Efficient Decoding of Reed-Solomon Codes Beyond Half the Minimum Distance*

Ron M. Roth Gitit Ruckenstein

Computer Science Department
Technion, Haifa 32000, Israel

e-mail: {ronny, gitit}@cs.technion.ac.il

Abstract

A list decoding algorithm is presented for $[n, k]$ Reed-Solomon (RS) codes over $GF(q)$, which is capable of correcting more than $\lfloor (n-k)/2 \rfloor$ errors. Based on a previous work of Sudan, an extended key equation (EKE) is derived for RS codes, which reduces to the classical key equation when the number of errors is limited to $\lfloor (n-k)/2 \rfloor$. Generalizing Massey's algorithm that finds the shortest recurrence that generates a given sequence, an algorithm is obtained for solving the EKE in time complexity $O(\ell \cdot (n-k)^2)$, where ℓ is a design parameter, typically a small constant, which is an upper bound on the size of the list of decoded codewords (the case $\ell = 1$ corresponds to classical decoding of up to $\lfloor (n-k)/2 \rfloor$ errors where the decoding ends with at most one codeword). This improves on the time complexity $O(n^3)$ needed for solving the equations of Sudan's algorithm by a naive Gaussian elimination. The polynomials found by solving the EKE are then used for reconstructing the codewords in time complexity $O((\ell \log^2 \ell)\, k\, (n + \ell \log q))$ using root-finders of degree-ℓ univariate polynomials.

Keywords: Decoding; Key equation; Polynomial interpolation; Reed-Solomon codes.

1 Introduction

An $[n, k]$ code $\mathcal{C}$ over the finite field $F = GF(q)$ is a k-dimensional linear subspace of F^n. It is well-known that $\mathcal{C}$ can recover uniquely and correctly any pattern of τ errors or less if and only if $\tau \leq \lfloor (d-1)/2 \rfloor$, where d is the minimum Hamming distance between any two distinct elements (codewords) of $\mathcal{C}$. Most of the decoding algorithms for error-correcting codes indeed work under the assumption that the number of errors is at most $\lfloor (d-1)/2 \rfloor$.

*This work was done in part at Hewlett-Packard Laboratories, Palo Alto, California, and Hewlett-Packard Laboratories, Israel. This work was supported by grant No. 95-522 from the United-States–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

An $[n, k]$ (generalized) Reed-Solomon (in short, RS) code $\mathcal{C}_{\mathrm{RS}}$ over F is defined by

$$\mathcal{C}_{\mathrm{RS}} = \{ \bar{c} = (f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n)) \ : \ f(x) \in F_k[x] \} , \tag{1}$$

where $\alpha_1, \alpha_2, \ldots \alpha_n$ are $n < q$ prescribed distinct nonzero elements in F, referred to as *code locators*, and $F_k[x]$ stands for the set of all polynomials of degree $< k$ in the indeterminate x over F. RS codes are known to be maximum-distance separable (MDS), i.e., their minimum Hamming distance equals $n-k+1$ [11, Ch. 11].

Let $\bar{v} = (v_1, v_2, \ldots, v_n)$ be the received word. Several efficient RS decoding algorithms are known for correcting up to $\tau = \lfloor (n-k)/2 \rfloor$ errors. The Berlekamp-Massey algorithm [2, Section 7.4],[12] and the algorithm of Sugiyama et *al.* [3, Ch. 7],[18] comprise three steps:

Step D0: Computing syndrome elements $S_0, S_1, \ldots, S_{n-k-1}$ from the received word $\bar{v}$. The syndrome elements are commonly written in a form of a polynomial $S(x) = \sum_{i=0}^{n-k-1} S_i x^i$.

Step D1: Solving the *key equation* (in short, KE) of RS codes

$$\Lambda(x) \cdot S(x) \ \equiv \ \Omega(x) \pmod{x^{n-k}}$$

for the *error-locator polynomial* $\Lambda(x)$ of degree $\le \tau$ and for the *error-evaluator polynomial* $\Omega(x)$ of degree $< \tau$ ($\le n-k-\tau$).

Step D2: Locating the errors (through computing the roots of $\Lambda(x)$) and finding their values (from $\Lambda(x)$ and $\Omega(x)$).

To specify the time complexity of algorithms, we count operations of elements in F and use the notation $h_1(m) = O(h_2(m))$ to mean that there exist positive constants c and m_0 such that $h_1(m) \le c \cdot h_2(m)$ for all integers $m \ge m_0$ [1, Ch. 1]. Basing on techniques presented in [1, Ch. 8], procedures for evaluating polynomials in $F_n[x]$ at n points in F, as well as interpolating such polynomials given their values at n points in F, can be carried out in time complexity $O(n \log^2 n)$.

Denote by $[n]$ the set of integers $\{1, 2, \ldots, n\}$. The syndrome elements in Step D0 are computed by

$$S_i \ = \ \sum_{j=1}^{n} v_j \eta_j \alpha_j^i \tag{2}$$

where

$$\eta_j^{-1} \ = \ \prod_{r \in [n] \setminus \{j\}} (\alpha_j - \alpha_r) \tag{3}$$

(see Proposition 4.1 below). Using a result by Kaminski et *al.* in [9], it can be shown that the time complexity of computing (2) is the same as that of evaluating a polynomial in $F_n[x]$ at the code locators α_j, $j \in [n]$, and is therefore $O(n \log^2 n)$.

Step D2 can be carried out through Chien search [4] and Forney's algorithm (see [3]). Both algorithms involve evaluation of polynomials at given points, implying that reconstructing the codewords in Step D2 can be executed in time complexity $O(n \log^2 n)$.

Writing $\Lambda(x) = \sum_{s=0}^{n-k-\tau} \Lambda_s x^s$, we obtain from the KE the following set of τ homogeneous equations in the coefficients of $\Lambda(x)$,

$$\sum_{s=0}^{n-k-\tau} \Lambda_s \cdot S_{i-s} \; = \; 0 \; , \quad n{-}k{-}\tau \leq i < n{-}k \; , \tag{4}$$

where $\Lambda_0 = 1$. Massey's algorithm [12] solves these equations in time complexity $O(\tau^2)$, and acceleration methods allow to reduce the complexity of Step D1 to $O((n{-}k)\log^2(n{-}k))$ [3].

Step D0 is commonly applied while the received word is read into the decoder, and Step D2 is carried out while the correct codeword is flushed out. On the other hand, Step D1 is executed only after the whole received word has been read but before any output is generated; hence, minimizing the complexity of Step D1 means reducing the *latency* of the decoder.

In a recent paper [17], Sudan presented a decoding algorithm for $[n, k]$ RS codes of the form (1) that corrects more than $\lfloor (n{-}k)/2 \rfloor$ errors. In this case, the decoding might not be unique, so the decoder's task is to find the list of codewords that differ from the received word in no more than τ locations. This task is referred to in the literature as *list decoding* [5].

If Gaussian elimination is used as an equation solver in Sudan's algorithm, then its time complexity is $O(n^3)$. The algorithm can be described as a method for interpolating the polynomial $f(x) \in F_k[x]$ through the set of points $\{(\alpha_j, v_j)\}_{j=1}^n$, while taking into account that some of the values v_j may be erroneous. The interpolation is done by computing from the received word $\bar{v}$ a nonzero bivariate polynomial $Q(x, y)$ that vanishes at the points $\{(\alpha_j, v_j)\}_{j=1}^n$ and then finding the linear factors, $y - g(x)$, of $Q(x, y)$. The codewords to which $\bar{v}$ is decoded are computed from the polynomials $g(x)$ through re-encoding.

Viewing Sudan's algorithm, it is intriguing to find its relationship with the classical RS decoding algorithms; specifically, can his algorithm be somehow regarded as an extension of the previously-known RS decoding algorithms, and, if so, can we reduce the time complexity of the counterpart of Step D1 in his algorithm from cubic in n to quadratic in $n{-}k$?

In this work, we provide positive answers to those questions. We use the algorithm in [17] as a basis for developing an *extended key equation* (in short, EKE) which reduces to the (classical) KE when $\tau = \lfloor (n{-}k)/2 \rfloor$. The EKE involves an integer parameter ℓ which provides an upper bound on the number of codewords that can be at Hamming distance $\leq \tau$ from any received word; we refer to those codewords as the *consistent codewords*. Specifically, the EKE takes the form

$$\sum_{t=1}^{\ell} \Lambda^{(t)}(x) \cdot x^{(t-1)(k-1)} \cdot S^{(t)}(x) \; \equiv \; \Omega(x) \quad (\text{mod } \; x^{n-k}) \; ,$$

where $S^{(1)}(x), S^{(2)}(x), \ldots, S^{(\ell)}(x)$ are 'syndrome polynomials' that can be computed from the received word and $\Lambda^{(1)}(x), \Lambda^{(2)}(x), \ldots, \Lambda^{(\ell)}(x)$, and $\Omega(x)$ are polynomials that satisfy certain degree constraints. Those polynomials are then used to find the consistent codewords as a final step. The KE is a special case of the EKE when $\ell = 1$.

To compute the polynomials $\{\Lambda^{(t)}(x)\}_{t=1}^{\ell}$, we first translate the EKE into a set of τ homogeneous linear equations and then apply a generalization of Massey's algorithm that takes advantage of the special structure of the equations and solves them in time complexity $O(\ell\tau^2)$, which is quadratic in $n{-}k$ assuming that ℓ is fixed (e.g., $\ell = 2$).

In the last step of our decoding algorithm, which appears also in Sudan's algorithm, the codewords are reconstructed from the polynomials $(\Lambda^{(t)}(x))_{t=1}^{\ell}$ through a procedure for finding linear factors of bivariate polynomials. We show that those factors can be found in time complexity $O((\ell \log^2 \ell)\, k\, (n + \ell \log q))$ using root-finders of degree-ℓ univariate polynomials. There are known general algorithms for factoring multivariate polynomials [19]; yet, those algorithms have relatively large complexity when applied to the particular application in this paper. We also mention the recent work of Gao and Shokrollahi [7] where they study the (slightly different) problem of finding linear factors of bivariate polynomials $Q(x, y)$ where the polynomial arithmetic is carried out modulo a power of x (in which case more solutions may exist).

We comment that increasing the number τ of correctable errors by increasing the number ℓ of consistent codewords in Sudan's algorithm, as well as in ours, requires decreasing the maximum rate k/n of the RS codes to which the algorithm is applied. For example, when $\ell = 2$ we will have $\tau > (n{-}k)/2$ only when $k \leq (n{+}1)/3$. An improvement of Sudan's work [17] has been recently reported by Guruswami and Sudan in [8], where the constraints on the rates have been relaxed.

This work is organized as follows. For the sake of completeness, we review Sudan's algorithm in Section 2. In Section 3, the EKE is derived, and then, in Section 4, we present an algorithm that solves the EKE for the polynomials $(\Lambda^{(t)}(x))_{t=1}^{\ell}$. In Section 5, we show how the consistent codewords can be efficiently computed from those polynomials. Complexity analysis is given in Section 6 and, finally, a summary and examples are presented in Section 7.

2 Sudan's algorithm

Given a prescribed upper bound ℓ on the number of consistent codewords, Sudan's algorithm corrects any error pattern of up to τ errors for

$$\tau \;=\; n - (m+1) - \ell(k-1)\;, \tag{5}$$

where m is the smallest nonnegative integer satisfying

$$(m+1)(\ell+1) + (k-1)\binom{\ell+1}{2} \;>\; n\;. \tag{6}$$

We will further assume throughout this paper that ℓ, k, and n satisfy the inequality

$$\ell \;+\; (k-1)\binom{\ell+1}{2} \;\leq\; n\;. \tag{7}$$

Indeed, it can be verified that if (7) did not hold, then, for the same values of k, n, and τ, (5) and (6) would still be satisfied if we decreased ℓ by 1 and chose $m = k{-}1$; this means that, for the given k, n and τ, we could assume a shorter list of consistent codewords. In particular, setting $\ell = 2$ in (7) implies $k \leq (n{+}1)/3$. Note that from (6) and the minimality of m we obtain

$$(m+1)(\ell+1) + (k-1)\binom{\ell+1}{2} \;\leq\; n + \ell + 1\;. \tag{8}$$

If we now fix k, n, and ℓ, then from (5) it follows that τ will be maximized if we choose the smallest possible m that satisfies (6). For $\ell = 1$ (the classical case) we thus obtain $m = \lceil (n-k)/2 \rceil$ and $\tau = n - k - m = \lfloor (n-k)/2 \rfloor$. For $\ell = 2$ we obtain $m = \lceil (n+1)/3 \rceil - k$ (assuming $k \leq (n+1)/3$, in conjunction with (7)) and $\tau = n+1 - 2k - m = \lfloor 2(n+1)/3 \rfloor - k$.

Define

$$N_t = n - \tau - t(k-1)\,, \tag{9}$$

where by (5), (6), and (8), we have

$$\tau + 1 \;\leq\; \sum_{t=1}^{\ell} N_t \;\leq\; \tau + \ell + 1\,. \tag{10}$$

Sudan's algorithm is based on the following two lemmas, taken from [17].

Lemma 2.1 *Whenever (5)–(6) hold, there exists a nonzero bivariate polynomial $Q(x, y)$ over F that satisfies*

$$Q(x, y) = \sum_{t=0}^{\ell} Q^{(t)}(x) y^t \;, \quad \deg Q^{(t)}(x) < N_t \;, \tag{11}$$

and

$$Q(\alpha_j, v_j) = 0 \;, \quad \forall j \in [n]\,. \tag{12}$$

Lemma 2.2 *Let $Q(x, y)$ be a nonzero bivariate polynomial over F satisfying (11)–(12) and let $f(x) \in F_k[x]$ be such that $f(\alpha_j) = v_j$ for at least $n - \tau$ locators α_j. Then $Q(x, f(x))$ is identically zero.*

Sudan's algorithm consists of the following steps:

Input: received word $\bar{v} = (v_1, v_2, \ldots, v_n)$.

Step S1: Find a nonzero bivariate polynomial $Q(x, y)$ over F satisfying (11)–(12).

Step S2: Output all the polynomials $g(x) \in F_k[x]$ such that

- $y - g(x)$ is a factor of $Q(x, y)$, and —
- $g(\alpha_j) = v_j$ for at least $n - \tau$ locators α_j.

3 Extended key equation

In this section, we derive the EKE based on Sudan's algorithm. Let $\bar{v} = (v_1, v_2, \ldots, v_n)$ be the received word and let $V(x)$ be the unique polynomial in $F_n[x]$ such that $V(\alpha_j) = v_j$ for all $j \in [n]$; the existence and the uniqueness of the polynomial $V(x)$ are implied by the Lagrange interpolation theorem [10, Ch. 1].

Lemma 3.1 *Let $Q(x,y) = \sum_{t=0}^{\ell} Q^{(t)}(x) y^t$ be a bivariate polynomial that satisfies (11). Then $Q(x,y)$ satisfies (12) if and only if there exists a polynomial $B(x)$ over F for which*

$$\sum_{t=0}^{\ell} Q^{(t)}(x) \cdot (V(x))^t \;=\; B(x) \cdot \prod_{j=1}^{n} (x - \alpha_j) \;, \tag{13}$$

where

$$\deg B(x) \;<\; \ell(n{-}k) - \tau \;. \tag{14}$$

Proof: By the definition of $V(x)$, an equivalent condition to (12) is that the univariate polynomial $Q(x, V(x))$ vanishes at each of the code locators α_j, $j \in [n]$. Alternatively, there is a polynomial $B(x)$ over F satisfying (13). Now, by (9) and (11),

$$\deg Q^{(t)}(x) \cdot (V(x))^t \;<\; n - \tau + t(n{-}k) \;, \quad t \in [\ell] \;,$$

implying that

$$\deg Q(x, V(x)) \;<\; n - \tau + \ell(n{-}k)$$

and that $B(x)$ must satisfy (14). ∎

Let $Q(x,y)$ be a polynomial satisfying (11). We will introduce the short-hand bivariate notation $Q^*(x,y) = \sum_{t=1}^{\ell} Q^{(t)}(x) y^t$. Define the following polynomials obtained by reversing the order of coefficients in $V(x)$, $\prod_{j=1}^{n}(x - \alpha_j)$, and $(Q^{(t)}(x))_{t=1}^{\ell}$, respectively:

$$\begin{array}{lcll} \overline{V}(x) & = & x^{n-1} V(x^{-1}) \;, & \\ G(x) & = & \prod_{j=1}^{n} (1 - \alpha_j x) \;, & \text{and} \\ \Lambda^{(t)}(x) & = & x^{N_t - 1} Q^{(t)}(x^{-1}) \;, & t \in [\ell] \;. \end{array} \tag{15}$$

Lemma 3.2 *Let $Q^*(x,y) = \sum_{t=1}^{\ell} Q^{(t)}(x) y^t$ satisfy*

$$\deg Q^{(t)}(x) < N_t \;, \quad t \in [\ell] \;, \tag{16}$$

and let $\overline{V}(x)$, $G(x)$, and $(\Lambda^{(t)}(x))_{t=1}^{\ell}$ be defined by (15). There exists a (unique) polynomial $Q^{(0)}(x)$ such that $Q(x,y) = Q^{(0)}(x) + Q^*(x,y)$ satisfies (11)–(12) if and only if there exists a (unique) polynomial $\overline{B}(x) \in F_{\ell(n-k)-\tau}[x]$ such that

$$\sum_{t=1}^{\ell} \Lambda^{(t)}(x) \cdot x^{(\ell-t)(n-k)} \cdot (\overline{V}(x))^t \;\equiv\; \overline{B}(x) \cdot G(x) \pmod{x^{\ell(n-k)}} \;. \tag{17}$$

Proof: We start with the "only if" part. Suppose that $Q^{(0)}(x)$ is such that $Q(x,y) = Q^{(0)}(x) + Q^*(x,y)$ satisfies (11)–(12). By Lemma 3.1, there exists a (unique) polynomial $B(x)$ satisfying (13)–(14). Define $\overline{B}(x)$ to be the polynomial of degree $< \ell(n{-}k) - \tau$ obtained by reversing the order of coefficients in $B(x)$, namely,

$$\overline{B}(x) \;=\; x^{\ell(n-k)-\tau-1} B(x^{-1}) \;. \tag{18}$$

Consider the (highest) $\ell(n{-}k)$ coefficients of x^i in both sides of (13) for i in the range $n - \tau \le i < n - \tau + \ell(n{-}k)$. Since $\deg Q^{(0)}(x) \le N_0 = n - \tau$, each of those coefficients in

$\sum_{t=1}^{\ell} Q^{(t)}(x)(V(x))^t$ must be equal to its counterpart in the right-hand side of (13). If we now reverse the order of coefficients in both sides of (13), then the coefficients of $1, x, \ldots, x^{\ell(n-k)-1}$ should be identical in the resulting two polynomials. Formally,

$$x^{n-\tau+\ell(n-k)-1} \cdot \sum_{t=1}^{\ell} Q^{(t)}(x^{-1}) \cdot (V(x^{-1}))^t \;\equiv\; x^{\ell(n-k)-\tau-1} \cdot B(x^{-1}) \cdot \prod_{j=1}^{n} (1-\alpha_j x) \pmod{x^{\ell(n-k)}} . \tag{19}$$

Using the definitions (15) and (18), the equation (19) becomes (17).

As for the "if" part, suppose that (17) holds for $\overline{B}(x) \in F_{\ell(n-k)-\tau}[x]$. Define $B(x)$ to be the polynomial in $F_{\ell(n-k)-\tau}[x]$ that is obtained by reversing the order of coefficients in $\overline{B}(x)$. If we reverse each side of (17), we get two polynomials of degree less than $n - \tau + \ell(n-k)$ that may differ only in their lowest $n - \tau = N_0$ coefficients. In other words, there exists some (unique) polynomial $Q^{(0)}(x) \in F_{N_0}[x]$ such that $Q^{(0)}(x) + \sum_{t=1}^{\ell} Q^{(t)}(x)(V(x))^t = B(x) \cdot \prod_{j=1}^{n}(x - \alpha_j)$. The bivariate polynomial $Q(x, y) = Q^{(0)}(x) + Q^*(x, y)$ thus satisfies (11)–(12), as required. ■

For $t \in [\ell]$, let $S_\infty^{(t)}(x) = \sum_{i=0}^{\infty} S_i^{(t)} x^i$ be the formal power series which is defined by

$$\frac{(\overline{V}(x))^t}{G(x)} \;=\; x^{(t-1)(n-1)} \cdot S_\infty^{(t)}(x) + U^{(t)}(x) , \tag{20}$$

where $U^{(t)}(x) \in F_{(t-1)(n-1)}[x]$. Indeed, since $G(0) = 1$, (20) is well-defined (see, for example, [10, Ch. 8]). Further, define the univariate syndrome polynomials $S^{(t)}(x) = \sum_{i=0}^{n-2-t(k-1)} S_i^{(t)} x^i$ and the bivariate syndrome polynomial $S(x, y) = \sum_{t=1}^{\ell} S^{(t)}(x) y^t$.

Proposition 3.3 *Let $Q^*(x, y) = \sum_{t=1}^{\ell} Q^{(t)}(x) y^t$ satisfy (16) and let $(\Lambda^{(t)}(x))_{t=1}^{\ell}$ be defined by (15). There exists a (unique) polynomial $Q^{(0)}(x)$ such that $Q(x, y) = Q^{(0)}(x) + Q^*(x, y)$ satisfies (11)–(12) if and only if there exists a (unique) polynomial $\Omega(x) \in F_{n-k-\tau}[x]$ that satisfies the EKE*

$$\sum_{t=1}^{\ell} \Lambda^{(t)}(x) \cdot x^{(t-1)(k-1)} \cdot S^{(t)}(x) \;\equiv\; \Omega(x) \pmod{x^{n-k}} . \tag{21}$$

Proof: By Lemma 3.2, all we need to prove is that the EKE (21) is equivalent to (17). Substituting

$$(\overline{V}(x))^t = (x^{(t-1)(n-1)} \cdot S_\infty^{(t)}(x) + U^{(t)}(x)) \cdot G(x)$$

into (17) and rearranging terms yields

$$\sum_{t=1}^{\ell} \Lambda^{(t)}(x) \cdot x^{(\ell-t)(n-k)+(t-1)(n-1)} \cdot S_\infty^{(t)}(x) \cdot G(x) \;\equiv\; \tilde{V}(x) \cdot G(x) \pmod{x^{\ell(n-k)}} , \tag{22}$$

where

$$\tilde{V}(x) \;=\; \overline{V}(x) - \sum_{t=1}^{\ell} \Lambda^{(t)}(x) \cdot x^{(\ell-t)(n-k)} \cdot U^{(t)}(x) .$$

Now,

$$(\ell-t)(n-k) + (t-1)(n-1) \;=\; (\ell-1)(n-k) + (t-1)(k-1)$$

and

$$\deg \Lambda^{(t)}(x) \cdot x^{(\ell-t)(n-k)} \cdot U^{(t)}(x) \;<\; N_t + (\ell-t)(n-k) + (t-1)(n-1) - 1 = \ell(n-k) - \tau \;.$$

Hence, $\deg \tilde{V}(x) < \ell(n-k) - \tau$ if and only if $\deg \overline{V}(x) < \ell(n-k) - \tau$. Since the polynomials $G(x)$ and $x^{\ell(n-k)}$ are relatively prime, we can rewrite (22) as

$$\sum_{t=1}^{\ell} \Lambda^{(t)}(x) \cdot x^{(\ell-1)(n-k)+(t-1)(k-1)} \cdot S_\infty^{(t)}(x) \;\equiv\; \tilde{V}(x) \pmod{x^{\ell(n-k)}} \;. \tag{23}$$

The left-hand side of (23) is divisible by $x^{(\ell-1)(n-k)}$ and, therefore, so must $\tilde{V}(x)$. Letting $\Omega(x) = \tilde{V}(x)/x^{(\ell-1)(n-k)}$, we get

$$\sum_{t=1}^{\ell} \Lambda^{(t)}(x) \cdot x^{(t-1)(k-1)} \cdot S^{(t)}(x) \;\equiv\; \Omega(x) \pmod{x^{n-k}} \;, \tag{24}$$

where we have replaced $S_\infty^{(t)}(x)$ by $S^{(t)}(x)$, since the coefficients of $S_\infty^{(t)}(x)$ that actually appear in (23) are those that correspond to the powers x^i for $0 \le i < n - 1 + t(k-1)$. Observe that $\deg \Omega(x) = \deg \tilde{V}(x) - (\ell-1)(n-k)$, so we indeed have $\deg \Omega(x) < n-k-\tau$ if and only if $\deg \overline{V}(x) < \ell(n-k) - \tau$. ■

4 Solving the EKE

In view of the analysis presented in Section 3, Step S1 in Sudan's algorithm splits into two steps, which may be denoted by D0 and D1, similarly to their counterparts in the classical decoding scheme. In Step D0, the bivariate syndrome polynomial $S(x,y) = \sum_{t=1}^{\ell} S^{(t)}(x)y^t$ is computed, and in Step D1, the bivariate polynomial $Q(x,y) = \sum_{t=0}^{\ell} Q^{(t)}(x)y^t$ is found by solving the EKE. The following proposition shows that the syndrome elements can be computed in Step D0 using a formula which is a generalization of (2).

Proposition 4.1 *Let $\eta_1, \eta_2, \ldots, \eta_n$ be as in (3) and $S_\infty^{(t)}(x) = \sum_{i=0}^{\infty} S_i^{(t)} x^i$ as in (20). Then*

$$S_i^{(t)} \;=\; \sum_{j=1}^{n} v_j^t \eta_j \alpha_j^i \;, \quad t \in [\ell] \;, \quad i \ge 0 \;. \tag{25}$$

Proof: For each $t \in [\ell]$, let $\overline{V}^{(t)}(x) \in F_n[x]$ and $\tilde{U}^{(t)}(x) \in F_{(t-1)(n-1)}[x]$ be the unique polynomials that satisfy

$$(\overline{V}(x))^t \;=\; \tilde{U}^{(t)}(x)G(x) + \overline{V}^{(t)}(x) \;. \tag{26}$$

Since $\overline{V}^{(t)}(\alpha_j^{-1}) \;=\; (\overline{V}(\alpha_j^{-1}))^t \;=\; \left(v_j \alpha_j^{-(n-1)}\right)^t$, we can express $\overline{V}^{(t)}(x)$ as an interpolation polynomial by

$$\overline{V}^{(t)}(x) \;=\; \sum_{j=1}^{n} v_j^t \alpha_j^{(1-t)(n-1)} \eta_j \prod_{r \in [n] \setminus \{j\}} (1 - \alpha_r x) \;.$$

This, in turn, implies

$$\frac{\overline{V}^{(t)}(x)}{G(x)} = \sum_{j=1}^{n} \frac{v_j^t \alpha_j^{(1-t)(n-1)}}{1-\alpha_j x} = \sum_{i=(1-t)(n-1)}^{\infty} \sum_{j=1}^{n} v_j^t \eta_j \alpha_j^i x^{i+(t-1)(n-1)} . \qquad (27)$$

Combining (20), (26), and (27) we obtain

$$\begin{aligned} x^{(t-1)(n-1)} \cdot S_\infty^{(t)}(x) &= \frac{(\overline{V}(x))^t}{G(x)} - U^{(t)}(x) = \frac{\overline{V}^{(t)}(x)}{G(x)} - U^{(t)}(x) + \tilde{U}^{(t)}(x) \\ &= x^{(t-1)(n-1)} \cdot \sum_{i=(1-t)(n-1)}^{\infty} \sum_{j=1}^{n} v_j^t \eta_j \alpha_j^i x^i - U^{(t)}(x) + \tilde{U}^{(t)}(x) , \end{aligned}$$

that is, $S_\infty^{(t)}(x) = \sum_{i=0}^{\infty} S_i^{(t)} x^i = \sum_{i=0}^{\infty} \sum_{j=1}^{n} v_j^t \eta_j \alpha_j^i x^i$. ■

Writing $\Lambda^{(t)}(x) = \sum_{s=0}^{N_t-1} \Lambda_s^{(t)} x^s$ and $Q^{(t)}(x) = \sum_{s=0}^{N_t-1} Q_s^{(t)} x^s = \sum_{s=0}^{N_t-1} \Lambda_s^{(t)} x^{N_t-1-s}$, from (21) we obtain

$$\sum_{t=1}^{\ell} \sum_{s=0}^{N_t-1} \Lambda_s^{(t)} \cdot S_{(t-1)(k-1)+i-s}^{(t)} = 0 , \quad n-k-\tau \le i < n-k , \qquad (28)$$

(compare with (4)) or

$$\sum_{t=1}^{\ell} \sum_{s=0}^{N_t-1} Q_s^{(t)} \cdot S_{i+s}^{(t)} = 0 , \quad 0 \le i < \tau . \qquad (29)$$

For two bivariate polynomials $a(x) = \sum_{s,t} a_{s,t} x^s y^t$ and $b(x) = \sum_{s,t} b_{s,t} x^s y^t$, we define the inner product $\langle a(x,y), b(x,y) \rangle$ by $\sum_{s,t} a_{s,t} b_{s,t}$. Using this notation, (29) takes the following short-hand form

$$\langle x^\rho Q^*(x,y), S(x,y) \rangle = 0 , \quad 0 \le \rho < \tau . \qquad (30)$$

The number of linear equations expressed by (30) (or (29)) is τ, namely, the maximum number of errors that we attempt to correct, and by (10) it is smaller than the number of unknowns in those equations. Comparing (30) to the set of linear equations derived directly from (11)–(12), we conclude that both the number of equations and the number of unknowns have been reduced by $n-\tau$.

The algorithm in Figure 1 solves (30) for the coefficients of $Q^*(x,y) = \sum_{t=1}^{\ell} Q^{(t)}(x) y^t$ under the degree constraints (16) assuming that the syndrome polynomial $S(x,y)$ is given. Equivalently, this algorithm is a method for solving the EKE (21).

Our algorithm and its analysis are based on the approach of Massey [12] and Sakata [16], as presented in [15]. Even though our algorithm bears similarity to Sakata's algorithm—and both Sakata's algorithm and ours generalize Massey's algorithm—the two generalizations are not the same. Our proof of the algorithm in Figure 1 is based on that of an algorithm by Feng and Tzeng in [6].

Let $\prec$ denote the (total) order defined in [15] over the set of pairs $\{(i,t) \mid i \in \mathbb{N}, t \in [\ell]\}$; that is,

$$(i,t) \prec (i',t') \quad \text{if and only if} \quad \left\{ \begin{array}{c} i + t(k-1) < i' + t'(k-1) \\ \text{or} \\ (\ i + t(k-1) = i' + t'(k-1) \quad \text{and} \quad t < t'\ ) \end{array} \right\} .$$

The notation $(i,t) \preceq (i',t')$ means that either $(i,t) = (i',t')$ or $(i,t) \prec (i',t')$, and $\mathsf{succ}_{\prec}(i,t)$ is the pair that immediately follows (i,t) with respect to the order defined by $\prec$. For a nonzero bivariate polynomial $T(x,y) = \sum_{t=1}^{\ell} \sum_i T_i^{(t)} x^i y^t$, we define $\mathsf{lead}(T(x,y))$ as the maximal pair (μ,ν), with respect to $\prec$ for which $T_\mu^{(\nu)} \neq 0$ and denote $\mathsf{lead}_x(T(x,y)) = \mu$ and $\mathsf{lead}_y(T(x,y)) = \nu$ ($\mathsf{lead}(0)$ is defined to be $(-\infty,-\infty)$).

The algorithm in Figure 1 scans the syndrome elements $S_\mu^{(\nu)}$ in the order defined by $\prec$ on (μ,ν), and maintains up to ℓ bivariate polynomials $T_1(x,y), T_2(x,y), \ldots, T_\ell(x,y)$, where $\mathsf{lead}_y(T_\nu(x,y)) = \nu$. An invariant of the algorithm is that $\langle x^{\rho'} \cdot T_\nu(x,y), S(x,y) \rangle = 0$ for $0 \le \rho' < \rho$. Lines 6 and 11 update $T_\nu(x,y)$ so that it generates the syndrome element $S_\mu^{(\nu)}$ as well. Note, however, that in our algorithm, unlike in Sakata's, $T_\nu(x,y)$ is *not* required to generate syndrome elements in rows other than ν. The update of $T_\nu(x,y)$ in line 6 does not change the value of $\mathsf{lead}(T_\nu(x,y))$, whereas in line 11, $\mathsf{lead}(T_\nu(x,y))$ grows (with respect to $\prec$), and the value of $T_\nu(x,y)$ right before the update is stored as a polynomial named $R(x,y)$. Unlike Sakata's algorithm (but similarly to Massey's algorithm), only one stored polynomial $R(x,y)$ is needed in every stage of our algorithm in order to update any of the polynomials $T_1(x,y), T_2(x,y), \ldots, T_\ell(x,y)$.

Basing on Lemma 2.1 and on the analysis in the current and previous sections, the validity of the algorithm in Figure 1 is implied by Proposition 4.2 below, the proof of which can be found in the appendix. Note that for our purpose of solving (30), any one of the polynomials returned by the algorithm will suffice.

Proposition 4.2 *For every $s \in [\ell]$, if there exists some nonzero polynomial $Q^*(x,y)$ with $\mathsf{lead}_x(Q^*(x,y)) = s$ that satisfies (16) and (30), then the algorithm in Figure 1 returns such a polynomial with a minimal value (with respect to $\prec$) of $\mathsf{lead}(Q^*(x,y))$.*

To complete Step D1, we need to compute the polynomial $Q^{(0)}(x)$ for which $Q^{(0)}(x) + Q^*(x,y) = Q(x,y)$ satisfies (11)–(12) (see Proposition 3.3). Since

$$Q^{(0)}(\alpha_j) \;=\; -Q^*(\alpha_j, v_j) \;=\; -\sum_{t=1}^{\ell} Q^{(t)}(\alpha_j) v_j^t\,, \quad j \in [n]\,, \tag{31}$$

the polynomial $Q^{(0)}(x)$ can be obtained by interpolation once we compute the right-hand side of (31) for $N_0 = n - \tau$ pairs (α_j, v_j).

5 Reconstructing the codewords

In this section, we present an efficient implementation of Step S2 in Sudan's algorithm given that the polynomial $Q(x,y) = \sum_{t=0}^{\ell} Q^{(t)}(x) y^t$ is known. Specifically, our goal is to compute all the *consistent polynomials* $f(x) \in F_k[x]$ for which the vector $(f(\alpha_1), f(\alpha_2), \ldots, f(\alpha_n))$ is at Hamming distance $\le \tau$ from the received word $\bar{v}$. Step S2 in Sudan's algorithm applies Lemma 2.2 and looks for all the polynomials $g(x) \in F_k[x]$ such that $Q(x, g(x))$ is identically zero; the polynomials $g(x)$ will be referred to as the *y-roots* of $Q(x,y)$ over the polynomial ring $F[x]$. The *y-degree* of $Q(x,y)$ is the degree of $Q(x,y)$ as a polynomial in y over $F[x]$.

The recursive procedure *Reconstruct* in Figure 2 computes a set of up to ℓ polynomials in $F_k[x]$ that contains as a subset all the y-roots of $Q(x,y)$; as such, this set also contains all

Input: Bivariate polynomial $S(x,y) = \sum_{t=1}^{\ell} S^{(t)}(x) y^t$, where $S^{(t)}(x) \in F_{\tau+N_t-1}[x]$;

Data structures:

ℓ bivariate polynomials $T_s(x,y) = \sum_{t=1}^{\ell} T_s^{(t)}(x) y^t$, $s \in [\ell]$, where $T_s^{(t)}(x) \in F_{N_t}[x]$;
index (μ, ν) such that $\nu \in [\ell]$ and $\mu \in \{0, 1, \ldots, \tau + N_\nu - 2\}$;
variable $\Delta \in F$;
integer variables ρ, r;
bivariate polynomial $R(x,y) = \sum_{t=1}^{\ell} R^{(t)}(x) y^t$, where $R^{(t)}(x) \in F_{N_t}[x]$;
set of indexes $\mathcal{L} \subseteq [\ell]$;

Initialize:

for $s = 1$ to ℓ do $T_s(x,y) \leftarrow y^s$;
$(\mu, \nu) \leftarrow (0, 1)$;
$R(x,y) \leftarrow 0$;
$r \leftarrow -1$;
$\mathcal{L} \leftarrow [\ell]$;

while $\mathcal{L} \neq \emptyset$ do $\{$
  if $\nu \in \mathcal{L}$ then $\{$
    $\rho \leftarrow \mu - \mathsf{lead}_x(T_\nu(x,y))$;
    $\Delta \leftarrow \langle x^\rho \cdot T_\nu(x,y), S(x,y) \rangle$;
    if $\Delta = 0$ or $\rho \leq r$ then $\{$
      if $\Delta \neq 0$ then $T_\nu(x,y) \leftarrow T_\nu(x,y) - \Delta \cdot x^{r-\rho} \cdot R(x,y)$;
      if $\rho = \tau - 1$ then $\{$
        output $T_\nu(x,y)$; $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\nu\}$; $\}$
    $\}$
    else /* $\Delta \neq 0$ and $\rho > r$ */
      if $\mu - r < N_\nu$ then $\{$

$$\begin{pmatrix} T_\nu(x,y) \\ R(x,y) \end{pmatrix} \leftarrow \begin{pmatrix} x^{\rho - r} \cdot T_\nu(x,y) - \Delta \cdot R(x,y) \\ \Delta^{-1} \cdot T_\nu(x,y) \end{pmatrix};$$

        $r \leftarrow \rho$;
      $\}$
      else $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\nu\}$;
  $\}$
  $(\mu, \nu) \leftarrow \mathsf{succ}_\prec(\mu, \nu)$;
$\}$

Figure 1: Algorithm for solving (30).

the consistent polynomials. The procedure *Reconstruct* is initially called with parameters $(Q, k, 0)$, where $Q = Q(x, y) = \sum_{t=0}^{\ell} Q^{(t)}(x)y^t$ is a nonzero bivariate polynomial with y-degree $\le \ell$, e.g., a polynomial that satisfies (11). The validity of *Reconstruct*, as established in Proposition 5.2 below, is based on the following lemma, which shows that the coefficients of a y-root $g(x)$ of $Q(x, y)$ can all be calculated recursively as roots of univariate polynomials.

---

procedure *Reconstruct*(bivariate polynomial $Q(x, y)$, integer k, integer i)

```
/*      A global array φ[0,...,k−1] is assumed.
        The initial call needs to be with Q(x,y) ≠ 0, k > 0, and i = 0.
*/
R1      find the largest integer r for which Q(x,y)/x^r is still a (bivariate) polynomial;
R2      M(x,y) ← Q(x,y)/x^r;
R3      find all the roots in F of the univariate polynomial M(0,y);
R4      for each of the distinct roots γ of M(0,y) do   {
R5          φ[i] ← γ;
R6          if i = k−1 then output φ[0],...,φ[k−1];
            else  {
R7              M^(x,y) ← M(x,y+γ);
R8              M~(x,y) ← M^(x,xy);
R9              Reconstruct(M~(x,y),k,i+1);
            }
        }
```

Figure 2: Recursive procedure for finding a superset of the consistent polynomials.

Lemma 5.1 *Let $g(x) = \sum_{s \ge 0} g_s x^s$ be a y-root of a nonzero bivariate polynomial $Q(x, y)$ over F. For $i \ge 0$, let $\psi_i(x) = \sum_{s \ge i} g_s x^{s-i}$ and let $Q_i(x, y)$ and $M_i(x, y)$ be defined inductively by $Q_0(x, y) = Q(x, y)$,*

$$M_i(x, y) = x^{-r_i} Q_i(x, y) \quad \textit{and} \quad Q_{i+1}(x, y) = M_i(x, xy + g_i) \ , \quad i \ge 0 \ ,$$

where r_i is the largest integer such that x^{r_i} divides $Q_i(x, y)$. Then, for every $i \ge 0$,

$$Q_i(x, \psi_i(x)) \ = \ 0 \quad \textit{and} \quad M_i(0, g_i) \ = \ 0 \ ,$$

while $M_i(0, y) \ne 0$.

Proof: First observe that the y-degrees of the polynomials $Q_i(x, y)$ are the same for all i and, so, $Q_i(x, y) \ne 0$ and r_i is well-defined. Also, since x does not divide $M_i(x, y)$ then $M_i(0, y) \ne 0$. Next, we prove that $Q_i(x, \psi_i(x)) = 0$ by induction on i, where the induction base $i = 0$ is obvious. As for the induction step, if $\psi_i(x)$ is a y-root of $Q_i(x, y)$, then $\psi_{i+1}(x) = (\psi_i(x) - g_i)/x$ is a y-root of $Q_i(xy + g_i)$ and hence of $Q_{i+1}(x, y) = M_i(x, xy + g_i) = x^{-r_i} Q_i(x, xy + g_i)$. Finally, by substituting $x = 0$ in $M_i(x, \psi_i(x)) = x^{-r_i} Q_i(x, \psi_i(x)) = 0$ we obtain $M_i(0, g_i) = M_i(0, \psi_i(0)) = 0$. ∎

Proposition 5.2 *Let $Q(x, y)$ be a nonzero bivariate polynomial. Every y-root in $F_k[x]$ of $Q(x, y)$ is found by the call Reconstruct$(Q, k, 0)$.*

Proof: Using the notations of Lemma 5.1, there is a recursion descend in *Reconstruct* where recursion level i is called with the parameters (Q_i, k, i) and $\phi[i]$ is set to g_i. ■

We show in Proposition 6.4 below that *Reconstruct* outputs at most ℓ polynomials. To complete the decoding, each of those polynomials is evaluated at the code locators, producing up to ℓ different codewords from which we select the consistent codewords as those that are at Hamming distance $\leq \tau$ from the received word $\bar{v}$. Our decoding algorithm for the RS code $\mathcal{C}_{\mathrm{RS}}$ defined by (1) is summarized in Figure 3.

Preliminary Step:
Given n, k, and distinct code locators $\alpha_1, \alpha_2, \ldots, \alpha_n$ in F, fix an upper bound on the allowed number ℓ of consistent codewords. Compute ℓ, m, and τ so that (5) is maximized, subject to (6) and the upper bound on ℓ. Compute the multipliers $\eta_1, \eta_2, \ldots, \eta_n$ by (3). Define N_t by (9).

Input: $\bar{v} = (v_1, v_2, \ldots, v_n)$.

Step D0:
Compute the syndrome elements

$$S_i^{(t)} = \sum_{j=1}^{n} v_j^t \eta_j \alpha_j^i \ , \quad t \in [\ell] \ , \quad 0 \leq i < n + t(k-1) - 1 \ .$$

Step D1:

- Using the algorithm in Figure 1, compute a polynomial $Q^*(x, y) = \sum_{t=1}^{\ell} Q^{(t)}(x) y^t$ that solves the equations
$$\langle x^\rho Q^*(x, y), S(x, y) \rangle = 0 \ , \quad 0 \leq \rho < \tau \ ,$$
under the degree constraints
$$\deg Q^{(t)}(x) < N_t \ , \quad t \in [\ell] \ .$$
- Compute the polynomial $Q^{(0)}(x) \in F_{N_0}[x]$ by (31).

Step D2:

- Call *Reconstruct* with the initial parameters $Q(x, y) = Q^{(0)}(x) + Q^*(x, y)$, k, and 0.
- For each $g(x) \in F_k[x]$ in the output of *Reconstruct*, compute the corresponding codeword $\bar{c} = (g(\alpha_1), g(\alpha_2), \ldots, g(\alpha_n))$. Output $\bar{c}$ if the Hamming distance between $\bar{c}$ and $\bar{v}$ is τ or less.

Figure 3: Summary of the decoding algorithm.

6 Complexity Analysis

Following is a complexity analysis of our algorithm, as presented in Figure 3. The time complexity of Step D0, in which all the coefficients of the bivariate syndrome polynomial

$S(x, y)$ are computed by (25) is at most ℓ times the complexity of computing the syndrome in the classical case using (2), namely $O(\ell n \log^2 n)$ (see the discussion in Section 1).

The time and space complexities of the algorithm in Figure 1, which solves (30) in Step D1, are given in Proposition 6.1 below. Throughout the proof of the proposition, an iteration of the algorithm in Figure 1 in which the variable ν takes the value s will be called an *iteration of type* s. If in addition $s \in \mathcal{L}$, the iteration will said to be *nontrivial*.

Proposition 6.1 *The time complexity of the algorithm in Figure 1 is $O(\ell\tau^2)$ and its space complexity is $O(\ell\tau)$.*

Proof: In every nontrivial iteration of the algorithm of Figure 1, the most time-consuming steps are the computations of Δ in line 4 and the polynomial updates in lines 6 and 11. The time complexity of all those computations is linear in the number of nonzero coefficients of the respective polynomial $T_s(x, y)$. The check in line 10 guarantees that $\mathsf{lead}(T_s(x, y)) \prec (N_s, s)$ and, so, the number of nonzero coefficients in $T_s(x, y)$ never exceeds $\sum_{t=1}^{\ell} N_t$. By (10), the number of coefficients in $T_s(x, y)$, as well as the time complexity of every computation in lines 4, 6, and 11, is $O(\tau)$.

As shown in the proof of Lemma A.2 in the appendix, the number of nontrivial iterations of type s is smaller than $N_s + \tau$ for every $s \in [\ell]$. The overall number of nontrivial iterations of all types throughout the execution of the algorithm can therefore be bounded from above by $\ell\tau + \sum_{s=1}^{\ell} N_s = O(\ell\tau)$, where the equality follows from (10). The time complexity of the whole algorithm is thus $O(\ell\tau^2)$.

As for the space complexity, most of the memory is allocated for the $\ell + 1$ polynomials $(T_s(x, y))_{s=1}^{\ell}$ and $R(x, y)$, where for each of them we allocate $\sum_{t=1}^{\ell} N_t = O(\tau)$ coefficients over F. ∎

Next we turn to the time complexity of computing the polynomial $Q^{(0)}(x)$ using (31). Note that $\deg Q^{(t)}(x) \le \deg Q^{(0)}(x) < N_0$ for every $t \in [\ell]$ and that by (8)

$$N_0 \le \frac{2(n + \ell + 1)}{\ell + 1} . \tag{32}$$

Hence, for every $t \in [\ell]$, we can evaluate $Q^{(t)}(x)$ at $n - \tau$ locators α_j in time complexity $O((n/\ell) \log^2 n)$. So, the right-hand side of (31) can be evaluated for $n - \tau$ pairs (α_j, v_j) in time $O(n \log^2 n)$; this will also be the time complexity of interpolating $Q^{(0)}(x)$ out of those computed values.

The rest of this section is devoted to proving Proposition 6.6 below in which the time and space complexities of the recursive procedure *Reconstruct* in Figure 2 are established. In each of the recursion levels i of *Reconstruct*, we find roots of nonzero polynomials $M(0, y) = M_i(0, y)$ with degree at most ℓ. It may seem at first that the number of root extractions could grow exponentially. However, Lemma 6.2 below shows that having more than one root of $M_i(0, y)$ in a given recursion level is compensated by having a multiple root in the respective polynomial $M_{i-1}(0, y)$ in the previous recursion level.

Lemma 6.2 *Let $M_1(x, y) = \sum_{t=0}^{\ell} M^{(t)}(x) y^t$ be a nonzero bivariate polynomial over F and let $\gamma \in F$ be a y-root of multiplicity h of $M(0, y)$. Define $M_2(x, y) = x^{-r} M_1(x, xy + \gamma)$, where r is the largest integer for which $x^r | M_1(x, xy + \gamma)$. Then $\deg M_2(0, y) \le h$.*

Proof: Similarly to the notations in Figure 2, we denote

$$\widehat{M}(x,y) \;=\; \sum_{t=0}^{\ell} \widehat{M}^{(t)}(x) y^t \;=\; M_1(x, y+\gamma)$$

and

$$\widetilde{M}(x,y) \;=\; \sum_{t=0}^{\ell} \widetilde{M}^{(t)}(x) y^t \;=\; M_1(x, xy+\gamma) \; .$$

Since γ is a root of multiplicity h of $M_1(0,y)$, then $y=0$ is a root of multiplicity h of $\widehat{M}(0,y)$. Thus, $\widehat{M}^{(t)}(0)=0$ for $0 \le t < h$ and $\widehat{M}^{(h)}(0) \neq 0$; equivalently, x divides $\widehat{M}^{(t)}(x)$ for $0 \le t < h$ but it does not divide $\widehat{M}^{(h)}(x)$. Noting that $\widetilde{M}^{(t)}(x) = \widehat{M}^{(t)}(x) x^t$ it follows that x divides $\widetilde{M}(x,y)$ but x^{h+1} does not.

The largest integer r such that x^r divides $\widetilde{M}(x,y)$ thus satisfies $1 \le r \le h$. Now,

$$M_2(x,y) \;=\; \frac{\widetilde{M}(x,y)}{x^r} \;=\; \sum_{t=0}^{r} \frac{\widehat{M}^{(t)}(x) x^t}{x^r} y^t + \sum_{t=r+1}^{\ell} \widehat{M}^{(t)}(x) x^{t-r} y^t \; .$$

Substituting $x=0$ in $M_2(x,y)$ yields a univariate polynomial $M_2(0,y)$ of degree $\le r \le h$. ■

Corollary 6.3 *Consider the special case where the very first execution of Step R2 in Reconstruct results in a polynomial $M(x,y)$ such that all the roots in F of $M(0,y)$ are simple. Then the polynomials obtained in Step R3 throughout all the subsequent recursive calls have degree at most 1, meaning that their roots can be found simply by solving linear equations over F.*

As for the general case, we have the following upper bounds.

Proposition 6.4 *Suppose that Reconstruct is initially called with the parameters $(Q,k,0)$, where $Q = Q(x,y) = \sum_{t=0}^{\ell} Q^{(t)}(x) y^t$ is a nonzero bivariate polynomial. Then the number of output polynomials produced by Reconstruct is at most ℓ and the overall number of recursive calls made to Reconstruct (in Step R9) is at most $\ell(k-1)$.*

Proof: For $0 \le s < k$, denote by ω_s the <u>sum</u> of the degrees of all the polynomials $M(0,y)$ that Step R3 is applied to when i equals s. When $s=0$, the degree of $M(0,y) = M_0(0,y)$ is at most ℓ, and by Lemma 6.2 we have $\omega_s \le \omega_{s-1}$ for every $s \in [k-1]$. It can therefore be proved by induction on s that $\omega_s \le \ell$. As a result, *Reconstruct* generates at most $\omega_{k-1} \le \ell$ outputs, and the number of executions of Step R9 is $\sum_{s=0}^{k-2} \omega_s \le \ell(k-1)$. ■

Lemma 6.5 *Assume a call to Reconstruct as in Proposition 6.4, and further assume that $Q(x,y)$ satisfies (11). Then the y-degree of each of the bivariate polynomials computed in any of the recursion levels is at most ℓ, and its x-degree is at most $m + \ell(k-1) = O(n/\ell)$.*

Proof: It is easy to see that Steps R2, R7, and R8 never increase the y-degree. As for the x-degree, let $M_i(x,y)$ be a polynomial computed in Step R2 in recursion level i and write $M_i(x) = \sum_{t=0}^{\ell} M_i^{(t)}(x) y^t$. The degree of $M_i^{(t)}(x)$ can increase with i only as a result of Step R8, and it is easy to check by induction on i that

$$\deg M_i^{(t)}(x) \;<\; N_t + ti \;=\; N_0 - t(k-1-i)$$

for every $0 \le i < k$. So, $\deg M_i^{(t)}(x) < N_0 \le 2(n+\ell+1)/(\ell+1)$ (see (32)). ■

Proposition 6.6 *Assume a call to Reconstruct as in Proposition 6.4, where $Q(x,y)$ satisfies (11). The time complexity of such an application to its full recursion depth is $O((\ell\log^2\ell)\,k\,(n+\ell\log q))$, and it can be implemented using space of overall size $O(n)$.*

Proof: Each execution of Step R1, R2, or R8 has time complexity which is proportional to the number of coefficients in the polynomials involved in that step. By Lemma 6.5, this number is $O(n)$. By Proposition 6.4, each of those steps is executed at most ℓk times throughout the recursion levels. Therefore, the contribution of Steps R1, R2, and R8 to the overall complexity of *Reconstruct* is $O(\ell kn)$.

By Proposition 6.4, the sum of the degrees of all the polynomials $M(0,y)$ that Step R3 is applied to at the ith recursion level is at most ℓ. The roots in $F=GF(q)$ of a polynomial of degree u can be found in expected time complexity $O((u^2\log^2 u)\,\log q)$ [10, Ch. 4],[14]; also recall that there are known efficient deterministic algorithms for root extraction when the characteristic of F is small [2, Ch. 10], and root extraction is particularly simple when $\ell=2$ [11, pp. 277–278]. Therefore, for any $0\le i<k$, the executions of Step R3 at recursion level i have accumulated time complexity $O((\ell^2\log^2\ell)\,\log q)$, and the contribution of Step R3 to the overall complexity of *Reconstruct* is $O((\ell^2\log^2\ell)\,k\,\log q)$.

Let $M(x,y)$ be the polynomial that Step R7 is applied to at some iteration level i. Writing $M(x,y)$ as a polynomial in x, we get

$$M(x,y) \;=\; \sum_{s=0}^{N_0-1} M^{[s]}(y)x^s\;.$$

If ℓ_s is the smallest integer t such that $N_t+ti\le s$, then it is easy to check that $\deg M^{[s]}(y)<\ell_s$ for every $0\le s<N_0$. The computed polynomial $\widehat{M}(x,y)$ in Step R7 can be written as

$$\widehat{M}(x,y) \;=\; \sum_{s=0}^{N_0-1}\widehat{M}^{[s]}(y)x^s \;=\; M(x,y+\gamma) \;=\; \sum_{s=0}^{N_0-1} M^{[s]}(y+\gamma)x^s\;,$$

and, so, we can compute each polynomial $\widehat{M}^{[s]}(y)$ by interpolating the values $M^{[s]}(\alpha+\gamma)$ at ℓ_s distinct points $\alpha\in F$. Therefore, each polynomial $\widehat{M}^{[s]}(y)$ can be found in time complexity $O(\ell_s\log^2\ell_s)$. We now observe that $\ell_s\le\ell$ and that

$$\sum_{s=0}^{N_0-1}\ell_s \;=\; \sum_{t=0}^{\ell}(N_t+ti) \;\le\; (\ell+1)N_0 \;\le\; 2(n+\ell+1)\;,$$

the latter inequality following from (32). Hence, each execution of Step R7 has time complexity $O(n\log^2\ell)$, and the contribution of Step R7 to the overall complexity of *Reconstruct* is therefore $O(kn\ell\log^2\ell)$.

Summing up the contributions of the steps of *Reconstruct*, the time complexity of an application of *Reconstruct* to its full recursion depth is $O((\ell\log^2\ell)\,k\,(n+\ell\log q))$. As for the space complexity, notice that the input parameter $Q(x,y)$ can be recomputed from r, γ, and the parameter $\widetilde{M}(x,y)$ to the next recursion level. So, instead of keeping the polynomials in each recursion level, we can recompute them after each execution of Step R9. Therefore, *Reconstruct* can be implemented using space of overall size $O(n)$. ∎

Finally, we point out that the time complexity of computing the consistent codewords out of the output polynomials of *Reconstruct* is $O(\ell n\log^2 n)$, as the re-encoding involves the evaluation of those polynomials at the code locators.

7 Summary

The three main decoding steps in our algorithm, as it appears in Figure 3, are denoted D0, D1, and D2, to point out their relationship with the classical decoding algorithms as outlined in Section 1. Steps D0 and D1 replace Step S1 in Sudan's algorithm that finds the bivariate polynomial $Q(x, y)$.

As shown in Section 6, the time complexity of Step D0 is $O(\ell n \log^2 n)$ and the time complexity of Step D1 is $O(\ell \tau^2)$. Compared to classical decoding, those figures are larger by a factor of ℓ. Step D2, which is presented in Section 5, is an efficient application of Step S2 in Sudan's algorithm and has time complexity $O((\ell \log^2 \ell)\, k\, (n + \ell \log q))$.

In cases where the particular use of the decoding algorithm does not dictate an upper bound on ℓ, we can select the value of ℓ that maximizes (5) subject to (6). By (7) we will thus have $\ell = O(\sqrt{n/k})$. For this value of ℓ, the time complexities of Steps D0, D1, and D2 are $O(n^{3/2} k^{-1/2} \log^2 n)$, $O((n-k)^2 \sqrt{n/k})$, and $O((\sqrt{nk} + \log q) n \log^2(n/k))$, respectively.

The next example is provided to illustrate the various decoding steps; the parameters were selected to be small enough so that the computation can be more easily verified by the reader.

Example 7.1 Let $F = GF(19)$, $n = 18$, and $k = 2$. When maximizing (5) we get $\tau = 12$ for $\ell = 4$ and $m = 1$. We select $\alpha_j = j$ and obtain from (3) that $\eta_j = -\alpha_j$.

Suppose we encode the polynomial $f(x) = 18 + 14x$ by (1) and get the following transmitted codeword, error vector, and received word:

$$\begin{aligned}
\bar{c} &= (\,13, 8, 3, 17, 12, 7, 2, 16, 11, 6, 1, 15, 10, 5, 0, 14, 9, 4\,)\\
\bar{e} &= (\,11, 16, 17, 12, 17, 0, 0, 2, 14, 0, 0, 0, 3, 0, 14, 8, 11, 15\,)\\
\bar{v} &= (\,5, 5, 1, 10, 10, 7, 2, 18, 6, 6, 1, 15, 13, 5, 14, 3, 1, 0\,)
\end{aligned}$$

The computation of the syndrome elements in Step D0 results in the following coefficients of the polynomials $(S^{(t)}(x) = \sum_{i=0}^{\tau+N_t-2} S_i^{(t)} x^i)_{t=1}^4$:

$$\begin{aligned}
S^{(1)}(x) &: \quad (\,13, 14, 5, 11, 3, 4, 10, 14, 13, 14, 11, 14, 17, 4, 0, 2\,)\\
S^{(2)}(x) &: \quad (\,4, 8, 14, 18, 9, 18, 5, 13, 11, 6, 8, 8, 16, 0, 12\,)\\
S^{(3)}(x) &: \quad (\,3, 12, 5, 7, 10, 18, 4, 14, 0, 14, 18, 11, 16, 3\,)\\
S^{(4)}(x) &: \quad (\,14, 13, 0, 13, 10, 1, 9, 3, 7, 8, 11, 0, 7\,)
\end{aligned}$$

Step D1 yields the following polynomials $Q^{(t)}(x)$,

$$\begin{aligned}
Q^{(0)}(x) &= 4 + 12x + 5x^2 + 11x^3 + 8x^4 + 13x^5\\
Q^{(1)}(x) &= 14 + 14x + 9x^2 + 16x^3 + 8x^4\\
Q^{(2)}(x) &= 14 + 13x + x^2\\
Q^{(3)}(x) &= 2 + 11x + x^2\\
Q^{(4)}(x) &= 17
\end{aligned}$$

where $Q^*(x, y) = \sum_{t=1}^{4} Q^{(t)}(x) y^t$ is the first polynomial returned by the algorithm in Figure 1 and $Q^{(0)}(x)$ is computed by (31).

When applying *Reconstruct* in Step D2 to $Q(x, y) = \sum_{t=0}^{4} Q^{(t)}(x)y^t$, we obtain the following four different solutions $g(x)$ for $f(x)$:

$$18 + 14x,\quad 18 + 15x,\quad 14 + 16x,\quad 8 + 8x\ .$$

The first two solutions share the same constant coefficient, 18, which is a multiple root of the polynomial $M(0, y) = Q(0, y) = 4 + 14y + 14y^2 + 2y^3 + 17y^4$ at recursion level $i = 0$. The first solution for $f(x)$ corresponds to the correct codeword. The second solution is not even a y-root of $Q(x, y)$ (yet, as commented by one of the reviewers, it is a prefix of the y-root $18 + 15x + 10x^2$). The third solution is a y-root of $Q(x, y)$ but not a consistent polynomial (the respective codeword has Hamming distance 15 from $\bar{v}$). And the fourth solution is a consistent polynomial but does not correspond to the correct codeword.

In this example, the algorithm in Figure 1 yields a second polynomial $Q(x, y)$ which is given by

$$\begin{array}{rcl}
Q^{(0)}(x) &=& 8 + 12x^2 + 9x^3 + 8x^4 \\
Q^{(1)}(x) &=& 5 + 14x + 7x^2 + 15x^3 + 4x^4 \\
Q^{(2)}(x) &=& 12 + 12x + 15x^2 + 4x^3 \\
Q^{(3)}(x) &=& 9 + 10x + 14x^2 \\
Q^{(4)}(x) &=& 13 + x
\end{array}$$

and the respective output of *Reconstruct* is

$$18 + 14x,\quad 13 + 9x,\quad 10 + x,\quad 8 + 8x\ ,$$

with only the first and fourth polynomials being y-roots of $Q(x, y)$ (as well as being consistent polynomials); the remaining irreducible factor of $Q(x, y)$ is $(13 + x)y^2 + (5 + 18x + 17x^2)y + (18 + 6x + 15x^2)$. ∎

In the example above, the common factors of the two solutions for $Q(x, y)$ correspond to the two (and all) consistent polynomials. However, there are examples where the common y-roots in $F_k[x]$ of all the polynomials $Q(x, y)$ generated in Figure 1 contain—in addition to the consistent polynomials—also inconsistent ones.

We comment that the connection between the error vector $\bar{e}$ and the polynomials that appear in the EKE seems to be less obvious than in the classical case; recall that when $\ell = 1$, $\bar{e}$ can be obtained from $\Lambda(x)$ and $\Omega(x)$ through Chien search [4] and Forney's algorithm [3]. It would be interesting to find such an intimate relationship between $\bar{e}$ and the polynomials that appear in the EKE also when $\ell > 1$.

We pointed out that Sudan's algorithms, as well as the decoding algorithm Figure 3 can correct more than $(n-k)/2$ errors only when $k \leq (n+1)/3$. Clearly, in many (if not most) practical applications, higher code rates are used. Therefore, it would be interesting to investigate whether the EKE presented in this paper and the algorithm in Section 4 can be generalized to higher rates. In particular, connecting the results of this work with the improvements presented in [8], might be possible (Nielsen and Høholdt have been working recently independently on a different approach to accelerate [8]; see [13]).

A Appendix

We present here the proof of Propositions 4.2, which makes use of Lemma A.1 and Lemma A.2 below. We omit the proof of Lemma A.1 as it is similar to proofs already contained in [12], [15], and [16].

Lemma A.1 *Let* $S(x,y) = \sum_{t=1}^{\ell} S^{(t)}(x)y^t$, $T(x,y) = \sum_{t=1}^{\ell} T^{(t)}(x)y^t$, *and* $R(x,y) = \sum_{t=1}^{\ell} R^{(t)}(x)y^t$ *be bivariate polynomials, let* Δ *and* δ *be elements of* F, *and let* r *and* ρ *be nonnegative integers such that* $\mathsf{lead}(x^r \cdot R(x,y)) \prec \mathsf{lead}(x^\rho \cdot T(x,y))$ *and*

$$\langle x^a \cdot T(x,y), S(x,y)\rangle = 0\ , \quad 0 \le a < \rho\ , \quad \textit{and} \quad \langle x^\rho \cdot T(x,y), S(x,y)\rangle = \Delta\ ,$$

$$\langle x^a \cdot R(x,y), S(x,y)\rangle = 0\ , \quad 0 \le a < r\ , \quad \textit{and} \quad \langle x^r \cdot R(x,y), S(x,y)\rangle = \delta\ .$$

(a) *Suppose that* $\rho \le r$ *and define* $A(x,y) = T(x,y) - \frac{\Delta}{\delta} \cdot x^{r-\rho} \cdot R(x,y)$. *Then*

1. $\langle x^a \cdot A(x,y), S(x,y)\rangle = 0, \quad 0 \le a \le \rho$;
2. $\mathsf{lead}(A(x,y)) = \mathsf{lead}(T(x,y))$; *and* —
3. $\mathsf{lead}(x^r \cdot R(x,y)) \prec \mathsf{lead}(x^{\rho+1} \cdot A(x,y))$.

(b) *Suppose that* $\rho > r$ *and define* $A(x,y) = x^{\rho-r} \cdot T(x,y) - \frac{\Delta}{\delta} \cdot R(x,y)$. *Then*

1. $\langle x^a \cdot A(x,y), S(x,y)\rangle = 0, \quad 0 \le a \le r$;
2. $\mathsf{lead}(A(x,y)) = \mathsf{lead}(x^{\rho-r} \cdot T(x,y))$; *and* —
3. $\mathsf{lead}(x^\rho \cdot T(x,y)) \prec \mathsf{lead}(x^{r+1} \cdot A(x,y))$.

As in Section 6, we use the term *iteration of type* s to mean an iteration of the algorithm in Figure 1 in which the variable ν takes the value s. Such an iteration is *nontrivial* if $s \in \mathcal{L}$.

Lemma A.2 *The algorithm in Figure 1 terminates, and each of its output polynomials, if there exist any, may serve as a polynomial* $Q^*(x,y)$ *that satisfies (30) under the constraint (16).*

Proof: By the conditions in lines 7 and 10, the number of nontrivial iterations of type s throughout an execution of the algorithm is always smaller than $N_s+\tau$. The set $\mathcal{L}$ thus always becomes empty and the algorithm terminates. Now, if a polynomial $T_s(x,y) = \sum_{t=0}^{\ell} T_s^{(t)}(x)y^t$ is returned as output in line 8, then, by Lemma A.1 and the condition in line 7, it may serve as $Q^*(x,y)$ in (30). By the condition in line 10 we have $\mathsf{lead}(T_s(x,y)) \prec (N_s, s)$, and from the definition of the order $\prec$ we get that $\deg T_s^{(t)}(x) < N_t$ for every $t \in [\ell]$. The polynomial $T_s(x,y)$ thus satisfies the degree constraint (16). ∎

Proof of Proposition 4.2: We reformulate the requirements of Proposition 4.2 through the $\tau \times \sum_{t=1}^{\ell} N_t$ matrix $\mathcal{S}$ defined as follows. The columns of $\mathcal{S}$ are indexed by ordered pairs (ϵ, σ), where $\sigma \in [\ell]$ and $0 \le \epsilon < N_\sigma$, and are ordered from left to right with respect to the order $\prec$ on their indexes. A column in $\mathcal{S}$ indexed by (ϵ, σ) is called a *column of type* σ and is given by $\mathcal{S}_{(\epsilon,\sigma)} = (S_\epsilon^{(\sigma)}, S_{\epsilon+1}^{(\sigma)}, \ldots, S_{\epsilon+\tau-1}^{(\sigma)})^T$. For instance, when $\ell = 2$ the matrix $\mathcal{S}$ takes the form

$$\left(\begin{array}{cccc|cccccc} S_0^{(1)} & S_1^{(1)} & \cdots & S_{k-2}^{(1)} & S_{k-1}^{(1)} & S_0^{(2)} & \cdots & S_{N_1-1}^{(1)} & S_{N_2-1}^{(2)} \\ S_1^{(1)} & S_2^{(1)} & \cdots & S_{k-1}^{(1)} & S_k^{(1)} & S_1^{(2)} & \cdots & S_{N_1}^{(1)} & S_{N_2}^{(2)} \\ \vdots & \vdots & \iddots & \vdots & \vdots & \vdots & \iddots & \vdots & \vdots \\ S_{\tau-1}^{(1)} & S_\tau^{(1)} & \cdots & S_{\tau+k-3}^{(1)} & S_{\tau+k-2}^{(1)} & S_{\tau-1}^{(2)} & \cdots & S_{\tau+N_1-2}^{(1)} & S_{\tau+N_2-2}^{(2)} \end{array}\right) .$$

We show that if $\mathcal{S}_{(\epsilon,s)}$ is the first (leftmost) column of type s in $\mathcal{S}$ that is linearly dependent on previous columns in $\mathcal{S}$, then a polynomial $Q^*(x,y)$ with $\mathsf{lead}(Q^*(x,y)) = (\epsilon, s)$ is returned as output by the algorithm. The respective linear dependency is given by the coefficients of $Q^*(x,y)$; namely, the coefficient of $x^i y^t$ in $Q^*(x,y)$ multiplies the column $\mathcal{S}_{(i,t)}$ in the linear combination. By Lemma A.2, it suffices to show that if $\mathsf{lead}(T_s(x,y))$ takes in the course of the algorithm a value *greater* than (ϵ, s) (with respect to $\prec$), then the column $\mathcal{S}_{(\epsilon,s)}$ is linearly independent of previous columns in $\mathcal{S}$ (this applies also to the case where $\mathsf{lead}(T_s(x,y))$ was supposed to take a value which is at least (N_s, s), thereby reaching line 13).

By Lemma A.1, line 11 is the only place in the algorithm where $\mathsf{lead}(T_\nu(x,y))$ can change. Therefore, all we have to show is that whenever line 11 is reached with given values of ν, ρ, r, and $\mathsf{lead}(T_\nu(x,y)) = (\xi, \nu)$, then each of the columns $\mathcal{S}_{(p,\nu)}$, $\xi \le p < \xi + \rho - r$, is linearly independent of the columns standing to its left in $\mathcal{S}$.

Let s_i, ρ_i, r_i, Δ_i, (ξ_i, s_i), and $\{T_{\sigma,i}(x,y)\}_{\sigma=1}^{\ell}$ denote the values of ν, ρ, r, Δ, $\mathsf{lead}(T_\nu(x,y))$, and $\{T_\sigma(x,y)\}_{\sigma=1}^{\ell}$ right before the ith execution of line 11; note that $r_i = \rho_{i-1}$. Let Y_i be the upper-left sub-matrix of $\mathcal{S}$ that consists of the columns $\mathcal{S}_P$ for $(0,1) \preceq P \prec (\xi_i{+}\rho_i{-}r_i, s_i)$, shortened to their first $\rho_i + 1$ entries. Define the sets $\mathcal{D}_i$ inductively as follows:

$$\mathcal{D}_0 = \emptyset \quad \text{and} \quad \mathcal{D}_i \;=\; \mathcal{D}_{i-1} \cup \{(\xi_i + j, s_i) \;:\; 0 \le j < \rho_i - r_i\} \;;$$

observing that $|\mathcal{D}_i| = |\mathcal{D}_{i-1}| + \rho_i - \rho_{i-1}$ and that $\rho_0 \equiv r_1 = -1$, we have $|\mathcal{D}_i| = \rho_i + 1$. We denote by Z_i the $(\rho_i{+}1) \times (\rho_i{+}1)$ sub-matrix of Y_i consisting of the columns of the latter indexed by $\mathcal{D}_i$. It can be verified that Z_i consists of all the columns of Y_i, except, possibly, a certain number of rightmost columns of each type *other* than s_i in Y_i.

The rest of the proof is devoted to showing that every column in Z_i is linearly independent of columns that stand to its left in Y_i. We show this in two steps:

- We first show that each column of Y_i that is not a column of Z_i is linearly dependent on previous columns in Y_i.

- We then prove that $\mathrm{rank}(Y_i) = \rho_i + 1$.

(We point out that in the classical case of $\ell = 1$, the matrices Y_i and Z_i coincide, thereby making the first step vacuous.)

Let (ϵ, σ) be the index of a column in Y_i that does not belong to Z_i. It is clear that $\sigma \ne s_i$. Since $(\epsilon, \sigma) \notin \mathcal{D}_i$, it follows that $\mathsf{lead}(T_{\sigma,i}(x,y)) \preceq (\epsilon, \sigma)$. Let a be an integer such that $0 \le a \le \rho_i$. We distinguish between two cases:

Case 1: $(\epsilon + a, \sigma) \prec (\xi_i + \rho_i, s_i)$. Here, the index (μ, ν) reaches the value $(\epsilon + a, \sigma)$ with a polynomial $T_\sigma(x,y)$ with $\mathsf{lead}(T_\sigma(x,y)) = \mathsf{lead}(T_{\sigma,i}(x,y)) \equiv (h, \sigma)$ before (μ, ν) takes the value $(\xi_i + \rho_i, s_i)$ with the polynomial $T_\nu(x,y) = T_{s_i,i}(x,y)$. In this case we have

$$\langle x^a \cdot x^{\epsilon-h} \cdot T_{\sigma,i}(x,y), S(x,y) \rangle \;=\; 0 \; .$$

Case 2: $(\xi_i + \rho_i, s_i) \prec (\epsilon + a, \sigma)$. To the smallest a' such that

$$\langle x^{a'} \cdot x^{\epsilon-h} \cdot T_{\sigma,i}(x,y), S(x,y) \rangle \;\neq\; 0 \; ,$$

we can apply Lemma A.1(a) with $T(x,y) \leftarrow x^{\epsilon-h} \cdot T_{\sigma,i}(x,y)$, $\rho \leftarrow a'$, $R(x,y) \leftarrow T_{s_i,i}(x,y)$, and $r \leftarrow \rho_i$, to obtain a polynomial $A(x,y)$ with $\mathsf{lead}(A(x,y)) = (\epsilon, \sigma)$ for which

$$\langle x^b \cdot A(x,y), S(x,y) \rangle \;=\; 0 \; , \quad 0 \le b \le a' \; .$$

By repeatedly applying Lemma A.1 as in Case 2, we can update the polynomial $A(x,y)$ while keeping $\mathsf{lead}(A(x,y)) = (\epsilon, \sigma)$ so that it satisfies

$$\langle x^a \cdot A(x,y), S(x,y) \rangle \;=\; 0 \; , \quad 0 \le a \le \rho_i \; .$$

The last equation means that the column $(Y_i)_{(\epsilon,\sigma)}$ is linearly dependent on columns standing to its left in Y_i. This completes the first step of our proof.

In our second step, we show that $\mathrm{rank}(Y_i) = \rho_i + 1$ by applying Gaussian elimination to the columns of Y_i, where the linear combinations applied to the columns will be determined by $T_{s_i,i}(x,y)$. By Lemma A.1 it follows that the sequence of updates carried out on $T_\nu(x,y)$ in the algorithm to produce $T_{s_i,i}(x,y)$ guarantees that

$$\langle x^a \cdot T_{s_i,i}(x,y), S(x,y) \rangle \;=\; \begin{cases} 0 & \text{if } 0 \le a < \rho_i \\ \Delta_i \neq 0 & \text{if } a = \rho_i \end{cases} \; .$$

This implies that when using the coefficients of $T_{s_i,i}(x,y)$ to compute a linear combination of the columns $(Y_i)_P$, $(0,1) \preceq P \preceq (\xi_i, s_i)$, we end up with a column vector $(0,0,\ldots,0,\Delta_i)^T \in F^{\rho_i+1}$.

Next, we take advantage of the Hankel-like structure of $\mathcal{S}$ and generalize our previous argument as follows. For every j in the range $0 \le j < \rho_i - \rho_{i-1}$, the linear combination of $(Y_i)_P$, $(0,1) \preceq P \preceq (\xi_i + j, s_i)$, yields a column vector $(0,0,\ldots,0,\Delta_i,\ldots)^T \in F^{\rho_i+1}$, where the number of leading zeros is $\rho_i - j$. We thus have,

$$\mathrm{rank}(Y_i) \;\ge\; \mathrm{rank}(Y_{i-1}) + \rho_i - \rho_{i-1} \; ,$$

which readily implies by induction the desired result. ■

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Massachusetts, 1974.

[2] E.R. Berlekamp, *Algebraic Coding Theory,* Second Edition, Aegean Park Press, Laguna Hills, California, 1984.

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

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

[5] P. Elias, *Error-correcting codes for list decoding, IEEE Trans. Inform. Theory,* 37 (1991), 5–12.

[6] 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,* 37 (1991), 1274–1287.

[7] S. Gao, M.A. Shokrollahi, *Computing roots of polynomials over function fields of curves,* draft.

[8] V. Guruswami, M. Sudan, *Improved decoding of Reed-Solomon and algebraic-geometric codes, IEEE Trans. Inform. Theory,* to appear.

[9] M. Kaminski, D.G. Kirkpatrick, N.H. Bshouty, *Addition requirements for matrix and transposed matrix products, J. Algorithms,* 9 (1988), 354–364.

[10] R. Lidl, H. Niederreiter, *Finite Fields,* Addison-Wesley, Reading, Massachusetts, 1983.

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

[12] J.L. Massey, *Shift-register synthesis and BCH decoding, IEEE Trans. Inform. Theory,* IT-15 (1969), 122–127.

[13] R. Refslund Nielsen, T. Høholdt, *Decoding Reed-Solomon codes beyond half the minimum distance,* preprint.

[14] M.O. Rabin, *Probabilistic algorithms in finite fields, SIAM J. Comput.,* 9 (1980), 273–280.

[15] K. Saints, C. Heegard, *Algebraic-geometric codes and multidimensional cyclic codes: a unified theory and algorithms for decoding using Gröbner bases, IEEE Trans. Inform. Theory,* IT-41 (1995), 1733–1751.

[16] S. Sakata, *Finding a minimal set of linear recurring relations capable of generating a given finite two-dimensional array, J. Symb. Comput.,* 5 (1988), 321–337.

[17] M. Sudan, *Decoding of Reed-Solomon codes beyond the error-correction bound, J. Compl.,* 13 (1997), 180–193.

[18] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, *A method for solving key equation for decoding Goppa codes, Inform. Control,* 27 (1975), 87–99.

[19] R. Zippel, *Effective Polynomial Computation,* Kluwer, Boston, 1993.