widetilde{G}_{P,l}(T) = G(T) \operatorname{rem} t_P^{l+1} = \operatorname{Hens}_P(a_0, l) + \cdots + \operatorname{Hens}_P(a_b, l) \cdot T^b \in K[t_P][T].$$

Proposition 2: Let $G(T) \in \mathcal{O}_P[T]$ and $\alpha \in K$ be such that $(G'(\alpha))(P) \neq 0$, and let l be a nonnegative integer, then the sequences at the bottom of the following page are well defined and for all $f \in \mathcal{O}_P$ such that $G(f) = 0$ and $f(P) = \alpha, \operatorname{Hens}_P(f, l) = \varphi_{\lceil \log_2(l+1) \rceil}$.

Proof: First,

$$(\widetilde{G}'_{P,l}(\alpha))(P) = (G'(\alpha))(P) \neq 0$$

TABLE I
ALGORITHM I

Algorithm 1

function Newton(G, P, α, l).
Input: A polynomial $G(T) \in \mathcal{O}_P[T]$ and $\alpha \in K$ such that $\big(G(\alpha)\big)(P) = 0$ and $(G'(\alpha))(P) \neq 0$. An integer $l \geq 0$.
Ouput: A polynomial $\varphi \in K[t_P]$ of degree less than or equal to l (φ is equal to $\mathrm{Hens}_P(f, l)$ if there exists $f \in \mathcal{O}_P$ such that $f(P) = \alpha$ and $G(f) = 0$).

1. $\widetilde{G}_{P,l}(T) \leftarrow G(T) \text{ rem } t_P^{l+1}$.
2. $\widetilde{G}'_{P,l}(T) \leftarrow$ the derivative of $\widetilde{G}_{P,l}(T)$.
3. $\eta \leftarrow \big(\widetilde{G}'_{P,l}(\alpha)\big)(P)^{-1}$. *// This is η_0*
4. $\varphi \leftarrow \alpha$. *// Compute φ_0*
5. **for** j **from** 0 **to** $\lceil \log_2(l+1) \rceil - 1$ **do**
   $\eta \leftarrow 2\eta - \widetilde{G}'_{P,l}(\varphi) \cdot \eta^2 \text{ rem } t_P^{\min(2^{j+1}, l+1)}$. *// Compute η_{j+1}*
   $\varphi \leftarrow \varphi - \widetilde{G}_{P,l}(\varphi) \cdot \eta \text{ rem } t_P^{\min(2^{j+1}, l+1)}$. *// Compute φ_{j+1}*
6. **return** φ. *// Return $\varphi_{\lceil \log_2(l+1) \rceil}$*

which guarantees that the sequences are well defined. Let ψ_j be the sequence defined by $\psi_0 = \alpha$ and for all $j \in \boldsymbol{N}$

$$\psi_{j+1} = \psi_j - G(\psi_j) \cdot G'(\psi_j)^{-1} \operatorname{rem} t^{2^{j+1}}.$$

From Corollary 1, we know $\psi_{j+1} = f \bmod t_P^{2^{j+1}}$ for all $j \in \boldsymbol{N}$. Furthermore, as for all $j \in \boldsymbol{N}$

$$G(T) = \widetilde{G}_{P,l}(T) \bmod t_P^{\min(2^{j+1}, l+1)}$$

we have

$$\eta_{j+1} = (G'(\varphi_{j+1}))^{-1} \bmod t_P^{\min(2^{j+1}, l+1)}$$

and

$$\varphi_{j+1} = \psi_{j+1} \bmod t_P^{\min(2^{j+1}, l+1)}.$$

For $j + 1 = \lceil \log_2(l+1) \rceil$, we have $2^{j+1} \geq l + 1$, thus

$$\varphi_{\lceil \log_2(l+1) \rceil} = \psi_{\lceil \log_2(l+1) \rceil} \operatorname{rem} t_P^{l+1} = f \operatorname{rem} t_P^{l+1} = \mathrm{Hens}_P(f, l). \quad \square$$

We therefore have Algorithm 1 (see Table I) to compute the Hensel development at P up to order l of a root f of G provided $(G'(f(P)))(P) \neq 0$.

B. Converting Hensel's Developments to Functions

For our purpose, we focus on the roots of $G(T) \in \mathcal{O}_P[T]$ that are in $\mathcal{L}(D)$ (with $P \notin \operatorname{Supp} D$). In general, the output φ of Algorithm 1 is not necessarily the Hensel development of a function of $\mathcal{L}(D)$. It is possible to decide fast if there exists a function $f \in \mathcal{L}(D)$ whose truncated development is φ, and retrieve f in this case, as described in Algorithm 2 (see Table II). Note that it is possible that f is not a root of $G(T)$. Hensel's lemma tells us that φ is the truncation of a root of $G(T)$ in the completion of the function field, which may not be in $\mathcal{L}(D)$ *a priori*.

However, when $G(T)$ satisfies Condition 3 of Theorem 1, then the output of Algorithm 1 is always the Hensel development of a function of $\mathcal{L}(D)$, as shown in the next theorem. Consequently, Algorithm 2 will never return *fail*.

Theorem 6: Let $G(T) = a_0 + \cdots + a_b T^b$ be a polynomial such that $a_j \in \mathcal{L}(\Delta - jD)$ for $0 \leq j \leq b$. Let P be a place of degree 1 such that $P \notin \operatorname{Supp}\Delta \cup \operatorname{Supp} D$, and $\alpha \in K$ be such that $(G(\alpha))(P) = 0$ and $(G'(\alpha))(P) \neq 0$. Then with the notation of Proposition 2, for all $j \in \{0, \ldots, \lceil \log_2(l_P+1) \rceil\}$, $\varphi_{j+1} \in \mathcal{L}(D)$. Consequently, the output $\mathbf{Newton}(G, P, \alpha, l_P)$ of Algorithm 1 is the Hensel development of a function of $\mathcal{L}(D)$.

Proof: We prove this by induction. First, note that $\varphi_0 = \alpha \in \mathcal{L}(D)$ and $\eta_0 = (\tilde{G}'_{P,l_P}(\alpha))(P)^{-1} \in \mathcal{L}(D - \Delta)$. Suppose now that for some $j \in \{0, \ldots, \lceil \log_2(l_P+1) \rceil\}$, $\varphi_j \in \mathcal{L}(D)$ and $\eta_j \in \mathcal{L}(D - \Delta)$. Then

$$\tilde{G}'_{P,l_P}(\varphi_j) \in \mathcal{L}(\Delta - D), \qquad \eta_j^2 \in \mathcal{L}(2D - 2\Delta)$$

and

$$\eta_{j+1} = 2\eta_j + \tilde{G}'_{P,l_P}(\varphi_j) \cdot \eta_j^2 \operatorname{rem} t_P^{\min(2^{j+1}, l+1)} \in \mathcal{L}(D - \Delta).$$

Furthermore, $\tilde{G}_{P,l_P}(\varphi_j) \in \mathcal{L}(\Delta)$, hence $\tilde{G}_{P,l_P}(\varphi_j) \cdot \eta_{j+1} \in \mathcal{L}(D)$ so

$$\varphi_{j+1} = \varphi_j - \tilde{G}_{P,l_P}(\varphi_j) \cdot \eta_{j+1} \operatorname{rem} t_P^{\min(2^{j+1}, l+1)} \in \mathcal{L}(D). \quad \square$$

We use a special kind of basis of the space $\mathcal{L}(D)$ to retrieve a function in this space from its Hensel development at a given place.

$$\begin{cases} \eta_0 = \left(\widetilde{G}'_{P,l}(\alpha)\right)(P)^{-1} & \text{and } \forall j \in \boldsymbol{N},\ \eta_{j+1} = 2\eta_j + \widetilde{G}'_{P,l}(\varphi_j) \cdot \eta_j^2 \operatorname{rem} t_P^{\min(2^{j+1}, l+1)} \\ \varphi_0 = \alpha & \text{and } \forall j \in \boldsymbol{N},\ \varphi_{j+1} = \varphi_j - \widetilde{G}_{P,l}(\varphi_j) \cdot \eta_{j+1} \operatorname{rem} t_P^{\min(2^{j+1}, l+1)} \end{cases}$$

TABLE II
ALGORITHM 2

Algorithm 2

function SeriesToFunction(P, φ).
Precomputed: A basis $\mathcal{B}_P = (f_{P,1}, \ldots, f_{P,\kappa})$ of $\mathcal{L}(D)$ in P-reduced echelon form, the integer l_P, and a basis $\tilde{\mathcal{B}}_P = (\tilde{f}_{P,1}, \ldots, \tilde{f}_{P,\kappa})$ where $\tilde{f}_{P,j} = \mathrm{Hens}_P(f_{P,j}, l_P)$ for $1 \le j \le \kappa$. The valuation sequence $V_P = (\mathrm{v}_P(f_{P,1}), \ldots, \mathrm{v}_P(f_{P,\kappa}))$.
Input: $P \notin \mathrm{Supp}\, D$ is a place of degree 1, φ is a polynomial in $K[t_P]$ of degree less than or equal to l_P.
Ouput: The unique $f \in \mathcal{L}(D)$ such that $\mathrm{Hens}_P(f, l_P) = \varphi$ if such an f exists, *fail* otherwise.

1. Set $f \leftarrow 0$.
2. **repeat**
   if $\varphi \neq 0$ **then**
   $v \leftarrow \mathrm{v}_P(\varphi)$
   if $v \in V$ **then**
   Let i be the index for which $v = V_{P,i}$.
   $\lambda \leftarrow \mathrm{init}_P(\varphi)$.
   $f \leftarrow f + \lambda f_{P,i}$.
   $\varphi \leftarrow \varphi - \lambda \tilde{f}_{P,i}$.
   else $f \leftarrow$ *fail*.
   until $(\varphi = 0)$ **or** $(f =$ *fail*$)$
3. **return** f.

Definition 1: A basis $(f_1, \ldots, f_\kappa)$ of $\mathcal{L}(D)$ is said to be in *P-reduced echelon form* if $v_P(f_1) < \cdots < v_P(f_\kappa)$, and such that $\mathrm{init}_P(f_i)$ is 1 for all $i \in \{1, \ldots, \kappa\}$.

The *P-valuation sequence* of D is the set of valuations $v_P(f_1), \ldots, v_P(f_\kappa)$, and we denote by l_P the integer

$$\max_{f \in \mathcal{L}(D)} v_P(D) = v_P(f_\kappa).$$

See Appendix A to find how to construct such a basis from any basis of $\mathcal{L}(D)$. Note that the definition of the valuation sequence does not depend on the choice of the basis in P-reduced echelon form.

We use the result of the following remark in Algorithm 2 to verify if a Hensel development corresponds to a function of $\mathcal{L}(D)$.

Remark 1: If $(f_1, \ldots, f_\kappa)$ is a basis of $\mathcal{L}(D)$ in P-reduced echelon form, for all nonzero function $f \in \mathcal{L}(D)$, if $f = \lambda_1 f_1 + \cdots + \lambda_\kappa f_\kappa \neq 0$, then $v_P(f) = v_P(f_i)$ where $i = \min\{l \in \{1, \ldots, \kappa\} \mid \lambda_l \neq 0\}$. Hence $v_P(f) \in \{v_P(f_1), \ldots, v_P(f_\kappa)\}$. In particular, $l_P = v_P(f_\kappa)$.

Example 1: In the case of the rational function field $\boldsymbol{F}_q(x)$, the polynomial $t = x - a$ is a uniformizer of the place $P = (x - a)$ and a base of $\mathcal{L}((k-1)P_\infty)$ in P-reduced echelon form is

$$(1, (x-a), (x-a)^2, \ldots, (x-a)^{k-1}).$$

The P-valuation sequence is $(0, 1, \ldots, k-1)$ and $l_P = k - 1$. Algorithm 2 behaves simply in that case: for a polynomial $\varphi = a_0 + \cdots + a_{k-1}t^{k-1}$, it will return the polynomial $f(x) = \varphi(x - a)$.

The following fact is of interest to reduce the cost of the algorithm:

Proposition 3: Let S be the set

$$\{i \in \{1, \ldots, n\} \mid (G'(y_i))(P_i) \neq 0\}.$$

The set

$$\Phi = \{\mathbf{SeriesToFunction}(P_i, \mathbf{Newton}(G, P_i, y_i, l_{P_i})),\ i \in S\}$$

is a set of m functions $f_1, \ldots, f_m \in \mathcal{L}(D)$ among which are all functions of $B^*_\tau(y)$. The subsets $s_j = \{i \in S \mid f_j(P_i) = y_i\}$ for $1 \le j \le m$ form a partition of S.

Proof: For each position $i \in S$, we know from Theorem 6 that the output φ of Algorithm 1 is the Hensel development of a function $f = \mathbf{SeriesToFunction}(P_i, \varphi) \in \mathcal{L}(D)$. If two such functions match at some place P_i, Theorem 4 indicates the two Hensel developments will coincide and therefore the two functions will also coincide. □

This corollary suggests that once a function f has been found, using successively Algorithm 1 then Algorithm 2, it is useless to apply these algorithm at all places P_i with $i \in S$ such that $f(P_i) = y_i$ because they will lead to the same function. Consequently, it is possible to remove the set $I = \{i \in S \mid f(P_i) = y_i\}$ from the set S before it continues to find other functions.

C. The Whole Algorithm

We are now able to compute $B^*_\tau(y)$ using Algorithm 3 (see Table III). We denote by $\mathcal{B}_1, \ldots, \mathcal{B}_n$ some bases in echelon form with respect to the places $P_1, \ldots, P_n$, respectively.

VI. COMPLEXITY

A. General Complexity

We denote by $\mathsf{M}(l)$ the cost of the product of two dense power series truncated at order l with coefficients in K. This can be done in $O(l^2)$

TABLE III
ALGORITHM 3

Algorithm 3

ListDecode(y, τ).
Precomputed: For each $i \in \{1, \ldots, n\}$, a basis $\mathcal{B}_{P_i} = (f_{P_i,1}, \ldots, f_{P_i,\kappa})$ of $\mathcal{L}(D)$ in P_i-reduced echelon form, the P_i-valuation sequence V_{P_i}, and a basis $\tilde{\mathcal{B}}_{P_i} = (\tilde{f}_{P_i,1}, \ldots, \tilde{f}_{P_i,\kappa})$ where $\tilde{f}_{P_i,j} = \mathrm{Hens}_P(f_{P_i,j}, l_{P_i})$ for $1 \leq j \leq \kappa$.
Input: $y \in \mathbf{F}_q^n$, and τ such that Theorem 1 applies.
Ouput: The set $B_\tau(y) = \{c \in C \mid \mathrm{d}(c, y) \leq \tau\}$.

1. $G(T) \leftarrow$ a polynomial of minimal degree satisfying the conditions of Theorem 1.
2. Set $B \leftarrow \varnothing$.
3. Set $S \leftarrow \{i \in \{1, \ldots, n\} \mid (G(y_i))'(P_i) \neq 0\}$.
4. **for** i **in** S **do**
   a) $\varphi \leftarrow$ **Newton**(G, P_i, y_i, l_{P_i}). *// In $\mathcal{L}(D)$*
   b) $f \leftarrow$ **SeriesToFunction**(P_i, φ). *// Never fails*
   c) $c \leftarrow \mathrm{ev}(f)$ *// Well defined since $f \in \mathcal{L}(D)$.*
   d) $I \leftarrow \{j \in \{1, \ldots, n\} \mid f(P_j) = y_j\}$.
   e) $S \leftarrow S \setminus I$. *// The indices in I would lead to the same function*
   f) **if** $|I| \geq n - \tau$ **then** *// Corresponds to a codeword in $B_\tau(y)$*
      Include c into the set B.
5. **return** B.

arithmetic operations in K with standard polynomial multiplication, $O(l^{1.59})$ operations with Karatsuba multiplication and $O(l \log l)$ operations with a fast Fourier transform (FFT) (which may require a field extension in order to get a primitive lth root of unity).

Almost all operations of Algorithm 3 involve the computation of Hensel developments, valuations and evaluations of functions of $K(\mathcal{X})$. The cost of these operations depend a lot on the implementation of the function field. However, we can give a cost of the main loop of Algorithm 1.

Proposition 4: The main loop of Algorithm 1 can be performed at place P in deterministic $O(b\mathsf{M}(l_P))$ arithmetic operations in K.

Proof: First, note that all operations of the kind $u \operatorname{rem} t_P^m$ do not cost anything since they just mean not to compute terms of valuation greater than or equal to m. Using Horner's rule [13, p. 93], we can compute $\tilde{G}_{P,l}(\varphi)$ and $\tilde{G}'_{P,l}(\varphi)$ in b multiplications and additions of truncated power series at order 2^{j+1}, this can be performed in $O(b\mathsf{M}(2^{j+1}))$ operations. The other operations to compute η and φ are also multiplications and additions of truncated power series at order 2^{j+1} hence the cost of one iteration is $O(b\mathsf{M}(2^{j+1}))$ operations. By convexity of the cost function M, we have $\mathsf{M}(2^{j+1}) \geq 2\mathsf{M}(2^j)$, let us denote by N the integer $\lceil \log_2 l_P + 1 \rceil$, the global cost is

$$\begin{aligned} &O(b(\mathsf{M}(1) + \cdots + \mathsf{M}(2^N))) \\ &\quad = O(b(\mathsf{M}(2^N)/2^N + \mathsf{M}(2^N)/2^{N-1} + \cdots + \mathsf{M}(2^N))) \\ &\quad = O(b\mathsf{M}(2^N)) = O(b\mathsf{M}(l_P)) \end{aligned}$$

operations. □

B. Case of Reed–Solomon Codes

We fix a transmission rate $R = k/n$, and study the asymptotic behavior of the algorithm when n goes to infinity.

Proposition 5: For an $[n, Rn]_q$-Reed–Solomon code, Step 2) of Sudan's algorithm can be performed deterministically in $O(n\mathsf{M}(n)\log n)$ arithmetic operations in $\boldsymbol{F}_q$ and even in $O(n^2 \log n)$ operations using an FFT.

Proof: For each coefficient $a_j(x)$ of $G(T)$, we can compute all $a_j(x - p_i)$ for $1 \leq i \leq n$ in $O(\mathsf{M}(n)\log n)$ operations using fast multipoint evaluation/interpolation (cf. [13, pp. 279–285]). This can also be done in $O(n \log n)$ operations using an FFT.

The global cost of the computation of $\tilde{G}_{P_i,k-1}(T)$ for $1 \leq i \leq n$ is, therefore, $O(b\mathsf{M}(n)\log n)$ operations without an FFT and $O(bn \log n)$ operations using an FFT.

For a given i, in the Newton step (Algorithm 1), the computation of derivatives can be done in $O(bn)$ operations and the computation of η requires $O(bn)$ operations, then from Proposition 4, we perform the loop in $O(b\mathsf{M}(n))$ operations. The back-translation (Algorithm 2) can also be done by fast multipoint evaluation/interpolation in $O(\mathsf{M}(n)\log n)$ operations or in $O(n \log n)$ operations using an FFT. Steps c) and d) are free as we have evaluated f at all $p_i, 1 \leq i \leq n$. Step e) can be done in $O(n)$ operations.

The degree in T of the polynomial $G(x, T)$ is bounded above by $\sqrt{2/R}+1$, and we can consider that b is constant. The **for** loop is run n times in the worst case, therefore, the global cost is $O(n\mathsf{M}(n)\log n)$ operations without an FFT, and $O(n^2 \log n)$ operations using an FFT.□

Invoking the result of [6], we know that Step 1 of Algorithm 3 in which one builds the polynomial $G(T)$ can be performed in $O(n^2)$ operations. In the case of Reed–Solomon codes, the FFT is feasible, thus we can conclude the following.

Corollary 2: The list-decoding of an $[n, Rn]_q$-Reed–Solomon code can be performed deterministically using Sudan's algorithm in $O(n^2 \log n)$ arithmetic operations in $\boldsymbol{F}_q$.

TABLE IV
ALGORITHM 4

Algorithm 4

procedure ReducedEchelonForm($\mathcal{B}$, P).
Input: A basis $\mathcal{B} = (f_1, \ldots, f_\kappa)$ of $\mathcal{L}(D)$ and a place P.
Specification: Transforms $\mathcal{B}$ in P-echelon form.

1. $to\_do \leftarrow [1, \ldots, \kappa]$
2. **while** $|to\_do| > 0$ **do**
   a) Let v be the minimal valuation of all functions f_i $i \in to\_do$.
   b) Let i be the minimum index for which $\mathrm{v}_P(f_i) = v$.
   c) $f_i \leftarrow f_i / \mathrm{init}_P(f_i)$.
   d) Remove i from to_do.
   e) $s \leftarrow \{ j \in to\_do \mid \mathrm{v}_P(f_j) = \mathrm{v}_P(f_i) \}$.
   f) **for** $j \in s$ **do** $f_j \leftarrow f_j - \mathrm{init}_P(f_j) \cdot f_i$.
3. Sort $\mathcal{B}$ with respect to increasing valuations.

VII. Conclusion

We have presented an algorithm to find specific roots of polynomials with coefficients in the function field of an algebraic curve. This algorithm is applied to the list decoding algorithm of AG codes designed by Shokrollahi and Wasserman, where it is needed to find roots which belong to the space $\mathcal{L}(D)$ defining the AG code. The algorithm computes Hensel developments of possible solutions, using Newton's method. Assuming that the polynomial to factor has *minimal degree* with respect to the conditions imposed by Shokrollahi–Wasserman's algorithm, initial values can be found to start the iterations. This removes the step of univariate factorization in root-finding algorithms. As a consequence, our algorithm is completly deterministic and fast. We also note that it very easy to implement with a very simple control structure (one loop), and that it avoids the use of algebraic extensions of the base field.

Our algorithm can be applied to any AG codes, whether the curve is singular or not. Since many operations are dependent on manipulations of functions on the curve, we cannot give a general complexity measure. But at least we have proven a quadratic complexity up to logarithmic factors in the case of Reed–Solomon codes.

For our algorithm to work, we need the polynomial $G(T)$ to be of minimal degree. The algorithm cannot be applied to the more powerful generalization of Guruswami and Sudan [2], where multiplicities are imposed on the polynomial $G(T)$. An adaptation of this algorithm will be presented in a future paper.

Appendix A
Computation of Reduced Echelon Form Bases

The construction of a basis of $\mathcal{L}(D)$ in P-reduced echelon form, as described in Algorithm 4 (see Table IV), corresponds to the construction of a matrix in reduced echelon form from the matrix of the P-adic expansions of the f_i, up to order l_P.

Appendix B
A $[17, 5, 13]_{17}$ Extended Reed–Solomon Code

We consider the field $K = \boldsymbol{F}_{17}$. A basis of the vector space $\mathcal{L}(4P_\infty)$, of polynomials of degree less than 5 is

$$(f_1 = 1, f_2 = x, f_3 = x^2, f_4 = x^3, f_5 = x^4).$$

We build the extended Reed–Solomon code of dimension 5 by evaluating polynomials of degree less than 5 on points $p_1 = 0, p_2 = 1, \ldots, p_n = 16$. Its actual minimum distance is $n - k + 1 = 13$. Consequently, its usual correction capability is $t = \lfloor (d-1)/2 \rfloor = 6$.

A generator matrix of C in reduced echelon form is shown at the bottom of this page.

According to Theorem 2, we can correct up to $\tau = 5$ errors. In reality, the algorithm gives decoding up to $\tau = 7$ errors. Consider now a vector $y = (10, 6, 0, 16, 11, 0, 4, 8, 10, 9, 4, 0, 14, 9, 11, 12, 15)$. A polynomial satisfying the conditions of Theorem 3 of minimal degree is

$$G(x,T) = xT^2 + (11x^4 + 10x^3 + 7x^2 + 12x)T + 15x^9 + 12x^8 + 3x^7 + 10x^6 + 8x^5 + 7x^4 + 7x^3 + x^2 + x$$

and $(\partial G/\partial T)(x,T)$ does not vanish on (p_i, y_i) for

$$i \in S = \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}.$$

Consider position $i = 2$. We take the basis of the vector space of polynomials of degree less than $k = 5$ consisting in powers of $t_{P_2} = (x - 1)$, namely, $(f_{P_2,1} = 1, f_{P_2,2} = x + 16, f_{P_2,3} = x^2 + 15x + 1,$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 5 & 15 & 1 & 2 & 7 & 6 & 7 & 2 & 1 & 15 & 5 \\ 0 & 1 & 0 & 0 & 0 & 12 & 10 & 15 & 10 & 8 & 1 & 11 & 5 & 14 & 14 & 11 & 7 \\ 0 & 0 & 1 & 0 & 0 & 10 & 11 & 7 & 8 & 13 & 10 & 10 & 13 & 8 & 7 & 11 & 10 \\ 0 & 0 & 0 & 1 & 0 & 7 & 11 & 14 & 14 & 5 & 11 & 1 & 8 & 10 & 15 & 10 & 12 \\ 0 & 0 & 0 & 0 & 1 & 5 & 15 & 1 & 2 & 7 & 6 & 7 & 2 & 1 & 15 & 5 & 1 \end{pmatrix}$$

$f_{P_2,4} = x^3 + 14x^2 + 3x + 16, f_{P_2,5} = x^4 + 13x^3 + 6x^2 + 13x + 1)$. We now compute

$$\tilde{G}_{P_2,4} = G(x-1) = (1+t_{P_2})T^2 + (6+15t_{P_2}+t_{P_2}^2+3t_{P_2}^3+11t_{P_2}^4)T + 13 + 13t_{P_2} + 9t_{P_2}^2 + 6t_{P_2}^3 + 6t_{P_2}^4.$$

We start Newton's method with $\alpha = y_2 = 6$. The derivative of $\tilde{G}_{P_2,4}$ is

$$\tilde{G}'_{P_2,4} = (2 + 2t_{P_2})T + 6 + 15t_{P_2} + t_{P_2}^2 + 3t_{P_2}^3 + 11t_{P_2}^4.$$

We initialize

- $\eta \leftarrow \tilde{G}'_{P_2,4}(\alpha)^{-1} = 1$
- $\varphi \leftarrow \alpha = 6$.

We have to loop $\lceil \log_2(l_{P_2} + 1) \rceil = 3$ times and

For $j = 0$:

- $\eta = 1 + 7t_{P_2}$
- $\varphi = 6 + 14t_{P_2}$.

For $j = 1$:

- $\eta = 1 + 13t_{P_2} + 2t_{P_2}^2 + 7t_{P_2}^3$
- $\varphi = 6 + 14t_{P_2} + 6t_{P_2}^2 + 14t_{P_2}^3$.

For $j = 2$:

- $\eta = 1 + 13t_{P_2} + 9t_{P_2}^2 + 4t_{P_2}^4$
- $\varphi = 6 + 14t_{P_2} + 6t_{P_2}^2 + 14t_{P_2}^3 + 11t_{P_2}^4$.

The truncated power series φ is the Hensel development of a function

$$f = 6 \cdot f_{P_2,1} + 14 \cdot f_{P_2,2} + 6 \cdot f_{P_2,3} + 14 \cdot f_{P_2,4} + 11 \cdot f_{P_2,5} = 11x^4 + 4x^3 + 13x^2 + 12.$$

We have $\mathrm{ev}(f) = (12, 6, 0, 6, 11, 11, 11, 8, 8, 9, 1, 0, 14, 9, 11, 4, 15)$, which is at distance $d(c, y) = 7 \leq \tau$ from y, hence we include it in the list B. Moreover, we know that we can remove the set of indices $I = \{j \in S \mid f(P_j) = y_j\} = \{2, 3, 5, 8, 10, 12, 13, 14, 15\}$ from our investigation.

Consider position $i = 4$. We take the basis of the vector space of polynomials of degree less than $k = 5$ consisting in powers of $t_{P_4} = (x - 3)$, namely, $(f_{P_4,1} = 1, f_{P_4,2} = x + 14, f_{P_4,3} = x^2 + 11x + 9, f_{P_4,4} = x^3 + 8x^2 + 10x + 7, f_{P_4,5} = x^4 + 5x^3 + 3x^2 + 11x + 13)$. We now compute

$$\tilde{G}_{P_4,4} = G(x-3) = (3+t_{P_4})T^2 + (2+16t_{P_4}+11t_{P_4}^2+6t_{P_4}^3+11t_{P_4}^4)T + 16 + 16t_{P_4} + 3t_{P_4}^2 + t_{P_4}^3 + 15t_{P_4}^4.$$

We start Newton's method with $\alpha = y_4 = 16$. The derivative of $\tilde{G}_{P_4,4}$ is

$$\tilde{G}'_{P_4,4} = (6 + 2t_{P_4})T + 2 + 16t_{P_4} + 11t_{P_4}^2 + 6t_{P_4}^3 + 11t_{P_4}^4.$$

We initialize

- $\eta \leftarrow \tilde{G}'_{P_4,4}(\alpha)^{-1} = 4$
- $\varphi \leftarrow \alpha = 16$.

We have to loop $\lceil \log_2(l_{P_4} + 1) \rceil = 3$ times and

For $j = 0$:

- $\eta = 4 + 14t_{P_4}$
- $\varphi = 16 + 13t_{P_4}$.

For $j = 1$:

- $\eta = 4 + 7t_{P_4} + 3t_{P_4}^2 + 15t_{P_4}^3$
- $\varphi = 16 + 13t_{P_4} + 13t_{P_4}^2 + 6t_{P_4}^3$.

For $j = 2$:

- $\eta = 4 + 7t_{P_4} + 4t_{P_4}^2 + 6t_{P_4}^4$
- $\varphi = 16 + 13t_{P_4} + 13t_{P_4}^2 + 6t_{P_4}^3 + 6t_{P_4}^4$.

The truncated power series φ is the Hensel development of a function

$$f = 16 \cdot f_{P_4,1} + 13 \cdot f_{P_4,2} + 13 \cdot f_{P_4,3} + 6 \cdot f_{P_4,4} + 6 \cdot f_{P_4,5} = 6x^4 + 2x^3 + 11x^2 + 10x + 10.$$

We have

$$\mathrm{ev}(f) = (10, 5, 16, 16, 3, 0, 4, 3, 10, 12, 4, 6, 12, 7, 1, 12, 15)$$

which is at distance $d(c, y) = 9 > \tau$ from y. So we discard this candidate.

Moreover, we know that we can remove the set of indices

$$I = \{j \in S \mid f(P_j) = y_j\} = \{4, 6, 7, 9, 11, 16\}$$

from our investigation.

Finally, the list contains one codeword, namely,

- $c_1 = (12, 6, 0, 6, 11, 11, 11, 8, 8, 9, 1, 0, 14, 9, 11, 4, 15)$.

Appendix C
A $[64, 3, 29]_{16}$ Hermitian Code

We consider the field $K = \boldsymbol{F}_{16} = \boldsymbol{F}_2[\omega]$ where the primitive element ω has minimal polynomial $\omega^4 + \omega + 1$. We consider the absolutely irreducible affine curve $\mathcal{X}$ with defining polynomial $X^5 + Y^4 + Y$ over K. Its genus is $g = 6$. The curve has 64 points of degree 1, namely, $p_1 = (0, 0)$, $p_2 = (0, 1)$, $p_3 = (0, \omega^5)$, $p_4 = (0, \omega^{10})$, $p_5 = (1, \omega)$, $p_6 = (1, \omega^2)$, $p_7 = (1, \omega^4)$, $p_8 = (1, \omega^8)$, $p_9 = (\omega, \omega^6)$, $p_{10} = (\omega, \omega^7)$, $p_{11} = (\omega, \omega^9)$, $p_{12} = (\omega, \omega^{13})$, $p_{13} = (\omega^2, \omega^3)$, $p_{14} = (\omega^2, \omega^{11})$, $p_{15} = (\omega^2, \omega^{12})$, $p_{16} = (\omega^2, \omega^{14})$, $p_{17} = (\omega^3, \omega)$, $p_{18} = (\omega^3, \omega^2)$, $p_{19} = (\omega^3, \omega^4)$, $p_{20} = (\omega^3, \omega^8)$, $p_{21} = (\omega^4, \omega^6)$, $p_{22} = (\omega^4, \omega^7)$, $p_{23} = (\omega^4, \omega^9)$, $p_{24} = (\omega^4, \omega^{13})$, $p_{25} = (\omega^5, \omega^3)$, $p_{26} = (\omega^5, \omega^{11})$, $p_{27} = (\omega^5, \omega^{12})$, $p_{28} = (\omega^5, \omega^{14})$, $p_{29} = (\omega^6, \omega)$, $p_{30} = (\omega^6, \omega^2)$, $p_{31} = (\omega^6, \omega^4)$, $p_{32} = (\omega^6, \omega^8)$, $p_{33} = (\omega^7, \omega^6)$, $p_{34} = (\omega^7, \omega^7)$, $p_{35} = (\omega^7, \omega^9)$, $p_{36} = (\omega^7, \omega^{13})$, $p_{37} = (\omega^8, \omega^3)$, $p_{38} = (\omega^8, \omega^{11})$, $p_{39} = (\omega^8, \omega^{12})$, $p_{40} = (\omega^8, \omega^{14})$, $p_{41} = (\omega^9, \omega)$, $p_{42} = (\omega^9, \omega^2)$, $p_{43} = (\omega^9, \omega^4)$, $p_{44} = (\omega^9, \omega^8)$, $p_{45} = (\omega^{10}, \omega^6)$, $p_{46} = (\omega^{10}, \omega^7)$, $p_{47} = (\omega^{10}, \omega^9)$, $p_{48} = (\omega^{10}, \omega^{13})$, $p_{49} = (\omega^{11}, \omega^3)$, $p_{50} = (\omega^{11}, \omega^{11})$, $p_{51} = (\omega^{11}, \omega^{12})$, $p_{52} = (\omega^{11}, \omega^{14})$, $p_{53} = (\omega^{12}, \omega)$, $p_{54} = (\omega^{12}, \omega^2)$, $p_{55} = (\omega^{12}, \omega^4)$, $p_{56} = (\omega^{12}, \omega^8)$, $p_{57} = (\omega^{13}, \omega^6)$, $p_{58} = (\omega^{13}, \omega^7)$, $p_{59} = (\omega^{13}, \omega^9)$, $p_{60} = (\omega^{13}, \omega^{13})$, $p_{61} = (\omega^{14}, \omega^3)$, $p_{62} = (\omega^{14}, \omega^{11})$, $p_{63} = (\omega^{14}, \omega^{12})$, $p_{64} = (\omega^{14}, \omega^{14})$. Its projective closure also contains the point $p_\infty = (0 : 1 : 0)$.

All points are smooth and we denote by P_i the place corresponding to p_i for $1 \leq i \leq 64$, and by P_∞ the place over the point p_∞.

We define the divisor $D = 7 \cdot P_\infty$. A basis of the Riemann–Roch space $\mathcal{L}(D)$ is $(f_1 = 1, f_2 = X, f_3 = Y)$.

We build the Hermitian AG code with support $(P_1, \ldots, P_n)$ and divisor D. It is a $[64, 3]$-code. Its Goppa designed distance is 57 and its actual minimum distance is 59. Consequently, its usual correction capability is $t = \lfloor (d-1)/2 \rfloor = 29$.

According to Theorem 2, we can correct up to $\tau = 28$ errors. In reality, the algorithm gives decoding up to $\tau = 31$ errors. Consider now a vector

$$\begin{aligned} y = (&\omega^7, 0, \omega^2, \omega^{12}, 0, \omega^2, 0, \omega^{12}, \omega^6, \omega^3, \\ &\omega^{11}, \omega^{10}, \omega^7, \omega^7, \omega, 0, \omega^4, 0, \omega^7, \omega^{14}, \omega^{14}, \\ &\omega^{13}, \omega^3, \omega, 1, \omega^{10}, \omega^{11}, \omega^{13}, \omega^{10}, \omega^3, 1, \\ &\omega, \omega^{10}, \omega^4, \omega^3, \omega^6, \omega^{11}, \omega^{13}, \omega^2, \omega^8, \omega^{12}, \omega^8 \\ &\omega^5, \omega^8, \omega^{11}, \omega^9, \omega^4, \omega^{14}, 1, 0, \omega^7, \omega^2, \\ &\omega, 1, \omega^7, \omega^3, \omega^{14}, \omega^{11}, \omega^8, 1, \omega^{10}, \omega^4, \omega^4, \omega^{14}). \end{aligned}$$

A polynomial satisfying the conditions of Theorem 1 of minimal degree is

$$G(T) = \omega^{13}X^4T^2 + (\omega^2X^5 + \omega^{14}X^4Y + \omega^4X^4)T + \omega^{11}X^6 + X^5Y + \omega^8X^5 + \omega^4X^4Y^2 + \omega X^4Y + X^4$$

and its derivative does not vanish on P_i for

$$\begin{aligned} i \in S = \{&5,6,7,8,9,10,11,12,13,14,15,\\ &16,17,18,19,20,21,22,23,25,26,27,28,29,30,\\ &31,32,33,34,35,36,38,39,40,41,42,43,44,45,\\ &47,48,49,50,52,53,54,55,56,57,58,59,60,\\ &61,62,63,64\}. \end{aligned}$$

Consider position $i = 5$. A local parameter at place P_5 is $t_{P_5} = (X-1)$. A basis of $\mathcal{L}(D)$ in P_5-reduced echelon form is

$$(f_{P_5,1} = 1, f_{P_5,2} = X+1, f_{P_5,3} = X+Y+\omega^4).$$

The P_5-valuation sequence is $V_{P_5} = (0,1,5)$, and we have $l_{P_5} = 5$.

We now compute

$$\begin{aligned} \tilde{G}_{P_5,5} = &(\omega^{13} + \omega^{13} t_{P_5}^4)T^2 + (\omega^5 + \omega^{13} t_{P_5} + \omega^5 t_{P_5}^4 + \omega^2 t_{P_5}^5)T\\ &+\omega^2 t_{P_5} + \omega^6 t_{P_5}^2 + \omega^{10} t_{P_5}^5. \end{aligned}$$

We start Newton's method with $\alpha = y_5 = 0$. The derivative of $\tilde{G}_{P_5,5}$ is $\tilde{G}'_{P_5,5} = \omega^5 + \omega^{13} t_{P_5} + \omega^5 t_{P_5}^4 + \omega^2 t_{P_5}^5$. We initialize

- $\eta \leftarrow \tilde{G}'_{P_5,5}(\alpha)^{-1} = \omega^{10}$
- $\varphi \leftarrow \alpha = 0$.

We have to loop $\lceil \log_2(l_{P_5}+1) \rceil = 3$ times and

For $j = 0$:

- $\eta = \omega^{10} + \omega^3 t_{P_5}$
- $\varphi = \omega^{12} t_{P_5}$.

For $j = 1$:

- $\eta = \omega^{10} + \omega^3 t_{P_5} + \omega^{11} t_{P_5}^2 + \omega^4 t_{P_5}^3$
- $\varphi = \omega^{12} t_{P_5}$.

For $j = 2$:

- $\eta = \omega^{10} + \omega^3 t_{P_5} + \omega^{11} t_{P_5}^2 + \omega^4 t_{P_5}^3 + \omega^3 t_{P_5}^4 + \omega^{13} t_{P_5}^5$
- $\varphi = \omega^{12} t_{P_5} + \omega^{14} t_{P_5}^5$.

The truncated power series φ is the Hensel development of a function

$$f = \omega^{12} \cdot f_{P_5,2} + \omega^{14} \cdot f_{P_5,3} = \omega^5 X + \omega^{14} Y + \omega^{10}.$$

We have $\mathrm{ev}(f) = (\omega^{10}, \omega^{11}, \omega^2, \omega^{13}, 0, \omega^4, \omega^{14}, \omega^9, \omega^{13}, \omega^{10}, \omega^{11}, \omega^2, \omega^3, \omega^7, \omega, 1, \omega^4, 0, \omega^9, \omega^{14}, \omega^7, 1, \omega^3, \omega, \omega^2, \omega^{10}, \omega^{11}, \omega^{13}, \omega^3, \omega^7, 1, \omega, \omega^{11}, \omega^2, \omega^{13}, \omega^{10}, \omega^{11}, \omega^{13}, \omega^2, \omega^{10}, \omega^{12}, \omega^6, \omega^5, \omega^8, 0, \omega^9, \omega^4, \omega^{14}, 1, \omega, \omega^7, \omega^3, \omega, 1, \omega^7, \omega^3, \omega^{14}, \omega^4, \omega^9, 0, 0, \omega^4, \omega^9, \omega^{14})$, which is at distance $d(c,y) = 31 \le \tau$ from y, hence we include it in the list B. Moreover, we know that we can remove the set of indices

$$\begin{aligned} I &= \{j \in S \mid f(P_j) = y_j\}\\ &= \{5,11,14,15,17,18,20,23,26,\\ &\quad 27,28,31,32,38,39,41,43,44,\\ &\quad 47,48,49,53,54,55,56,57,62,64\} \end{aligned}$$

from our investigation.

Consider position $i = 6$. A local parameter at place P_6 is $t_{P_6} = (X-1)$. A basis of $\mathcal{L}(D)$ in P_6-reduced echelon form is $(f_{P_6,1} = 1, f_{P_6,2} = X+1, f_{P_6,3} = X+Y+\omega^8)$. The P_6-valuation sequence is $V_{P_6} = (0,1,5)$, and we have $l_{P_6} = 5$.

We now compute

$$\begin{aligned} \tilde{G}_{P_6,5} = &(\omega^{13} + \omega^{13} t_{P_6}^4)T^2 + (\omega^8 + \omega^{13} t_{P_6} + \omega^8 t_{P_6}^4 + \omega^2 t_{P_6}^5)T\\ &+\omega^4 + \omega t_{P_6} + \omega^6 t_{P_6}^2 + \omega^4 t_{P_6}^4 + t_{P_6}^5. \end{aligned}$$

We start Newton's method with $\alpha = y_6 = \omega^2$. The derivative of $\tilde{G}_{P_6,5}$ is $\tilde{G}'_{P_6,5} = \omega^8 + \omega^{13} t_{P_6} + \omega^8 t_{P_6}^4 + \omega^2 t_{P_6}^5$. We initialize

- $\eta \leftarrow \tilde{G}'_{P_6,5}(\alpha)^{-1} = \omega^7$
- $\varphi \leftarrow \alpha = \omega^2$.

We have to loop $\lceil \log_2(l_{P_6}+1) \rceil = 3$ times and

For $j = 0$:

- $\eta = \omega^7 + \omega^{12} t_{P_6}$
- $\varphi = \omega^2 + \omega^{11} t_{P_6}$.

For $j = 1$:

- $\eta = \omega^7 + \omega^{12} t_{P_6} + \omega^2 t_{P_6}^2 + \omega^7 t_{P_6}^3$
- $\varphi = \omega^2 + \omega^{11} t_{P_6}$.

For $j = 2$:

- $\eta = \omega^7 + \omega^{12} t_{P_6} + \omega^2 t_{P_6}^2 + \omega^7 t_{P_6}^3 + \omega^2 t_{P_6}^4 + \omega^5 t_{P_6}^5$
- $\varphi = \omega^2 + \omega^{11} t_{P_6} + \omega^7 t_{P_6}^5$.

The truncated power series φ is the Hensel development of a function $f = \omega^2 \cdot f_{P_6,1} + \omega^{11} \cdot f_{P_6,2} + \omega^7 \cdot f_{P_6,3} = \omega^8 X + \omega^7 Y + \omega^7$. We have $\mathrm{ev}(f) = (\omega^7, 0, \omega^2, \omega^{12}, \omega^7, \omega^2, 0, \omega^{12}, \omega^6, \omega^3, \omega^4, \omega^{10}, \omega^7, \omega^2, \omega^{12}, 0, 0, \omega^{12}, \omega^7, \omega^2, \omega^{14}, \omega^{13}, \omega^5, \omega, 1, \omega^{11}, \omega^8, \omega^9, \omega^{10}, \omega^3, \omega^6, \omega^4, \omega^{10}, \omega^4, \omega^3, \omega^6, \omega^{11}, 1, \omega^9, \omega^8, \omega^9, \omega^8, 1, \omega^{11}, \omega^{11}, \omega^9, 1, \omega^8, \omega^{12}, 0, \omega^7, \omega^2, \omega^3, \omega^{10}, \omega^4, \omega^6, \omega^9, \omega^{11}, \omega^8, 1, \omega^{10}, \omega^3, \omega^4, \omega^6)$, which is at distance $d(c,y) = 28 \le \tau$ from y, hence we include it in the list B. Moreover, we know that we can remove the set of indices

$$\begin{aligned} I &= \{j \in S \mid f(P_j) = y_j\}\\ &= \{6,7,8,9,10,12,13,16,19,21,22,25,29,30,\\ &\quad 33,34,35,36,40,42,45,50,52,58,59,60,61,63\} \end{aligned}$$

from our investigation.

Finally, the list contains two codewords, namely,

- $c_1 = (\omega^7, 0, \omega^2, \omega^{12}, \omega^7, \omega^2, 0, \omega^{12}, \omega^6, \omega^3, \omega^4, \omega^{10}, \omega^7, \omega^2, \omega^{12}, 0, 0, \omega^{12}, \omega^7, \omega^2, \omega^{14}, \omega^{13}, \omega^5, \omega, 1, \omega^{11}, \omega^8, \omega^9, \omega^{10}, \omega^3, \omega^6, \omega^4, \omega^{10}, \omega^4, \omega^3, \omega^6, \omega^{11}, 1, \omega^9, \omega^8, \omega^9, \omega^8, 1, \omega^{11}, \omega^{11}, \omega^9, 1, \omega^8, \omega^{12}, 0, \omega^7, \omega^2, \omega^3, \omega^{10}, \omega^4, \omega^6, \omega^9, \omega^{11}, \omega^8, 1, \omega^{10}, \omega^3, \omega^4, \omega^6)$
- $c_2 = (\omega^{10}, \omega^{11}, \omega^2, \omega^{13}, 0, \omega^4, \omega^{14}, \omega^9, \omega^{13}, \omega^{10}, \omega^{11}, \omega^2, \omega^3, \omega^7, \omega, 1, \omega^4, 0, \omega^9, \omega^{14}, \omega^7, 1, \omega^3, \omega, \omega^2, \omega^{10}, \omega^{11}, \omega^{13}, \omega^3, \omega^7, 1, \omega, \omega^{11}, \omega^2, \omega^{13}, \omega^{10}, \omega^{11}, \omega^{13}, \omega^2, \omega^{10}, \omega^{12}, \omega^6, \omega^5, \omega^8, 0, \omega^9, \omega^4, \omega^{14}, 1, \omega, \omega^7, \omega^3, \omega, 1, \omega^7, \omega^3, \omega^{14}, \omega^4, \omega^9, 0, 0, \omega^4, \omega^9, \omega^{14})$.

ACKNOWLEDGMENT

The authors wish to thank the two anonymous referees who have sent a very complete report on our work, including many helpful remarks.

REFERENCES

[1] N. Bourbaki, *Commutative Algebra*. New York: Springer-Verlag, 1989, ch. 1–7.

[2] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon codes and algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1757–1767, Sept. 1999.

[3] T. Høholdt and R. R. Nielsen, "Decoding Hermitian codes with Sudan's algorithm," in *Proc. AAECC-13 (Lecture Notes in Computer Science)*. Berlin, Germany: Springer-Verlag, 1999, vol. 1719, pp. 260–270.

[4] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1988.

[5] J. Neukirch, *Algebraic Number Theory*. Berlin, Germany: Springer-Verlag, 1999.

[6] V. Olshevsky and A. Shokrollahi, "A displacement structure approach to efficient decoding of Reed-Solomon and algebraic-geometric codes," in *Proc. STOC 99*, 1999, to be published.

[7] R. Pellikaan, B. Z. Shen, and G. J. M. van Wee, "Which codes are algebraic-geometric?," *IEEE Trans. Inform. Theory*, vol. 37, pp. 583–602, May 1991.

[8] M. A. Shokrollahi and H. Wasserman, "List decoding of algebraic-geometric codes," *IEEE Trans. Inform. Theory*, vol. 45, pp. 432–437, Mar. 1999.

[9] H. Stichtenoth, *Algebraic Function Fields and Codes*. Berlin, Germany: Springer-Verlag, 1993.

[10] M. Sudan, "Decoding of Reed-Solomon codes beyond the error-correction bound," *J. Complexity*, vol. 13, pp. 180–193, 1997.
[11] ——, "Decoding Reed-Solomon codes beyond the error-correction diameter," in *Proc. 35th Annu. Allerton Conf. Communication, Control and Computing*, 1997.
[12] M. A. Tsfasman and S. G. Vlăduţ, "Algebraic-geometric codes," in *Mathematics and Its Applications*. Norwell, MA: Kluwer , 1991.
[13] J. von zur Gathen and J. Gerhard, *Computer Algebra*. Cambridge, U.K.: Cambridge Univ. Press, 1999.

On the Equivalence of the Berlekamp–Massey and the Euclidean Algorithms for Decoding

Agnes E. Heydtmann and Jørn M. Jensen

***Abstract*—The Berlekamp–Massey algorithm (BMA) and the Euclidean algorithm (EA) for decoding have been considered as two different algorithms for solving the same problem, namely, the one given by the key equation. In this correspondence we argue that they are essentially identical by showing how one can be adapted to perform the same arithmetics as the other. As a tool we use Feng and Tzeng's Fundamental Iterative algorithm that when adapted to the syndrome matrix is regarded as equivalent to the BMA.**

***Index Terms*—Alternant codes, Berlekamp–Massey algorithm (BMA), decoding, Euclidean algorithm (EA), fundamental iterative algorithm, key equation, Reed–Solomon codes, syndromes.**

I. Introduction

Decoding alternant codes such as Bose–Chaudhuri–Hocquenghem (BCH), Reed–Solomon, and Goppa codes consists essentially of solving the so-called key equation. Both the Berlekamp–Massey algorithm (BMA) [1], [2] and the Euclidean algorithm for decoding (EA) by Sugiyama *et al.* [3] serve this purpose in seemingly different ways, but whereas the first is more efficient (compare Sections III-E and IV-C), the latter has been considered simpler. Dornstetter [4] uncovered some of the parallels of the two algorithms, but still many think of these by now adopted as classical decoding algorithms as two different methods for solving one problem. Later, Feng and Tzeng developed the Fundamental Iterative Algorithm (FIA) [5]. This algorithm can manipulate matrices in a general setting, but when used on the syndrome matrix—and we will use the name FIA in this sense—its equivalence to the BMA is widely accepted. However, the FIA is of great tutorial value as it basically performs a kind of Gaussian elimination on the syndrome matrix which makes it accessible to the intuition of the reader. In our opinion, the BMA is much easier to comprehend and to work with in the version of the FIA, and we will make use of that when demonstrating the equivalence. Through the usage of the FIA it will become apparent that every single coefficient in the treated syndrome matrix and the different versions of the locator polynomial in the BMA correspond to coefficients of the various polynomials of the repeated divisions performed by the EA, and for every step in one algorithm there is a corresponding one in the other.

In order to show which parts of the two algorithms correspond precisely to each other, the FIA is extended to compute the error evaluator and coevaluator simultaneously to the error locator. This increases its complexity, of course, but it makes the FIA equivalent to and as complex as the EA.

Further, it is shown how in the EA some steps can be eliminated such that it either calculates the error evaluator or the error coevaluator. This way, the EA becomes equivalent to and as complex as the FIA extended by either the error evaluator or coevaluator and thus equivalent and as complex as the original BMA. Dornstetter [4] also eliminated steps from the EA such that it became equivalent to the BMA. However, it is not described in that article that the removed steps correspond to the removal of the calculation of the error coevaluator (which is where his and the present work coincide), but instead, it is claimed that both algorithms compute the same thing, whereas one is more complex than the other.

This correspondence has been written in the hope of being fairly self-contained, focusing on the intuition of every step. Section II introduces notation and the setting. Section III presents the BMA in the version of the FIA and the extension to the calculation of the error evaluator and coevaluator in addition to the error locator. In Section IV, the EA is adapted in two steps to increasingly resemble the BMA and we discuss which parts of the two algorithms correspond to each other. Section V is reserved for some remarks on the changes made and conclusions will be drawn in Section VI. The Appendix contains the proofs of the two main theorems.

Simulation of the discussed decoding processes has been done with the computer algebra system SINGULAR [6].

II. Preliminaries

For simplicity, we will consider a narrow-sense Reed–Solomon code $\mathcal{RS}$ instead of alternant codes to which all results can easily be generalized. Let $\mathcal{RS}$ be of designed minimal distance $2t+1$ over the finite field $\mathbb{F}_q$. The block length is $n = q-1$ and a primitive element $\alpha \in \mathbb{F}_q$ is chosen. The generator polynomial g of $\mathcal{RS}$ is of the form

$$g = (x-\alpha)(x-\alpha^2)\dots(x-\alpha^{2t})$$

and the code is the ideal

$$\mathcal{RS} = \langle g \rangle = \{ fg \bmod x^n - 1 \mid f \in \mathbb{F}_q[x] \}$$

in the factor ring $\mathbb{F}_q[x]/\langle x^n - 1 \rangle$. By the degree of g, the dimension is $k = n - 2t$ and $\mathcal{RS}$ is a maximum-distance-separable code. For Reed–Solomon codes in general see, for example, MacWilliams and Sloan [7].

Suppose the codeword $c = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$ is sent but $r = c + e = r_0 + r_1 x + \cdots + r_{n-1}x^{n-1}$ is received with error $e = e_0 + e_1 x + \cdots + e_{n-1}x^{n-1}$. The goal is to decode r to c which involves finding e. We assume that the number of errors ϵ is bounded by t and suppose errors occurred in locations $l_1, \dots, l_\epsilon$. The *syndromes* are then the elements

$$\begin{aligned} S_i &= r(\alpha^{i+1}) = c(\alpha^{i+1}) + e(\alpha^{i+1}) \\ &\overset{(*)}{=} e(\alpha^{i+1}) = \sum_{j=1}^{\epsilon} e_{l_j} \alpha^{(i+1)l_j}, \qquad 0 \le i \le 2t-1 \end{aligned}$$

where $(*)$ holds by the fact that $c \in \langle g \rangle$. The *syndrome polynomial* is

$$S = S_0 + S_1 x + \cdots + S_{2t-1}x^{2t-1}.$$

Manuscript received April 5, 1999; revised November 16, 1999 and May 4, 2000. The work of A. E. Heydtmann was supported by a graduate student scholarship (HSPIII) of the German Academic Exchange Service.

The authors are with the Department of Mathematics, Technical University of Denmark, DK-2800 Lyngby, Denmark (e-mail: agnes@mat.dtu.dk; j.m.jensen@mat.dtu.dk).

Communicated by R. M. Roth, Associate Editor for Coding Theory.
Publisher Item Identifier S 0018-9448(00)09657-7.

We define further the *error locator polynomial*

$$\begin{aligned}\Lambda &= \prod_{i=1}^{\epsilon}(1-\alpha^{l_j}x)\\ &= \Lambda_\epsilon x^\epsilon + \cdots + \Lambda_1 x + \Lambda_0\end{aligned}$$

the *error evaluator polynomial*

$$\begin{aligned}\Omega &= \sum_{j=1}^{\epsilon} e_{l_j}\alpha^{l_j}\prod_{\substack{i=1\\ i\neq j}}^{\epsilon}(1-\alpha^{l_i}x)\\ &= \Omega_{\epsilon-1}x^{\epsilon-1} + \cdots + \Omega_1 x + \Omega_0\end{aligned}$$

and what Pretzel [8, pp. 244, 274] calls the *error coevaluator polynomial*

$$\begin{aligned}\Psi &= \sum_{j=1}^{\epsilon} e_{l_j}\alpha^{(2t+1)l_j}\prod_{\substack{i=1\\ i\neq j}}^{\epsilon}(1-\alpha^{l_i}x)\\ &= \Psi_{\epsilon-1}x^{\epsilon-1} + \cdots + \Psi_1 x + \Psi_0.\end{aligned}$$

Note that $\Lambda_0 = 1$. It is obvious that Λ and Ω as well as Λ and Ψ do not have any common factors. It is readily verified that the four polynomials S, Λ, Ω, and Ψ are related by the *key equation*

$$\Lambda \cdot S = \Omega - \Psi x^{2t} \tag{1}$$

or more commonly

$$\Lambda \cdot S \equiv \Omega \bmod x^{2t}. \tag{1'}$$

Finding Λ will give us the error locations and enables us to find Ω and/or Ψ easily. With the help of one of Ω or Ψ, error values can then be obtained with the Forney algorithm

$$e_{l_i} = -\frac{\Omega(\alpha^{-l_i})}{\Lambda'(\alpha^{-l_i})} = -\frac{\Psi(\alpha^{-l_i})}{\alpha^{2tl_i}\Lambda'(\alpha^{-l_i})} \tag{2}$$

where

$$\Lambda' = -\sum_{j=1}^{\epsilon}\alpha^{l_j}\prod_{i=1\,i\neq j}^{\epsilon}(1-\alpha^{l_i}x)$$

is the formal derivative of Λ. In this way, we get all components of e. Considering (1) as an equation in three unknowns Λ, Ω, and Ψ, there are obviously many solutions to it. However, constraining it by the known facts about the desired polynomials, namely, $\gcd(\Lambda,\Omega)=1$, $\gcd(\Lambda,\Psi)=1$, $\Lambda(0)=1$, $\deg\Lambda \le t$, $\deg\Omega < t$, and $\deg\Psi < t$, the solution is unique—compare with Pretzel [8, pp. 256–258].

Therefore, in decoding which consists in finding $l_1,\ldots,l_\epsilon$ and $e_{l_1},\ldots,e_{l_\epsilon}$ accordingly, the following three steps are involved:

- calculating the syndromes S_i, $0 \le i \le 2t-1$;
- given the syndrome polynomial S, finding the error locator Λ and error evaluator Ω usually with the BMA, see, for example, Berlekamp [1] and Massey [2], or the EA by Sugiyama *et al.* [3] which also calculates the error coevaluator Ψ;
- given Λ and Ω or Ψ, computing the actual error locations by a random search of roots of Λ and then values by the Forney algorithm (2).

In this correspondence, we are focusing on the computationally most expensive part of solving the key equation (1).

III. The Berlekamp–Massey Algorithm

A. A Matrix Approach to the Key Equation

We can rephrase (1) in matrix form as

$$\begin{pmatrix} 0 & \cdots & 0 & S_0\\ \vdots & \iddots & \iddots & \vdots\\ 0 & S_0 & \cdots & S_{\epsilon-1}\\ S_0 & S_1 & \cdots & S_\epsilon\\ S_1 & S_2 & \cdots & S_{\epsilon+1}\\ \vdots & \vdots & & \vdots\\ S_{2t-\epsilon-1} & S_{2t-\epsilon} & \cdots & S_{2t-1}\\ \vdots & & \iddots & 0\\ & & \iddots & \vdots\\ S_{2t-1} & 0 & \cdots & 0 \end{pmatrix}\begin{pmatrix}\Lambda_\epsilon\\ \Lambda_{\epsilon-1}\\ \vdots\\ \Lambda_1\\ 1\end{pmatrix} = \begin{pmatrix}\Omega_0\\ \vdots\\ \Omega_{\epsilon-1}\\ 0\\ 0\\ \vdots\\ 0\\ -\Psi_0\\ \vdots\\ -\Psi_{\epsilon-1}\end{pmatrix}. \tag{3}$$

The terms corresponding to $x^\epsilon,\ldots,x^{2t-1}$ in the product $\Lambda\cdot S$ are 0 and, as will be seen, this information is already enough to find Λ. Consider the middle part of (3)

$$\begin{pmatrix} S_0 & S_1 & \cdots & S_\epsilon\\ S_1 & S_2 & \cdots & S_{\epsilon+1}\\ \vdots & \vdots & & \vdots\\ S_{2t-\epsilon-1} & S_{2t-\epsilon} & \cdots & S_{2t-1}\end{pmatrix}\begin{pmatrix}\Lambda_\epsilon\\ \Lambda_{\epsilon-1}\\ \vdots\\ \Lambda_1\\ 1\end{pmatrix} = \begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix}. \tag{3'}$$

The coefficients of Λ give a linear dependence of the column number $\epsilon+1$ in (3′) in terms of the previous columns. However ϵ is also an unknown and instead we study the $t\times(t+1)$ *syndrome matrix*

$$\boldsymbol{S} = \begin{pmatrix} S_0 & S_1 & \cdots & S_\epsilon & \cdots & S_t\\ S_1 & S_2 & \cdots & S_{\epsilon+1} & \cdots & S_{t+1}\\ \vdots & \vdots & & \vdots & & \vdots\\ S_{t-1} & S_t & \cdots & S_{t+\epsilon-1} & \cdots & S_{2t-1}\end{pmatrix}. \tag{4}$$

Comparing with the matrix in (3′) shows that the first $\epsilon+1$ columns are linearly dependent in $\boldsymbol{S}$ and the two matrices are identical if $\epsilon = t$.

By $\lambda = \lambda_{\nu-1}x^{\nu-1}+\cdots+\lambda_1 x+\lambda_0$ describing a linear combination of the first ν columns $\boldsymbol{A}_1,\ldots,\boldsymbol{A}_\nu$ of a matrix $\boldsymbol{A}$, we mean

$$\lambda_{\nu-1}\boldsymbol{A}_1 + \cdots + \lambda_1\boldsymbol{A}_{\nu-1} + \lambda_0\boldsymbol{A}_\nu.$$

By such λ describing a linear dependence of the first ν columns of a matrix $\boldsymbol{A}$, we naturally mean that the above linear combination is 0. Theorem 1 shows how to find Λ.

Theorem 1: Any $\lambda = \lambda_\epsilon x^\epsilon + \cdots + \lambda_1 x + \lambda_0$ describing a linear dependence of the leading $\epsilon+1$ columns $\boldsymbol{S}_1,\ldots,\boldsymbol{S}_{\epsilon+1}$ of the syndrome matrix $\boldsymbol{S}$ equals the error locator polynomial Λ up to a constant factor.

The proof is left to the reader.

Thus any algorithm that produces a linear dependence between as few initial columns of the syndrome matrix $\boldsymbol{S}$ as possible can easily be extended to finding the error locator polynomial Λ. Once Λ is found, Ω or Ψ can be obtained easily and thus (3), as well as the key equation (1), can be solved by such an approach.

The next section presents Feng and Tzeng's algorithm [5] that finds a linear dependence—if there is one—of as few initial columns as possible in an arbitrary matrix. This algorithm will then be adapted to the special case of the syndrome matrix.

Input: matrix $\mathbf{A} = (a_{\mu\nu})_{1\le\mu\le\mu_{\max},1\le\nu\le\nu_{\max}}$
Output: if possible $\lambda = \lambda_d x^d + \cdots + \lambda_1 x + \lambda_0$ with $\lambda_0 = 1$ describing a linear dependence between the lowest number $d+1$ of linearly dependent initial columns of $\mathbf{A}$

Start with row index $\mu_1 = 1$ and column index $\nu = 1$.
Initialize $\lambda^{(1)} = 1$.
repeat
  Calculate discrepancy

$$\delta = \sum_{j=0}^{\nu-1} \lambda_j^{(\nu)} a_{\mu_\nu\, \nu-j}.$$

  if $\delta \neq 0$ **then**
    if $\mu_\nu = \mu_j$ for some $1 \le j < \nu$ **then**
      Update $\lambda^{(\nu)}$ to

$$\lambda^{(\nu)} - \frac{\delta}{\delta_j}\lambda^{(j)} x^{\nu-j}.$$

      Increment μ_ν by 1.
    else $\{\mu_\nu \neq \mu_j$ for all $1 \le j < \nu\}$
      Set $\delta_\nu = \delta$.
      Increment ν by 1 and set $\mu_\nu = 1$.
      Initialize $\lambda^{(\nu)} = 1$.
    end if
  else $\{\delta = 0\}$
    Increment μ_ν by 1.
  end if
until $\mu_\nu > \mu_{\max}$ or $\nu > \nu_{\max}$
if $\mu_\nu > \mu_{\max}$ **then**
  return $\lambda^{(\nu)}$
end if

Fig. 1. The FIA.

B. The FIA

Let $\boldsymbol{A}$ be an arbitrary matrix with columns $\boldsymbol{A}_i$. In order to find the smallest number of linearly dependent initial columns of $\boldsymbol{A}$ one can perform a kind of Gaussian elimination on $\boldsymbol{A}$. Start with the first column. By subtracting multiples of previous columns from the current column obtain columns with as many initial zeros as possible until one column consists entirely of zeros. An example of the result of such a process could be the matrix

$$\begin{pmatrix} \delta_1 & 0 & 0 & 0 & 0 & 0 & 0 & \\ & \delta_2 & 0 & 0 & 0 & 0 & 0 & \\ & & 0 & 0 & \delta_5 & 0 & 0 & \\ & & 0 & \delta_4 & & 0 & 0 & \boldsymbol{A}_8 \\ & & \delta_3 & & & 0 & 0 & \\ & & & & & \delta_6 & 0 & \\ & & & & & & 0 & \end{pmatrix}$$

where the δ_j's stand for nonzero entries and the empty components for arbitrary ones. Suppose in column number j all nonzero entries could be eliminated. By the involved process it is clear that the original columns $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{j-1}$ are linearly independent, that columns $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_j$ are linearly dependent and the linear dependence is found by keeping track of the operations.

This describes the FIA developed by Feng and Tzeng [5] and a formal description is given in Fig. 1. Note that the linear combinations and thereby also the final linear dependency are given in terms of a polynomial. This is motivated by the fact that we want to use the FIA on the syndrome matrix when finding the error locator polynomial.

To understand the formula updating $\lambda^{(\nu)}$, suppose the FIA reached column number ν when performed on matrix $\boldsymbol{A}$ with entries $a_{\mu\nu}$. At that point each

$$\lambda^{(j)} = \lambda_{j-1}^{(j)} x^{j-1} + \cdots + \lambda_1^{(j)} x + \lambda_0^{(j)}, \qquad 1 \le j \le \nu$$

describes the linear combination

$$\tilde{\boldsymbol{A}}_j = \lambda_{j-1}^{(j)} \boldsymbol{A}_1 + \cdots + \lambda_1^{(j)} \boldsymbol{A}_{j-1} + \lambda_0^{(j)} \boldsymbol{A}_j \neq 0, \qquad 1 \le j \le \nu \tag{5}$$

with $\lambda_0^{(j)} = 1$. Note that if no initial entries in the νth column have been eliminated then $\lambda^{(\nu)} = 1$ and that, in general, $\lambda^{(j)}(0) = 1, 1 \le j \le \nu$. By subtracting multiples of the $\tilde{\boldsymbol{A}}_j$ as many initial entries in column $\tilde{\boldsymbol{A}}_\nu$ as possible should be eliminated. Suppose $\tilde{\boldsymbol{A}}_\nu$ has initial zeros up to the μ_νth entry

$$\delta = \sum_{j=0}^{\nu-1} \lambda_j^{(\nu)} a_{\mu_\nu \nu-j} \neq 0$$

which we call a *discrepancy* from zero in row μ_ν. In order to eliminate δ, a multiple of $\tilde{\boldsymbol{A}}_j$ which has the same number of initial zeros and then a discrepancy $\delta_j \neq 0$ in row $\mu_j = \mu_\nu$, should be subtracted. The new $\tilde{\boldsymbol{A}}_\nu$ is thereby going to be

$$\begin{aligned} \tilde{\boldsymbol{A}}_\nu - \frac{\delta}{\delta_j}\tilde{\boldsymbol{A}}_j &\overset{(5)}{=} \lambda_{\nu-1}^{(\nu)} \boldsymbol{A}_1 + \cdots + \lambda_1^{(\nu)} \boldsymbol{A}_{\nu-1} + \lambda_0^{(\nu)} \boldsymbol{A}_\nu \\ &\quad - \frac{\delta}{\delta_j}\left(\lambda_{j-1}^{(j)} \boldsymbol{A}_1 + \cdots + \lambda_1^{(j)} \boldsymbol{A}_{j-1} + \lambda_0^{(j)} \boldsymbol{A}_j\right) \\ &= \left(\lambda_{\nu-1}^{(\nu)} - \frac{\delta}{\delta_j}\lambda_{j-1}^{(j)}\right)\boldsymbol{A}_1 + \cdots + \left(\lambda_{\nu-j}^{(\nu)} - \frac{\delta}{\delta_j}\lambda_0^{(j)}\right)\boldsymbol{A}_j \\ &\quad + \lambda_{\nu-j-1}^{(\nu)} \boldsymbol{A}_{j+1} + \cdots + \lambda_1^{(\nu)} \boldsymbol{A}_{\nu-1} + \lambda_0^{(\nu)} \boldsymbol{A}_\nu. \end{aligned}$$

The polynomial $\lambda^{(\nu)}$ should then be updated as follows:

$$\begin{aligned} &\left(\lambda_{\nu-1}^{(\nu)} - \frac{\delta}{\delta_j}\lambda_{j-1}^{(j)}\right) x^{\nu-1} + \cdots + \left(\lambda_{\nu-j}^{(\nu)} - \frac{\delta}{\delta_j}\lambda_0^{(j)}\right) x^{\nu-j} \\ &\quad + \lambda_{\nu-j-1}^{(\nu)} x^{\nu-j-1} + \cdots + \lambda_1^{(\nu)} x + \lambda_0^{(\nu)} \\ &\quad = \lambda_{\nu-1}^{(\nu)} x^{\nu-1} + \cdots + \lambda_1^{(\nu)} x + \lambda_0^{(\nu)} \\ &\qquad - \frac{\delta}{\delta_j}\left(\lambda_{j-1}^{(j)} x^{j-1} + \cdots + \lambda_1^{(j)} x + \lambda_0^{(j)}\right) x^{\nu-j} \\ &\quad = \lambda^{(\nu)} - \frac{\delta}{\delta_j}\lambda^{(j)} x^{\nu-j} \end{aligned}$$

which is exactly what is done in Fig. 1. The factor $x^{\nu-j}$ can be thought as "aligning" the two polynomials $\lambda^{(\nu)}$ and $\lambda^{(j)}$ such that the latter can "mend the defect" of the first. The above should have made apparent why it is not actually necessary to find the linear combinations of the column vectors $\boldsymbol{A}_j$. Calculating discrepancies and λ-polynomials as in Fig. 1 is enough and once all nonzero entries could be eliminated in one column, the algorithm ends with a polynomial that describes the corresponding linear dependence.

C. Adjustment of the Fundamental Iterative Algorithm to the Syndrome Matrix

We want to make use of the FIA in finding the error locator polynomial. When this algorithm is applied to the syndrome matrix, a linear dependence will be found in column $\epsilon+1$ by Section III-A, and the error locator polynomial Λ will be the output since it operates in such a way that division by a constant factor is not necessary. Recall, however, that the syndrome matrix has a special structure, a so-called Hankel form, which the above algorithm does not take into account.

Let us investigate. From now on let $\boldsymbol{S}$ denote a nontrivial syndrome matrix, i.e., there occurred $0 < \epsilon \le t$ errors and by the key equation (1), the first column of $\boldsymbol{S}$ is not the zero vector. Suppose the FIA is run

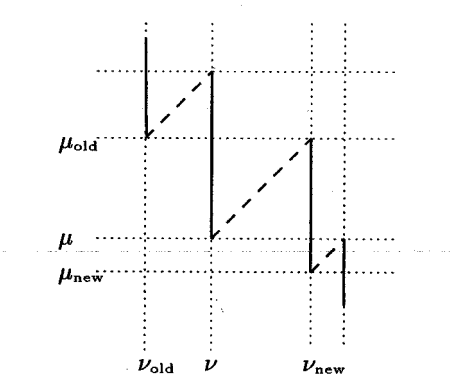

Fig. 2. Illustration of the adjusted FIA in column ν. Dotted lines represent certain rows and columns of the syndrome matrix, respectively. The solid lines indicate where discrepancies are computed, the diagonal dashed lines mark jumps to new columns.

on $\boldsymbol{S}$ and stalls in column ν, row μ with polynomial Λ. By "stalling" in row μ we mean that the Λ-polynomial can eliminate $\mu-1$ initial entries in the respective column, but not the nonzero discrepancy δ in row μ—compare with Fig. 2. We go to the next column $\nu+1$. Note that the linear combination of columns $\nu+1,\ldots,2$ defined by Λ gives a vector where $\mu-2$ initial entries could be eliminated, but in row $\mu-1$ we get the same discrepancy δ, because of the structure of $\boldsymbol{S}$. So it is sensible to reuse Λ instead of restarting with the constant polynomial 1 as the original FIA does. Now either the algorithm stalled in row $\mu-1$ before such that we can eliminate discrepancy δ and update or it did not, in which case we can move to the next column and use Λ again to eliminate $\mu-3$ initial entries in column $\nu+2$, but get discrepancy δ in row $\mu-2$. In this way, the algorithm finds a column where it can eliminate this discrepancy δ. We call those versions of the error locator, discrepancies, column and row indices Λ^{old}, δ_{old}, ν_{old}, and μ_{old} when the FIA has previously stalled in the respective column ν_{old}, row μ_{old} such that $\mu_{\text{old}}<\mu$ and the algorithm has never stalled in a row between μ_{old} and the current μ. It turns out that a jump to column $\nu+\mu-\mu_{\text{old}}$ is going to be successful in the way that discrepancy δ in row μ_{old} of this column can be eliminated by subtracting a multiple of $\tilde{\boldsymbol{S}}_{\nu_{\text{old}}}$.

The following theorem formalizes these observations.

Theorem 2: Let the FIA be run on syndrome matrix $\boldsymbol{S}$. Suppose it stalls with polynomial Λ in column number ν with zero entries in $\mu-1\geq\mu_{\text{old}}$ initial positions and

$$\delta=\sum_{i=0}^{\nu-1}\Lambda_i S_{\nu+\mu-i-2}\neq 0$$

as discrepancy in row μ which cannot be eliminated, and it stalled previously in column ν_{old}, row μ_{old} with polynomial Λ^{old} and discrepancy δ_{old} such that $\mu_{\text{old}}<\mu$ and the algorithm has never stalled in a row between μ_{old} and μ. Then the $\nu+\mu-\mu_{\text{old}}-1$ initial columns of $\boldsymbol{S}$ are linearly independent and in the $\nu_{\text{new}}=\nu+\mu-\mu_{\text{old}}$th column $\boldsymbol{S}_{\nu_{\text{new}}}$ nonzero entries in the μ initial rows can be eliminated in the following way:

1) the initial $\mu_{\text{old}}-1$ entries by setting $\Lambda^{\text{new}}=\Lambda$ and $\tilde{\boldsymbol{S}}_{\nu_{\text{new}}}$ to the corresponding linear combination;
2) the μ_{old}-th entry $\delta\neq 0$ by updating Λ^{new} and correspondingly $\tilde{\boldsymbol{S}}_{\nu_{\text{new}}}$ to

$$\Lambda^{\nu_{\text{new}}}-\frac{\delta}{\delta_{\text{old}}}\Lambda^{\text{old}}x^{\nu_{\text{new}}-\nu_{\text{old}}}\quad\text{and}\quad\tilde{\boldsymbol{S}}_{\nu_{\text{new}}}-\frac{\delta}{\delta_{\text{old}}}\tilde{\boldsymbol{S}}_{\nu_{\text{old}}};\quad(6)$$

3) the $\mu_{\text{old}}+1$ up to μth entry by finding nonzero discrepancies

$$\delta_{\text{new}}=\sum_{i=0}^{\nu_{\text{new}}-1}\Lambda_i^{\text{new}}S_{\nu+\mu_{\text{new}}-i-2}\quad\mu_{\text{old}}+1\leq\mu_{\text{new}}\leq\mu$$

and then eliminating them by updating Λ^{new} and correspondingly $\tilde{\boldsymbol{S}}_{\nu_{\text{new}}}$ to

$$\Lambda^{\text{new}}-\frac{\delta_{\text{new}}}{\delta}\Lambda x^{\mu_{\text{new}}-\mu_{\text{old}}}$$

$$\text{and}\quad\tilde{\boldsymbol{S}}_{\nu_{\text{new}}}-\frac{\delta_{\text{new}}}{\delta}\tilde{\boldsymbol{S}}_{\nu_{\text{new}}-(\mu_{\text{new}}-\mu_{\text{old}})}.\quad(7)$$

Further, if we alter the FIA accordingly, set $\mu_{\text{old}}=0$ in the beginning and always jump to column $\nu_{\text{new}}=\nu+\mu-\mu_{\text{old}}$, row μ_{old}, then we get $\nu_{\text{new}}=\mu+1$ in general.

For the proof, refer to Feng and Tzeng [5].

Theorem 2 shows how to adjust the FIA in order to run it more efficiently on syndrome matrices. In the beginning, set $\Lambda^{\text{old}}=0$ and $\Lambda=1$. Now, if the algorithm stalls in a column with polynomial Λ, then it is not necessary to consider certain subsequent columns and they may be jumped over. In the new column, it is known how many entries can be eliminated and how to update Λ efficiently. Adjusting the FIA in this way is also due to Feng and Tzeng [5]. In Fig. 3, a formal description of this adjusted FIA is depicted. It includes, however, the generation of the error evaluator polynomial which will first be discussed in the next section. Note that only three "generations" of variables are needed.

From now on, when discussing the BMA we mean more specifically the adjusted FIA. The minor difference to the original publications by Berlekamp [1] and Massey [2] consists of the following. Instead of using Λ^{old} and δ_{old} from column ν_{old}, row μ_{old} when updating in column ν_{new}, row μ_{old}, they use the intermediate Λ and δ from column ν, row μ_{old} whenever the discrepancy δ is nonzero. This is, however, insignificant for the discussions in this correspondence as will be seen in Section IV-B. Either way, Example 1 will be run exactly the same way.

D. Generating Ω and/or Ψ Simultaneously with Λ

So far we have not mentioned how to generate the error evaluator polynomial Ω or coevaluator Ψ. Of course, this can be done by simply computing $\Lambda\cdot S$, then by the key equation (1), they are immediately given. But it can also be done in an iterative fashion. At every stage of the adjusted FIA run on the syndrome matrix $\boldsymbol{S}$ generating Λ, we have an equation of the same kind as in (1). Suppose it has reached column ν and row μ with discrepancy δ in this place, then

$$\Lambda\cdot S=\Omega-\Psi x^{\nu+\mu-2}\quad\text{or}\quad\Lambda\cdot S\equiv\Omega\bmod x^{\nu+\mu-2}\qquad(8)$$

where Ω is a polynomial of degree $<\nu-1$ and Ψ one of degree $\leq 2t-\mu$. In matrix form that is

$$\begin{pmatrix}0&\cdots&0&S_0\\ \vdots&&&\vdots\\ 0&&\cdots&S_{\nu-2}\\ S_0&&&S_{\nu-1}\\ \vdots&&&\vdots\\ S_{\mu-2}&\cdots&&S_{\mu+\nu-3}\\ S_{\mu-1}&\cdots&&S_{\mu+\nu-2}\\ &&&\vdots\\ \vdots&&&S_{2t-1}\\ &&&0\\ &&\cdots&\vdots\\ S_{2t-1}&0&\cdots&0\end{pmatrix}\begin{pmatrix}\Lambda_{\nu-1}\\ \vdots\\ \Lambda_1\\ 1\end{pmatrix}=\begin{pmatrix}\Omega_0\\ \vdots\\ \Omega_{\nu-2}\\ 0\\ \vdots\\ 0\\ -\Psi_0\\ \\ \\ \vdots\\ \\ \\ -\Psi_{2t-\mu-1}\end{pmatrix}\qquad(8')$$

Input: syndrome matrix $\mathbf{S}$
Output: error locator and evaluator polynomial Λ and Ω
Set $\nu = 2$, $\mu = 0$, $\Lambda = 0$, $\Omega = -1$ and $\delta = 1$.
Start with column index $\nu_{\text{new}} = 1$ and row index $\mu_{\text{new}} = 1$.
Initialize $\Lambda^{\text{new}} = 1$ and $\Omega^{\text{new}} = 0$.
repeat
Calculate discrepancy

$$\delta_{\text{new}} = \sum_{j=0}^{\nu_{\text{new}}-1} \Lambda_j S_{\nu_{\text{new}}+\mu_{\text{new}}-j-2}.$$

if $\delta_{\text{new}} \neq 0$ **then**
if $\mu_{\text{new}} \leq \mu$ **then**
Update Λ^{new} and Ω^{new} respectively by

$$\Lambda^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}\Lambda x^{\mu_{\text{new}}-\nu+1},$$
$$\Omega^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}\Omega x^{\mu_{\text{new}}-\nu+1}.$$

else $\{\mu_{\text{new}} > \mu\}$
Set

$$\delta_{\text{old}} = \delta, \qquad \delta = \delta_{\text{new}},$$
$$\mu_{\text{old}} = \mu, \qquad \mu = \mu_{\text{new}},$$
$$\nu_{\text{old}} = \nu, \qquad \nu = \nu_{\text{new}},$$
$$\Lambda_{\text{old}} = \Lambda, \qquad \Lambda = \Lambda_{\text{new}},$$
$$\Omega_{\text{old}} = \Omega, \qquad \Omega = \Omega_{\text{new}}.$$

Set $\mu_{\text{new}} = \mu_{\text{old}}$, $\nu_{\text{new}} = \mu + 1$.
Set Λ^{new} and Ω^{new} to

$$\Lambda - \frac{\delta}{\delta_{\text{old}}}\Lambda^{\text{old}} x^{\nu_{\text{new}}-\nu_{\text{old}}},$$
$$\Omega - \frac{\delta}{\delta_{\text{old}}}\Omega^{\text{old}} x^{\nu_{\text{new}}-\nu_{\text{old}}}.$$

end if
end if
Increment μ_{new} by 1.
until $\mu_{\text{new}} > t$
return Λ^{new} and Ω^{new}

Fig. 3. The BMA in matrix form.

similarly to (3). In column $\epsilon + 1$ of $\boldsymbol{S}$ when the discrepancy in the tth row has been eliminated the actual error locator is obtained and (8) becomes

$$\Lambda \cdot S = \Omega - \Psi x^{t+\epsilon} \quad \text{or} \quad \Lambda \cdot S \equiv \Omega \bmod x^{t+\epsilon}$$

for $\nu = \epsilon + 1$ and $\mu = t$ with $\deg \Omega < \epsilon$ which, by Theorem 1, is equivalent to the key equation (1). This illustrates that Ω or Ψ can be generated simultaneously to Λ in the adjusted FIA. When Λ is the current preliminary version of the error locator in column ν row μ, current Ω and Ψ polynomials can be generated by building the corresponding linear combinations of entries in the first $\nu - 1$ rows and the last $2t - \mu$ rows as in (8′). However, it can be done more efficiently from the knowledge of previous polynomials Ω^{old} and Ψ^{old} similarly to the computation of Λ. The following theorem describes how Ω and Ψ can be initialized and updated iteratively in the setting of the BMA.

Theorem 3: Set $\Omega = -1$, $\Omega^{\text{new}} = 0$, and $\Psi = -1$, $\Psi^{\text{new}} = -S$ and perform the adjusted FIA on syndrome matrix $\boldsymbol{S}$. Equation (8) holds at every stage for the polynomials of one "generation" if the algorithm is supplemented in the following way:

1) when the algorithm jumps from column ν to ν_{new}, polynomials Ω^{new} and Ψ^{new}, respectively, are set to
$$\Omega - \frac{\delta}{\delta_{\text{old}}}\Omega^{\text{old}} x^{\nu_{\text{new}}-\nu_{\text{old}}} \qquad \left(\Psi - \frac{\delta}{\delta_{\text{old}}}\Psi^{\text{old}}\right) \cdot x^{-1}; \qquad (6')$$
2) when a discrepancy $\delta_{\text{new}} \neq 0$ is found in row μ_{new} where $\mu_{\text{old}} < \mu_{\text{new}} \leq \mu$, Ω^{new} and Ψ^{new}, respectively, are updated to
$$\Omega^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}\Omega x^{\mu_{\text{new}}-\nu+1} \qquad \left(\Psi^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}\Psi\right) \cdot x^{-1} \qquad (7')$$
and, additionally;
3) whenever a discrepancy $\delta = 0$ has been found the current Ψ is divided by x.

For the proof, refer to the Appendix.

Fig. 3 gives the formal description of the adjusted version of the FIA which includes the generation of Ω as it is conventionally the error evaluator that is used in the Forney algorithm (2). Note that this is also reasonable in this setting as more memory space is needed to store versions of Ψ than those of Ω. Correctness of the algorithm is given by Theorem 2 and 3.

To illustrate the course of this version of the FIA, a small example will be given.

Example 1: Consider the $(15, 1, 15)$ Reed–Solomon code over $\mathbb{F}_{16}$ with a primitive element α satisfying $\alpha^4 = \alpha + 1$. The generator polynomial is $g = (x - \alpha) \ldots (x - \alpha^{14})$ and we assume that $c = 0$ was sent, but $r = x + x^2 + x^6 + x^7 + x^8 + x^{13}$ received. The different values of the variables Λ, Ω, ν, μ, and $\delta \neq 0$ when the BMA is run on $\boldsymbol{S}$ as well as the way the algorithm traverses the matrix $\boldsymbol{S}$ are shown in Fig. 4. As a result, we obtain the error locator polynomial

$$\Lambda = \alpha^7 x^6 + \alpha^9 x^5 + \alpha^{14} x^4 + \alpha^{12} x^3 + \alpha^{14} x + 1$$
$$= (1 - \alpha^1 x)(1 - \alpha^2 x)(1 - \alpha^6 x)(1 - \alpha^7 x)(1 - \alpha^8 x)(1 - \alpha^{13} x)$$

which indicates the right error locations. With the help of the error evaluator polynomial $\Omega = \alpha^9 x^4 + \alpha^{12} x^2 + \alpha^{14}$ and the Forney algorithm (2) error values can be determined.

Fig. 4 illustrates the course of this FIA on the syndrome matrix of Example 1.

E. Complexity of the BMA

We investigate the time complexity of the BMA when $\epsilon \leq t$ errors have occurred. For simplicity, only multiplications and divisions will be considered. Our worst case assumption, depicted in Fig. 5, is that $S_0 \neq 0$ and that the BMA can in one column only eliminate the two nonzero discrepancies in rows μ_{old} and $\mu = \mu_{\text{old}} + 1$ and stalls again in $\mu_{\text{new}} = \mu + 1$. Updating in column ν_{new}, therefore, involves

1) finding one discrepancy in column ν row μ and one in column $\nu_{\text{new}} = \nu + 1$ row μ. This results in maximally $\nu + \nu_{\text{new}} - 2 = 2\nu - 1$ multiplications, except in column 2 (where $\Lambda = 1$);
2) updating Λ^{new} twice resulting in two divisions, zero multiplications in column 2, and maximally $\nu_{\text{old}} + \nu - 2 = 2\nu - 3$ multiplications subsequently; and
3) updating Ω resulting in zero multiplications in column 2 and maximally $\nu_{\text{old}} + \nu - 2 = 2\nu - 3$ multiplications subsequently.

After column $\epsilon + 1$ row ϵ has been reached and the discrepancy there eliminated, all subsequent discrepancies are 0, but need calculating. Therefore, $(t - \epsilon)\epsilon$ further multiplications are carried out in the end. Adding up over all the columns gives a maximum of

$$(t - \epsilon)\epsilon + 1 + \sum_{\nu=1}^{\epsilon} 6\nu - 7 = (t - \epsilon)\epsilon + 3\epsilon(\epsilon + 1) - 7\epsilon$$
$$= t\epsilon + 2\epsilon^2 - 4\epsilon + 1$$

Λ	Ω	ν	μ	δ
0	-1	2	0	1
1	0	1	1	α^{14}
1	α^{14}	2	1	α^{13}
$\alpha^{14}x+1$	α^{14}	2	2	α^{12}
$\alpha^{13}x^2+\alpha^{14}x+1$	α^{14}	3	5	α^{10}
$\alpha^{12}x^5+\alpha^{13}x^4+\alpha^{13}x^2+\alpha^{14}x+1$	$\alpha^{12}x^4+\alpha^{14}$	6	4	α^{8}
$\alpha^{12}x^5+\alpha^{4}x^4+\alpha^{12}x^3+\alpha^{14}x+1$	$\alpha^{12}x^4+\alpha^{12}x^2+\alpha^{14}$	6	6	α^{4}
$\alpha^{7}x^6+\alpha^{9}x^5+\alpha^{14}x^4+\alpha^{12}x^3+\alpha^{14}x+1$	$\alpha^{9}x^4+\alpha^{12}x^2+\alpha^{14}$	7	8	

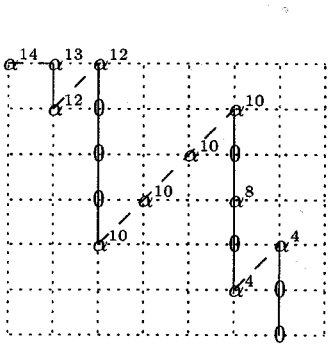

Fig. 4. Example 1: Variable values during the BMA and illustration of the course with discrepancies.

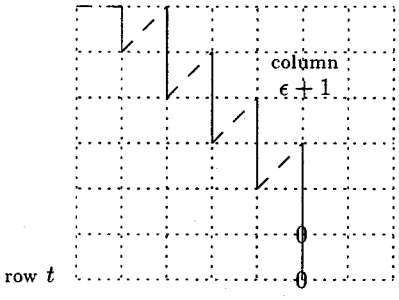

Fig. 5. Illustration of the worst case situation of the course of the BMA on the syndrome matrix—all discrepancies are $\neq 0$ except those marked 0.

multiplications and 2ϵ divisions. This is the best upper bound on the number of multiplications known to the authors.

IV. The EA for Decoding

A. The Original Algorithm

We now compare the BMA from the previous section to the EA. The ancient extended EA performed on two elements a and b of a Euclidean ring (such as, for example, any polynomial ring in one variable) calculates repeatedly elements f, g, and h of the same ring such that

$$h = fa + gb$$

until $h = \gcd(a, b)$. The key equation (1) reminds us of the above as we can write

$$\Omega = \Lambda \cdot S + \Psi \cdot x^{2t}$$

but Ω is, in general, not the greatest common divisor of x^{2t} and S. However, Sugiyama *et al.* [3] proved that by applying the extended EA to polynomials x^{2t} and S, terminating and normalizing when certain degree conditions are satisfied, then error locator, evaluator, and coevaluator polynomials Λ, Ω, and Ψ are obtained. The resulting EA is shown in Fig. 6(a). Terminating as soon as $\deg \rho_{\text{new}} < t$ has the consequence that $\deg a_{22} \leq t$ by comparison of degrees. Note that after every iteration we have the equality

$$\begin{pmatrix} \rho \\ \rho_{\text{new}} \end{pmatrix} = A \begin{pmatrix} x^{2t} \\ S \end{pmatrix}. \tag{9}$$

Thereby we obtain the equation

$$\rho_{\text{new}} = a_{21}x^{2t} + a_{22}S$$

and dividing by the constant $a_{22}(0)$ gives the error locator polynomial as $\Lambda = a_{22}/a_{22}(0)$, the error evaluator $\Omega = \rho_{\text{new}}/a_{22}(0)$, and the error coevaluator $\Psi = a_{21}/a_{22}(0)$.

B. A First Adjustment

At first sight, the EA seems to be different from the BMA, since intermediate results do not coincide. Two immediately obvious reasons why they cannot coincide in general, are

1) whereas the BMA starts its arithmetic computations with $S_1, S_2, \ldots$ the EA starts with $S_{2t-1}, S_{2t-2}, \ldots$; and
2) the preliminary version of the error locator polynomial a_{22} is not kept to have constant term 1 at all times.

We denote the reciprocal of a polynomial $f \in \mathbb{F}[x]$ by $f^{\text{rec}} = x^{\deg f} f(x^{-1})$. Further, we define the *reversed syndrome polynomial*

$$\bar{S} = x^{2t-1}S(x^{-1}) = S_0x^{2t-1} + S_1x^{2t-2} + \cdots + S_{2t-1}.$$

Note that $\bar{S}$ is generally not equal to S^{rec}. We investigate running the EA on $\bar{S}$ instead. When doing so it cannot be expected that the error locator polynomial turns up as a normalized version of a_{22}, but rather as the reciprocal of it. Further, instead of normalizing at the end, one can keep a_{22} monic (since we think it will be the reciprocal of Λ) throughout the course of the EA. The algorithm we propose is shown in Fig. 6(b). Note in particular the equality

$$\begin{pmatrix} 0 & 1 \\ Q & -q \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -q_0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & 0 \\ -q_{d-1}x^{d-1} & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ Q & x^d \end{pmatrix}$$

and the similarity to the algorithm in Fig. 6(a). Further, recall polynomial division. Whenever a new term of q is obtained an intermediate ρ_{new} is calculated that has degree $\geq \deg \rho$ until the division has reached its end. By these considerations, (9) holds not only after each polynomial division where $\deg \rho_{\text{new}} < \deg \rho$, but also when one polynomial division is not yet complete and an intermediate ρ_{new} has been calculated with $\deg \rho_{\text{new}} \geq \deg \rho$.

The coefficient of the term of largest degree in a general polynomial $f \in \mathbb{F}[x]$ is called the *leading coefficient* of f, abbreviated $\operatorname{lc} f$. The reason why we choose $\operatorname{lc} q = -1$ is that $-q$ and a_{22} are then always monic.

Correctness of the adjusted EA is given by the following theorem.

Theorem 4: Suppose the BMA from Fig. 3, but including the generations of Ψ as in (6′), (7′), and the first version of the adjusted EA from Fig. 6(b) are run on the nontrivial syndromes of the same error

Input: syndrome polynomial $S \neq 0$, integer t
Output: error locator, evaluator and coevaluator polynomials Λ, Ω, Ψ
Initialize $\rho_{\text{old}} = x^{2t}$, $\rho = S$ and

$$A = (a_{ij})_{1 \le i,j \le 2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

repeat
Compute quotient q of ρ_{old} by ρ and remainder ρ_{new} such that

$$\rho_{\text{old}} = q\rho + \rho_{\text{new}}, \quad \deg \rho_{\text{new}} < \deg \rho.$$

Update A by

$$\begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} A.$$

Set $\rho_{\text{old}} = \rho$, $\rho = \rho_{\text{new}}$.
until $\deg \rho < t$
return $a_{22}/a_{22}(0)$, $\rho/a_{22}(0)$, $a_{21}/a_{22}(0)$

(a)

Input: reversed syndrome polynomial $\bar{S} \neq 0$, integer t
Output: error locator, evaluator and coevaluator polynomials Λ, Ω, Ψ
Initialize $\rho_{\text{old}} = x^{2t}$, $\rho = \bar{S}$ and

$$A = (a_{ij})_{1 \le i,j \le 2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

repeat
Compute constant Q, quotient $q = -x^d + q_{d-1}x^{d-1} + \cdots + q_0$ of ρ_{old} by ρ and remainder ρ_{new} such that

$$Q\rho_{\text{old}} = q\rho + \rho_{\text{new}}, \quad \deg \rho_{\text{new}} < \deg \rho.$$

Update A simultaneously to obtaining terms of q by

$$\begin{pmatrix} 1 & 0 \\ -q_0 & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & 0 \\ -q_{d-1}x^{d-1} & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ Q & x^d \end{pmatrix} A.$$

Set $\rho_{\text{old}} = \rho$, $\rho = \rho_{\text{new}}$.
until $\deg \rho < t$
return a_{22}^{rec}, $-x^{\deg a_{22}-1} a_{21}(x^{-1})$, $-\rho^{\text{rec}}$

(b)

Fig. 6. (a) The original EA. (b) The adjusted EA—version 1.

pattern. After every update of Λ^{new}, Ω^{new}, a_{22}^{new}, and a_{21}^{new} and setting μ_{new} to the index of the next row with discrepancy $\delta_{\text{new}} \neq 0$, we have

$$\text{for } a_{22}^{\text{new}} \neq 0$$
$$a_{22}^{\text{new rec}} = \Lambda^{\text{new}}, \quad \deg a_{22}^{\text{new}} = \nu_{\text{new}} - 1 \tag{10}$$

$$\text{for } a_{22}^{\text{new}} \neq 0$$
$$-x^{\deg a_{22}^{\text{new}} - 1} a_{21}^{\text{new}}(x^{-1}) = \Omega^{\text{new}}, \quad \deg a_{21}^{\text{new}} < \nu_{\text{new}} - 1 \tag{11}$$

$$-\rho_{\text{new}}^{\text{rec}} = \Psi^{\text{new}}, \quad \deg \rho_{\text{new}} = 2t - \mu_{\text{new}}. \tag{12}$$

In particular, $\operatorname{lc} \rho_{\text{new}} = \delta_{\text{new}}$. Furthermore, there is a one-to-one correspondence between column shifts in the BMA and the start of a new polynomial division in the adjusted EA as well as a one-to-one correspondence between every update of Λ and Ω within one column in the BMA and the number of nontrivial updates of a_{21} and a_{22} within one polynomial division in the adjusted EA.

Refer to the Appendix for the proof.

Example 2: Let us reconsider the code and the received word from Example 1 and run the adjusted EA on the corresponding $\bar{S}$ of degree 13. The different values of the variables a_{22}, a_{21}, ρ, Q, and the terms $q_i x^i$ of the quotient polynomial q are given in the table in Fig. 7. As expected by Theorem 4 the four column shifts illustrated in the diagram in Fig. 4 in the BMA, correspond to four polynomial divisions in the adjusted EA. Furthermore, the additional updates in columns 2 and 6 correspond to the calculation of a further term of q in the first and the third polynomial division. And comparison with the coefficients of the polynomials in the table of Fig. 4 shows that at every point where $a_{22}^{\text{new}} \neq 0$, we have the equalities $a_{22}^{\text{rec}} = \Lambda$, $-x^{\deg a_{22}-1} a_{21}(x^{-1}) = \Omega$, and $\operatorname{lc} \rho = \delta$.

In order to understand intuitively what is going on in the two algorithms compare with Fig. 8. Suppose the BMA finds discrepancy $\delta \neq 0$ in column ν, row μ. By (8) we get that

$$\Lambda S = \Omega - \Psi x^{\nu+\mu-2} \equiv \Omega \bmod x^{\nu+\mu-2}.$$

Its coefficients are the entries in the obvious order of the product

$$\begin{pmatrix} 0 & \cdots & 0 & S_0 \\ \vdots & \iddots & \iddots & \\ 0 & & & \vdots \\ S_0 & & & \\ & & & S_{2t-1} \\ \vdots & & \iddots & 0 \\ & & \iddots & \vdots \\ S_{2t-1} & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \\ 1 \end{pmatrix}$$

where the BMA has established that entries $\nu, \ldots, \nu + \mu - 2$ are 0. In other words, coefficients of Ω occupy entries $1, \ldots, \nu - 1$, entry $\nu + \mu - 1$ is a nonzero discrepancy δ, and coefficients of $-\Psi$ occupy all entries $\nu + \mu - 1, \ldots, 2t + \nu - 1$.

Having carried out the EA to this point, the product $a_{22}\bar{S} = \rho - a_{21}x^{2t}$ has the same coefficients, only in reversed order by Theorem 4. The polynomial $-a_{21}$ corresponds to Ω and, therefore, $\deg a_{21} < \nu - 1$ and, similarly, $-\rho$ corresponds to Ψ, thus $\operatorname{lc} \rho = \delta$ and $\deg \rho = 2t - \mu$.

If we also want to compensate for the minor difference between the FIA and the original BMA in the EA, then when dividing ρ_{old} by ρ and coming across a version of ρ_{new} that has the same degree as ρ, the polynomials ρ with corresponding a_{22} and a_{21} should be replaced by ρ_{new} and corresponding versions of a_{22}^{new} and a_{21}^{new} for the next polynomial division. By Theorem 4, this corresponds exactly to using versions Λ, Ω, Ψ, and $\delta \neq 0$ from column ν, row μ_{old} for updating in column ν_{new}, row μ_{old} in the course of the FIA.

C. Complexity of the Adjusted EA

With the first adjustment we have obtained the desired result in so far as the BMA and the EA suddenly without having changed much, resemble each other quite a bit. Updating Λ and Ω in the one corresponds exactly to updating the matrix A in the other. However, doing the polynomial division ρ_{old} by ρ seems to be more involved than simply computing discrepancies. It corresponds to computing Ψ additionally in the BMA which is not necessary.

We investigate the time complexity of the adjusted EA when $\epsilon \le t$ errors have occurred. For simplicity, only multiplications and divisions will be considered. Our worst case assumption is that $S_0 \neq 0$ as well

a_{22}	a_{21}	ρ	Q	$q_i x^i$
0	1	x^{14}		
1	0	$\bar{S} = \alpha^{14}x^{13} + \cdots$	α^{14}	x
x	α^{14}	$\alpha^{13}x^{13} + \cdots$		α^{14}
$x + \alpha^{14}$	α^{14}	$\alpha^{12}x^{12} + \cdots$	α^{13}	x
$x^2 + \alpha^{14}x + \alpha^{13}$	$\alpha^{14}x$	$\alpha^{10}x^9 + \cdots$	α^{13}	x^3
$x^5 + \alpha^{14}x^4 + \alpha^{13}x^3 + \alpha^{13}x + \alpha^{12}$	$\alpha^{14}x^4 + \alpha^{12}$	$\alpha^8 x^{10} + \cdots$		$\alpha^{13}x$
$x^5 + \alpha^{14}x^4 + \alpha^{12}x^2 + \alpha^4 x + \alpha^{12}$	$\alpha^{14}x^4 + \alpha^{12}x^2 + \alpha^{12}$	$\alpha^4 x^8 + \cdots$	α^9	x
$x^6 + \alpha^{14}x^5 + \alpha^{12}x^3 + \alpha^{14}x^2 + \alpha^9 x + \alpha^7$	$\alpha^{14}x^5 + \alpha^{12}x^3 + \alpha^9 x$	$\alpha^{14}x^4 + \cdots$		

Fig. 7. Example 2: Variable values during the EA.

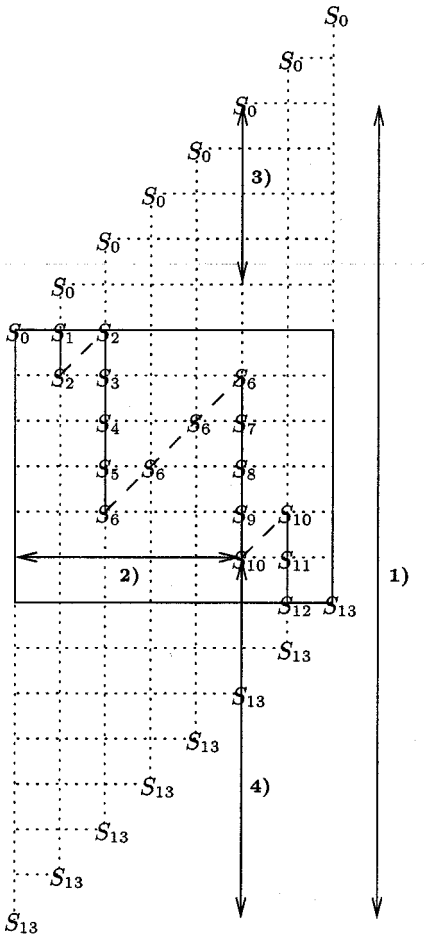

Fig. 8. Illustration of Theorem 4 through Example 2. A matrix similar to the one in (3) is depicted. The solid box sets the syndrome matrix apart from the rest. When the BMA stalls in column 6, row 6 of $\boldsymbol{S}$, the number of coefficients in ΛS, and $a_{22}\bar{S}$ corresponds to the length of the arrow marked $\mathbf{1}$), the number of coefficients of Λ and a_{22} corresponds to the length of the arrow marked $\mathbf{2}$), the number of coefficients of Ω and a_{21} corresponds to the length of the one marked $\mathbf{3}$) and the number of coefficients of ρ is given by the length of the arrow marked $\mathbf{4}$). Note that some initial coefficients might be trivial and, therefore, we do not discus the degree of the polynomials in question.

as that in each division $\deg q = 1$ and q's constant term is not trivial. In other words, the degrees of the remainder polynomials only fall by 1 in each step. This corresponds exactly to the worst case assumption described in Section III-E and Fig. 5. One polynomial division, therefore, involves

1) maximally two divisions and $\deg \rho_{\text{old}} + \deg \rho$ multiplications to calculate quotient q;
2) updating a_{22}^{new} twice resulting in no multiplications during the first division and maximally $\deg a_{22}^{\text{old}} + \deg a_{22}$ multiplications subsequently; and
3) updating a_{21}^{new} resulting in no multiplications during the first division and maximally $\deg a_{21}^{\text{old}} + \deg a_{21} + 2 \leq \deg a_{22}^{\text{old}} + \deg a_{22}$ multiplications subsequently.

After ϵ polynomial divisions the final result will be obtained and adding up over all of them gives a maximum of

$$\begin{aligned} &2 + \sum_{\nu=1}^{\epsilon} \deg \rho_{\text{old}} + \deg \rho + 2 \deg a_{22}^{\text{old}} + 2 \deg a_{22} \\ &\quad = 2 + \sum_{i=1}^{\epsilon} 2t - (i-1) + 2t - i + 2(i-2) + 2(i-1) \\ &\quad = 4t\epsilon - 5\epsilon + \epsilon(\epsilon+1) + 2 = 4t\epsilon + \epsilon^2 - 4\epsilon + 2 \end{aligned}$$

multiplications and 2ϵ divisions which is strictly larger than the complexity of the BMA. Note that the complexity of the original EA in the worst case will be the same as the one of the adjusted version and that the one of the BMA would also be this large if one included the error coevaluator Ψ in the calculations as in (6′), (7′).

D. A Second and Final Adjustment

We have mentioned that at first sight it seems impossible to get around the computation of ρ in the EA. It corresponds exactly to the computation of Ψ in the BMA which one certainly can avoid. The observation of this correspondence enables us to see how the first version of the adjusted EA can be made more efficient and resemble the BMA exactly: the remainder polynomials ρ of which several versions are calculated during one polynomial division, are mostly superfluous. Only their leading coefficients are needed to compute coefficients of the quotients q which are essential for updating the matrix A. We know that after every update we have the equality (9) and thereby

$$a_{22}^{\text{new}}\bar{S} = \rho_{\text{new}} - a_{21}^{\text{new}} x^{2t}$$

where the right-hand side sorts terms of the product $a_{22}^{\text{new}}\bar{S}$ into those of degree $<2t$ and of degree $\geq 2t$. This shows that to obtain $\operatorname{lc}\rho_{\text{new}}$ only the appropriate coefficient of $a_{22}^{\text{new}}\bar{S}$ needs computing. For example, suppose the EA was run without calculating remainder polynomials ρ explicitly, only leading coefficients and degrees of previous versions of ρ are known at a given stage. When a new polynomial division is started, then by comparison with the proof

Input: reversed syndrome polynomial $\bar{S} \neq 0$, integer t
Output: error locator and evaluator polynomial Λ and Ω
Initialize $\deg \rho_{\text{old}} = 2t$, $\operatorname{lc} \rho_{\text{old}} = 1$, $\deg \rho = \deg \bar{S}$, $\operatorname{lc} \rho = \operatorname{lc} \bar{S}$ and

$$A = (a_{ij})_{1 \le i,j \le 2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

repeat
Update A by

$$\begin{pmatrix} 0 & 1 \\ -\frac{\operatorname{lc} \rho}{\operatorname{lc} \rho_{\text{old}}} & x^{\deg \rho_{\text{old}} - \deg \rho} \end{pmatrix} A.$$

Compute coefficients $x^{\deg \rho_{\text{old}}-1}, x^{\deg \rho_{\text{old}}-2}, \ldots$ in $a_{22}^{\text{new}} \bar{S}$ until a non-zero one is found and $\deg \rho_{\text{new}}$, $\operatorname{lc} \rho_{\text{new}}$ obtained or until $\deg \rho_{\text{new}} < t$ has been established.
while $\deg \rho_{\text{new}} \ge \deg \rho$ **do**
Update A by

$$\begin{pmatrix} 1 & 0 \\ -\frac{\operatorname{lc} \rho_{\text{new}}}{\operatorname{lc} \rho} x^{\deg \rho_{\text{new}} - \deg \rho} & 1 \end{pmatrix} A.$$

Compute coefficients $x^{\deg \rho_{\text{new}}-1}, x^{\deg \rho_{\text{new}}-1}, \ldots$ in $a_{22}^{\text{new}} \bar{S}$ until a non-zero one is found and $\deg \rho_{\text{new}}$, $\operatorname{lc} \rho_{\text{new}}$ obtained or until $\deg \rho_{\text{new}} < t$ has been established.
end while
until $\deg \rho_{\text{new}} < t$
return a_{22}^{rec} and $-x^{\deg a_{22}-1} a_{21}(x^{-1})$

Fig. 9. The adjusted EA—Version 2.

of Theorem 4, Q and d in Fig. 6(b) can be set to $-\operatorname{lc} \rho / \operatorname{lc} \rho_{\text{old}}$ and $\deg \rho_{\text{old}} - \deg \rho$. Also, $\deg \rho_{\text{old}} > \deg \rho_{\text{new}}$ and, therefore, the coefficients of $x^{\deg \rho_{\text{old}}-1}, x^{\deg \rho_{\text{old}}-2}, \ldots$ in the product $a_{22}^{\text{new}} \bar{S}$ can be calculated until a nonzero one is found. This must then be $\operatorname{lc} \rho_{\text{new}}$, and $\deg \rho_{\text{new}}$ is obtained simultaneously. Now, if also $\deg \rho > \deg \rho_{\text{new}}$, then a new polynomial division has to be started. Otherwise, the current one continues and the same can be repeated such that coefficients of q from Fig. 6(b) are of the form $\operatorname{lc} \rho_{\text{new}} / \operatorname{lc} \rho$ and the corresponding exponents are $\deg \rho_{\text{new}} - \deg \rho$. Note that the computation of those coefficients of $a_{22}^{\text{new}} \bar{S}$ corresponds exactly to the computation of discrepancies—compare with Theorem 4. Fig. 9 gives the formal description of this procedure. By the discussions in Sections III-E, IV-C, and Theorem 4 the complexity of the BMA and the one of this second adjusted version of the EA are equal. The superfluous computation of the entire remainder polynomials has been replaced by discrepancy calculation and the two algorithms do essentially the same computations.

V. Discussion

In this correspondence, we have chosen to alter the EA to perform the same operations as the BMA. It should be clear from the illustration that instead one could similarly alter the BMA to run on a kind of reversed syndrome matrix and to produce error coevaluator Ψ in addition to error locator Λ and error evaluator Ω. It would then perform the same operations as the original EA.

Note that it is not necessary to run the adjusted EA on the reversed syndrome polynomial $\bar{S}$. Giving S as input will result in obtaining the error coevaluator Ψ from

$$\Lambda \cdot S = \Omega - \Psi x^{2t}$$

which can be used just as well in the Forney algorithm (2) to obtain error values.

It is worth mentioning that calculating with the reversed syndrome polynomial and the reciprocal of Λ turns the key equation into the following:

$$\begin{aligned} \prod_{i=1}^{\epsilon} (x - \alpha^{l_j}) \cdot \bar{S} &= \Lambda^{\text{rec}} \cdot \bar{S} \\ &= x^{\epsilon} \Lambda(x^{-1}) \cdot x^{2t-1} S(x^{-1}) \\ &= x^{2t+\epsilon-1} (\Lambda \cdot S)(x^{-1}) \\ &= x^{2t+\epsilon-1} (\Omega(x^{-1}) - \Psi(x^{-1}) x^{-2t}) \\ &= x^{2t} \sum_{j=1}^{\epsilon} e_{l_j} \alpha^{l_j} \prod_{\substack{i=1 \\ i \neq j}}^{\epsilon} (x - \alpha^{l_i}) \\ &\quad - \sum_{j=1}^{\epsilon} e_{l_j} \alpha^{(2t+1) l_j} \prod_{\substack{i=1 \\ i \neq j}}^{\epsilon} (x - \alpha^{l_i}). \end{aligned} \tag{13}$$

The above suggest an alternate definition of all the involved polynomials

$$\begin{aligned} \Lambda^* &= \prod_{i=1}^{\epsilon} (x - \alpha^{l_j}) \\ \Omega^* &= \sum_{j=1}^{\epsilon} e_{l_j} \alpha^{l_j} \prod_{\substack{i=1 \\ i \neq j}}^{\epsilon} (x - \alpha^{l_i}) \\ \Psi^* &= \sum_{j=1}^{\epsilon} e_{l_j} \alpha^{(2t+1) l_j} \prod_{\substack{i=1 \\ i \neq j}}^{\epsilon} (x - \alpha^{l_i}) \end{aligned}$$

avoiding the awkwardness of α^{-l_i} indicating error locations rather than α^{l_i}. This alternate definition has already been put forth by O'Sullivan [9] in a slightly different version. He turns the syndrome polynomial into a power series in $1/x$ and can then omit the coevaluator polynomial from the key equation. But dividing (13) as well as $\bar{S}$ by x^{2t} resulting in $S^* = 1/x \cdot S(x^{-1})$ and running the adjusted EA on polynomials 1 and S^* (instead of x^{2t} and $\bar{S}$) will find the Λ^* and the new Ω^*. O'Sullivan's perspective on the key equation which he generalized in [9] might be the better one, and the long confusion whether BMA and EA are the same algorithms or not might be due to the wrong perspective.

VI. Conclusion

The main contribution of this correspondence is the clear illustration of all the parallels that can be found between the BMA and the EA. In order to do this, the FIA needed to be extended to calculate the error evaluator and coevaluator as in Fig. 3, (6′) and (7′). The coevaluator polynomial does not seem to come in naturally for neither [1], [2], nor [4], but because of its inclusion, we can easily go in both directions of how the EA can be adapted to the BMA as well as the other way.

Finally, it can be concluded that the Berlekamp-Massey and the Euclidean algorithms are essentially the same, since they can both be adapted in a simple way to perform the same operations as the other.

Appendix

Proof of Theorem 3

We prove by induction.

Note that in the beginning before anything has been eliminated

$$\begin{aligned} \Lambda \cdot S = 0 &= -1 + 1 \cdot x^0 \\ &= \Omega - \Psi x^{\nu+\mu-2} \quad \text{and} \\ \Lambda^{\text{new}} \cdot S = S &= 0 + S \cdot x^0 \\ &= \Omega^{\text{new}} - \Psi^{\text{new}} x^{\nu_{\text{new}} + \mu_{\text{new}} - 2} \end{aligned}$$

where

$$\deg \Omega = 0 < 1 = \nu - 1$$
$$\deg \Omega^{\text{new}} < 0 = \nu_{\text{new}} - 1$$

and

$$\deg \Psi = 0 \le 2t = 2t - \mu$$
$$\deg \Psi^{\text{new}} \le 2t = 2t - \mu_{\text{new}}.$$

Suppose that the algorithm reaches column ν and row μ of the syndrome matrix $\boldsymbol{S}$ with discrepancy $\delta \neq 0$ such that $\mu > \mu_{\text{old}}$. In other words, it jumps to column $\nu_{\text{new}} = \mu + 1$, row $\mu_{\text{new}} = \mu_{\text{old}}$ where δ is eliminated. With μ_{new} set to $\mu_{\text{old}} + 1$ where the next discrepancy is we get

$$\begin{aligned}\Lambda^{\text{new}} \cdot S &= \left(\Lambda - \frac{\delta}{\delta_{\text{old}}}\Lambda^{\text{old}} x^{\nu_{\text{new}}-\nu_{\text{old}}}\right)\cdot S \\ &= \Lambda\cdot S - \frac{\delta}{\delta_{\text{old}}}\Lambda^{\text{old}}\cdot S x^{\nu_{\text{new}}-\nu_{\text{old}}} \\ &= \Omega - \Psi x^{\nu+\mu-2} \\ &\quad - \frac{\delta}{\delta_{\text{old}}}(\Omega^{\text{old}} - \Psi^{\text{old}} x^{\nu_{\text{old}}+\mu_{\text{old}}-2})\cdot x^{\nu_{\text{new}}-\nu_{\text{old}}} \\ &= \Omega^{\text{new}} - \left(\Psi x^{-1} - \frac{\delta}{\delta_{\text{old}}}\Psi^{\text{old}} x^{-1}\right)\cdot x^{\nu_{\text{new}}+\mu_{\text{old}}-1} \\ &= \Omega^{\text{new}} - \Psi^{\text{new}} x^{\nu_{\text{new}}+\mu_{\text{old}}-1} \\ &= \Omega^{\text{new}} - \Psi^{\text{new}} x^{\nu_{\text{new}}+\mu_{\text{new}}-2} \qquad (14)\end{aligned}$$

Expression (14) holds by induction.

Suppose now, on the other hand, that the algorithm reaches row μ_{new} in column ν_{new} with discrepancy $\delta_{\text{new}} \neq 0$ such that $\mu_{\text{new}} \le \mu$, then Λ^{new} and Ω^{new} are updated. Going from μ_{new} to $\mu_{\text{new}} + 1$ where the next discrepancy is, we calculate

$$\begin{aligned}&\left(\Lambda^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}\Lambda x^{\mu_{\text{new}}-\nu+1}\right)\cdot S \\ &\quad = \Lambda^{\text{new}}\cdot S - \frac{\delta_{\text{new}}}{\delta}\Lambda\cdot S x^{\mu_{\text{new}}-\nu+1} \\ &\quad = \Omega^{\text{new}} - \Psi^{\text{new}} x^{\nu_{\text{new}}+\mu_{\text{new}}-2} \\ &\qquad - \frac{\delta_{\text{new}}}{\delta}(\Omega - \Psi x^{\nu+\mu-2})\cdot x^{\mu_{\text{new}}-\nu+1} \\ &\quad = \Omega^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}\Omega x^{\mu_{\text{new}}-\nu+1} \\ &\qquad - (\Psi^{\text{new}} x^{-1} - \frac{\delta_{\text{new}}}{\delta}\Psi x^{-1}) x^{\nu_{\text{new}}+(\mu_{\text{new}}+1)-2} \qquad (14')\end{aligned}$$

where again (14′) holds by induction.

Finally, when a discrepancy $\delta = 0$ is found, it means that the current Ψ has no constant term and, therefore, (8) also holds after such division of Ψ by x.

Proof of Theorem 4

Lemma 1: Let $f, g, h \in \mathbb{F}[x]\backslash\{0\}$, $\lambda, \mu \in \mathbb{F}^*$, and $i \in \mathbb{N}_0$ such that

$$f = \lambda g + \mu x^i h.$$

1) If $\deg f = \deg g$, then
$$f^{\text{rec}} = \lambda g^{\text{rec}} + \mu x^{\deg g - (i+\deg h)} h^{\text{rec}}.$$
2) If $\deg f = i + \deg h$, then
$$f^{\text{rec}} = \lambda x^{i+\deg h - \deg g} g^{\text{rec}} + \mu h^{\text{rec}}.$$

Proof:

1)
$$\begin{aligned}f^{\text{rec}} &= x^{\deg f} f(x^{-1}) \\ &= x^{\deg g}(\lambda g(x^{-1}) + \mu x^{-i} h(x^{-1})) \\ &= x^{\deg g}\left(\lambda x^{-\deg g} g^{\text{rec}} + \mu x^{-(i+\deg h)} h^{\text{rec}}\right) \\ &= \lambda g^{\text{rec}} + \mu x^{\deg g - (i+\deg h)} h^{\text{rec}}.\end{aligned}$$

2)
$$\begin{aligned}f^{\text{rec}} &= x^{\deg f} f(x^{-1}) \\ &= x^{i+\deg h}(\lambda g(x^{-1}) + \mu x^{-i} h(x^{-1})) \\ &= x^{i+\deg h}\left(\lambda x^{-\deg g} g^{\text{rec}} + \mu x^{-(i+\deg h)} h^{\text{rec}}\right) \\ &= \lambda x^{i+\deg h - \deg g} g^{\text{rec}} + \mu h^{\text{rec}}.\end{aligned}$$

□

First a remark on notation. Suppose A from the adjusted EA has just been updated, then $a_{11}^{\text{new}} = a_{21}$ and $a_{12}^{\text{new}} = a_{22}$ where a_{21}, a_{22} denote entries from the previous polynomial division. In this way we can restrict our considerations on three "generations" of a_{21} and a_{22}—old, current, and new. Note that $\deg a_{21} < \deg a_{21}^{\text{new}}$ and $\deg a_{22} < \deg a_{22}^{\text{new}}$.

We prove by induction on the steps of the two algorithms.

In the beginning, we have

$$\begin{aligned}&a_{22}^{\text{old rec}} = 0 = \Lambda^{\text{old}} \\ &a_{22}^{\text{rec}} = 1 = \Lambda, && \deg a_{22} = 0 = \nu - 1 \\ &&& \deg a_{21}^{\text{old}} = 0 < \nu_{\text{old}} - 1 \\ &-x^{\deg a_{22}-1} a_{21}(x^{-1}) = 0 = \Omega, && \deg a_{21} < \nu - 1 \\ &-\rho_{\text{old}}^{\text{rec}} = -1 = \Psi^{\text{old}}, && \deg \rho_{\text{old}} = 2t = 2t - \mu_{\text{old}} \\ &-\rho^{\text{rec}} = -S x^{-(\mu-1)} = \Psi, && \deg \rho = 2t - \mu.\end{aligned}$$

In particular, $\operatorname{lc}\rho_{\text{old}} = 1 = \delta_{\text{old}}$ and $\operatorname{lc}\rho = S_{\mu-1} = \delta$. The claim of Theorem 4 holds in the beginning and we may assume in the investigation of a step that the theorem holds for previous steps of the two algorithms.

Suppose the BMA finds a nonzero discrepancy. We distinguish between two cases.

The first case is that this discrepancy $\delta \neq 0$ in column ν, row μ cannot be eliminated and a column change is forced such that $\nu_{\text{new}} = \mu + 1$, $\mu_{\text{new}} = \mu_{\text{old}}$, and

$$\Lambda^{\text{new}} = \Lambda - \frac{\delta}{\delta_{\text{old}}}\Lambda^{\text{old}} x^{\nu_{\text{new}}-\nu_{\text{old}}}.$$

This implies $\mu > \mu_{\text{old}}$ and $\mu \le t$, thus

$$t \le 2t - \mu = \deg \rho < \deg \rho_{\text{old}} = 2t - \mu_{\text{old}}$$

such that the adjusted EA starts a new polynomial division of ρ_{old} by ρ. It sets $d = \deg \rho_{\text{old}} - \deg \rho$ and $Q = -\operatorname{lc}\rho/\operatorname{lc}\rho_{\text{old}}$ such that

$$a_{22}^{\text{new}} = -\frac{\operatorname{lc}\rho}{\operatorname{lc}\rho_{\text{old}}} a_{22}^{\text{old}} + x^{\deg \rho_{\text{old}} - \deg \rho} a_{22}. \qquad (15)$$

Note that $a_{22}^{\text{new}} = x^{\mu}$ during the first polynomial division and, therefore, $a_{22}^{\text{new rec}} = 1 = \Lambda$. Later when $a_{22}^{\text{old}} \neq 0$ we can reciprocate (15) with the help of 2) in Lemma 1

$$a_{22}^{\text{new rec}} = a_{22}^{\text{rec}} - \frac{\operatorname{lc}\rho}{\operatorname{lc}\rho_{\text{old}}} x^{\deg \rho_{\text{old}} - \deg \rho + \deg a_{22} - \deg a_{22}^{\text{old}}} a_{22}^{\text{old rec}}.$$

But

$$a_{22}^{\text{old rec}} = \Lambda^{\text{old}}, \quad a_{22}^{\text{rec}} = \Lambda, \quad \frac{\operatorname{lc}\rho}{\operatorname{lc}\rho_{\text{old}}} = \frac{\delta}{\delta_{\text{old}}}$$

and

$$\begin{aligned}&\deg \rho_{\text{old}} - \deg \rho + \deg a_{22} - \deg a_{22}^{\text{old}} \\ &\quad = 2t - \mu_{\text{old}} - 2t + \mu + \nu - 1 - \nu_{\text{old}} + 1 \\ &\quad = \mu - \mu_{\text{old}} + \nu - \nu_{\text{old}} \\ &\quad = \nu_{\text{new}} - \nu_{\text{old}}\end{aligned}$$

by induction. Therefore, we get

$$a_{22}^{\text{new rec}} = \Lambda - \frac{\delta}{\delta_{\text{old}}}\Lambda^{\text{old}} x^{\nu_{\text{new}}-\nu_{\text{old}}} = \Lambda^{\text{new}}$$

as in (6). As $a_{22}^{\text{new}} = x^{\mu}$ during the first polynomial division we obtain $\deg a_{22}^{\text{new}} = \nu_{\text{new}} - 1$. Subsequently, when $a_{22}^{\text{old}} \neq 0$ we have $\deg a_{22}^{\text{old}} = \nu_{\text{old}} - 1$ and $\deg a_{22} = \nu - 1$ by induction and, therefore,

$$\begin{aligned}\deg a_{22}^{\text{new}} &= \deg a_{22} + \deg \rho_{\text{old}} - \deg \rho \\ &= \nu - 1 + 2t - \mu_{\text{old}} - 2t + \mu \\ &= \nu_{\text{new}} - 1.\end{aligned}$$

The second case is that this discrepancy $\delta_{\text{new}} \neq 0$ in column ν_{new}, row μ_{new} can be eliminated and Λ^{new} is updated to

$$\Lambda^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}\Lambda x^{\mu_{\mathbf{new}}-\nu+1}$$

without column shift. That is, we have $\mu_{\text{old}} < \mu_{\text{new}} \leq \mu$ and, therefore,

$$2t - \mu_{\text{new}} = \deg \rho_{\text{new}} \geq \deg \rho = 2t - \mu$$

such that the adjusted EA continues the polynomial division ρ_{old} by ρ. The next term of the quotient polynomial q that is calculated is, therefore, $\text{lc}\,\rho_{\text{new}}/\text{lc}\,\rho x^{\deg \rho_{\mathbf{new}}-\deg \rho}$ and a_{22}^{new} is updated to

$$-\frac{\text{lc}\,\rho_{\text{new}}}{\text{lc}\,\rho}x^{\deg \rho_{\mathbf{new}}-\deg \rho}a_{22} + a_{22}^{\text{new}}. \tag{15$'$}$$

Reciprocating with 1) in Lemma 1 gives $a_{22}^{\text{new rec}}$ as

$$a_{22}^{\text{new rec}} - \frac{\text{lc}\,\rho_{\text{new}}}{\text{lc}\,\rho}x^{\deg a_{22}^{\mathbf{new}}-\deg \rho_{\mathbf{new}}+\deg \rho-\deg a_{22}}a_{22}^{\text{rec}}.$$

But

$$a_{22}^{\text{rec}} = \Lambda \qquad a_{22}^{\text{new rec}} = \Lambda^{\text{new}} \qquad \frac{\text{lc}\,\rho_{\text{new}}}{\text{lc}\,\rho} = \frac{\delta_{\text{new}}}{\delta}$$

and

$$\begin{aligned}&\deg a_{22}^{\text{new}} - \deg\, \rho_{\text{new}} + \deg \rho - \deg a_{22} \\ &\quad = \nu_{\text{new}} - 1 - 2t + \mu_{\text{new}} + 2t - \mu - \nu + 1 \\ &\quad = \mu_{\text{new}} - \nu + 1\end{aligned}$$

by induction. Therefore, we get

$$a_{22}^{\text{new rec}} = \Lambda^{\text{new}} - \frac{\delta_{\text{new}}}{\delta}x^{\mu_{\mathbf{new}}-\nu+1}\Lambda$$

as in (7). Also $\deg a_{22}^{\text{new}} = \nu_{\text{new}} - 1$ since $\mu_{\text{new}} > \mu_{\text{old}}$ and, thereby, $\nu - 1 + \mu - \mu_{\text{new}} < \nu_{\text{new}} - 1$. Hence, claim (10) is proven.

By the above we have also shown that there is a one-to-one correspondence between column shifts in the BMA and the start of a new polynomial division in the adjusted EA as well as a one-to-one correspondence between updates of Λ and Ω within one column in the BMA and the number of nontrivial updates of a_{21} and a_{22} within one polynomial division in the adjusted EA.

Claims (11) and (12) are left to prove. In both cases, whether a new polynomial division is started or continued, a polynomial

$$\rho_{\text{new}} = a_{21}^{\text{new}}x^{2t} + a_{22}^{\text{new}}\bar{S} \tag{16}$$

is obtained and we rewrite

$$a_{22}^{\text{new}}\bar{S} = \rho_{\text{new}} - a_{21}^{\text{new}}x^{2t} \tag{16$'$}$$

where the right-hand side sorts terms of the product $a_{22}^{\text{new}}\bar{S}$ into those of degree $< 2t$ and of degree $\geq 2t$.

With respect to claim (11)

$$\begin{aligned}\Lambda S &= a_{22}^{\text{new rec}}S \\ &= x^{\deg a_{22}^{\mathbf{new}}+2t-1}a_{22}^{\text{new}}(x^{-1})\bar{S}(x^{-1}) \\ &\overset{(16')}{=} x^{\deg a_{22}^{\mathbf{new}}+2t-1}(\rho_{\text{new}}(x^{-1}) - a_{21}^{\text{new}}(x^{-1})x^{-2t}) \\ &= x^{\nu_{\mathbf{new}}+\mu_{\mathbf{new}}-2}\rho_{\text{new}}^{\text{rec}} - x^{\deg a_{22}^{\mathbf{new}}-1}a_{21}^{\text{new}}(x^{-1}) \\ &\equiv -x^{\deg a_{22}^{\mathbf{new}}-1}a_{21}^{\text{new}}(x^{-1}) \bmod x^{\nu_{\mathbf{new}}+\mu_{\mathbf{new}}-2}\end{aligned}$$

and then

$$-x^{\deg a_{22}^{\mathbf{new}}-1}a_{21}^{\text{new}}(x^{-1}) = \Omega^{\text{new}}$$

as well as $-\rho_{\text{new}}^{\text{rec}} = \Psi^{\text{new}}$ by (8). Further (16$'$) indicates that

$$\deg a_{22}^{\text{new}}\bar{S} = \deg a_{21}^{\text{new}}x^{2t}$$

and, therefore,

$$\begin{aligned}\deg a_{21}^{\text{new}} &\leq \nu_{\text{new}} - 1 + 2t - 1 - 2t \\ &< \nu_{\text{new}} - 1.\end{aligned}$$

Thus claim (11) follows.

We have already seen that $-\rho_{\text{new}}^{\text{rec}} = \Psi^{\text{new}}$. Finally, let μ_{new} be the row index of the next discrepancy $\delta_{\text{new}} \neq 0$. By comparison of the coefficients of ΛS and $a_{22}^{\text{new}}\bar{S}$ from (16$'$) where $a_{22}^{\text{new rec}} = \Lambda$ it can be seen that $\delta_{\text{new}} = \text{lc}\,\rho_{\text{new}}$ and $\deg \rho_{\text{new}} = 2t - \mu_{\text{new}}$. Note also that the BMA and the EA stop at the same time.

ACKNOWLEDGMENT

The authors wish to thank P. H. Sørensen and T. Grønne [10] for their contributions.

REFERENCES

[1] E. R. Berlekamp, *Algebraic Coding Theory*. New York: McGraw-Hill, 1968.
[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, 1975.
[4] J. L. Dornstetter, "On the equivalence between Berlekamp's and Euclid's algorithms," *IEEE Trans. Inform. Theory*, vol. IT-33, pp. 428–431, May 1987.
[5] G.-L. Feng and K. K. Tzeng, "A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 1274–1287, Sept. 1991.
[6] G.-M. Greuel, G. Pfister, and H. Schönemann. (1998, June) Singular Version 1.2 User Manual. Centre for Computer Algebra, University of Kaiserslautern. [Online]. Available: http://www.singular. uni-kl.de
[7] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1977.
[8] O. Pretzel, *Error-Correcting Codes and Finite Fields*, ser. Oxford Applied Mathematics and Computing Science. Oxford, U.K.: Oxford Univ. Press, 1992.
[9] M. E. O'Sullivan, The key equation for one-point codes and efficient error evaluation, unpublished manuscript, 1998.
[10] P. H. Sørensen and T. Grønne, "Indkodning og dekodning af cykliske koder i dellegemer," Master's thesis, Tech. Univ. Denmark, Lyngby, Aug. 1995.