

New List Decoding Algorithms for Reed-Solomon and BCH Codes

Yingquan Wu

Link_A_Media Devices Corp.

December 10, 2008

Abstract

In this paper we devise a rational curve fitting algorithm and apply it to the list decoding of Reed-Solomon and BCH codes. The resulting list decoding algorithms exhibit the following significant properties.

- The algorithm achieves the limit of list error correction capability (LECC) $n(1 - \sqrt{1 - D})$ for a (generalized) $(n, k, d = n - k + 1)$ Reed-Solomon code, which matches the Johnson bound, where $D \triangleq \frac{d}{n}$ denotes the normalized minimum distance. The algorithmic complexity is $O(n^6(1 - \sqrt{1 - D})^8)$. In comparison with the Guruswami-Sudan algorithm, which exhibits the same LECC, the proposed requires a multiplicity (which dictates the algorithmic complexity) significantly smaller than that of the Guruswami-Sudan algorithm in achieving a given LECC, except for codes with code-rate below 0.15. In particular, for medium-to-high rate codes, the proposed algorithm reduces the multiplicity by orders of magnitude. Moreover, for any $\epsilon > 0$, the intermediate LECC $t = \lfloor \epsilon \cdot \frac{d}{2} + (1 - \epsilon) \cdot (n - \sqrt{n(n - d)}) \rfloor$ can be achieved by the proposed algorithm with multiplicity $m = \lfloor \frac{1}{\epsilon} \rfloor$. Its list size is shown to be upper bounded by a constant with respect to a fixed normalized minimum distance D , rendering the algorithmic complexity quadratic in nature, $O(n^2)$.
- By utilizing the unique properties of the Berlekamp algorithm, the algorithm achieves the LECC limit $\frac{n}{2}(1 - \sqrt{1 - 2D})$ for a narrow-sense (n, k, d) binary BCH code, which matches the Johnson bound for binary codes. The algorithmic complexity is $O(n^6(1 - \sqrt{1 - 2D})^8)$. Moreover, for any $\epsilon > 0$, the intermediate LECC $t = \lfloor \epsilon \cdot \frac{d}{2} + (1 - \epsilon) \cdot \frac{n - \sqrt{n(n - 2d)}}{2} \rfloor$ can be achieved by the proposed algorithm with multiplicity $m = \lfloor \frac{1}{2\epsilon} \rfloor$. Its list size is shown to be upper bounded by a constant, rendering the algorithmic complexity quadratic in nature, $O(n^2)$.

Index Terms—List decoding, Berlekamp-Massey algorithm, Berlekamp algorithm, Reed-Solomon codes, BCH codes, Johnson bound, Rational curve-fitting algorithm.

I. INTRODUCTION

Reed-Solomon codes are the most commonly used error correction codes in practice. Their widespread applications include magnetic and optical data storage, wireline and wireless communications, and satellite communications. A Reed-Solomon code (n, k) over a finite field $\text{GF}(q)$ satisfies $n < q$ and achieves the maximally separable distance, i.e., $d = n - k + 1$. Its algebraic decoding has been extensively explored but remains a challenging research topic.

It is well-known that efficient algorithms exist to decode up to half the minimum distance with complexity $O(dn)$, namely, the Berlekamp-Massey algorithm [2, 19] and the Euclidean algorithm [26], which utilize the frequency spectrum property, and the Berlekamp-Welch algorithm, which utilizes the polynomial characteristics [27]. Koetter [15] devised a one-pass algorithm, building on the Berlekamp-Massey algorithm, to implement the generalized minimum distance (GMD) decoding, which otherwise requires $\lfloor \frac{d+1}{2} \rfloor$ rounds. Berlekamp [3] devised a one-pass algorithm, building on the Berlekamp-Welch algorithm, to implement $d + 1$ GMD decoding. Kamiya [14] presented one-pass GMD decoding algorithms and a one-pass Chase decoding algorithm for BCH codes utilizing the Berlekamp-Welch algorithm.

Sudan [25] discovered a polynomial-time algorithm, building on the Berlekamp-Welch algorithm, for (list) decoding Reed-Solomon codes beyond the classical correction capability $\lfloor \frac{d-1}{2} \rfloor$, however, the algorithm is effective only when the code rate $\frac{k}{n} < \frac{1}{3}$. Schmidt, Sidorenko, and Bossert [23] virtually extend one codeword to a sequence of interleaved codewords which yields a multiple-sequence linear shift register synthesis, and exploit the generalized Berlekamp-Massey algorithm, whose complexity is quadratic in nature, to correct errors beyond half the minimum distance. The algorithm succeeds if there is a unique solution within a certain capability, which is larger than the conventional error correction capability when the code rate is below $\frac{1}{3}$. Its error correction capability and rate threshold largely coincide with those of the Sudan algorithm in [25], whereas its algorithmic complexity is much lower than the Sudan algorithm. Guruswami and Sudan [9] devised an improved version of [25], which is capable of decoding beyond half the minimum distance over all rates. More specifically, the algorithm lists all codewords up to distance $n(1 - \sqrt{1 - D})$ (where D denotes the normalized minimum distance $D \triangleq \frac{d}{n}$) from the received word, while the algorithmic complexity is polynomial in nature. Its performance matches the Johnson bound [12], which gives a general lower bound on the number of errors one can correct using small lists in any code, as a function of the normalized minimum distance D . McEliece [20] characterized the average list size of the Guruswami-Sudan algorithm and showed that the list most likely contains only one codeword. Guruswami and Rudra [11] showed the optimality of the list error correction capability (LECC) $n(1 - \sqrt{1 - D})$ in the sense that the number of codewords lying slightly beyond the boundary can be superpolynomially large in code length n . Koetter and Vardy [16] showed a natural way to translate the soft-decision reliability information

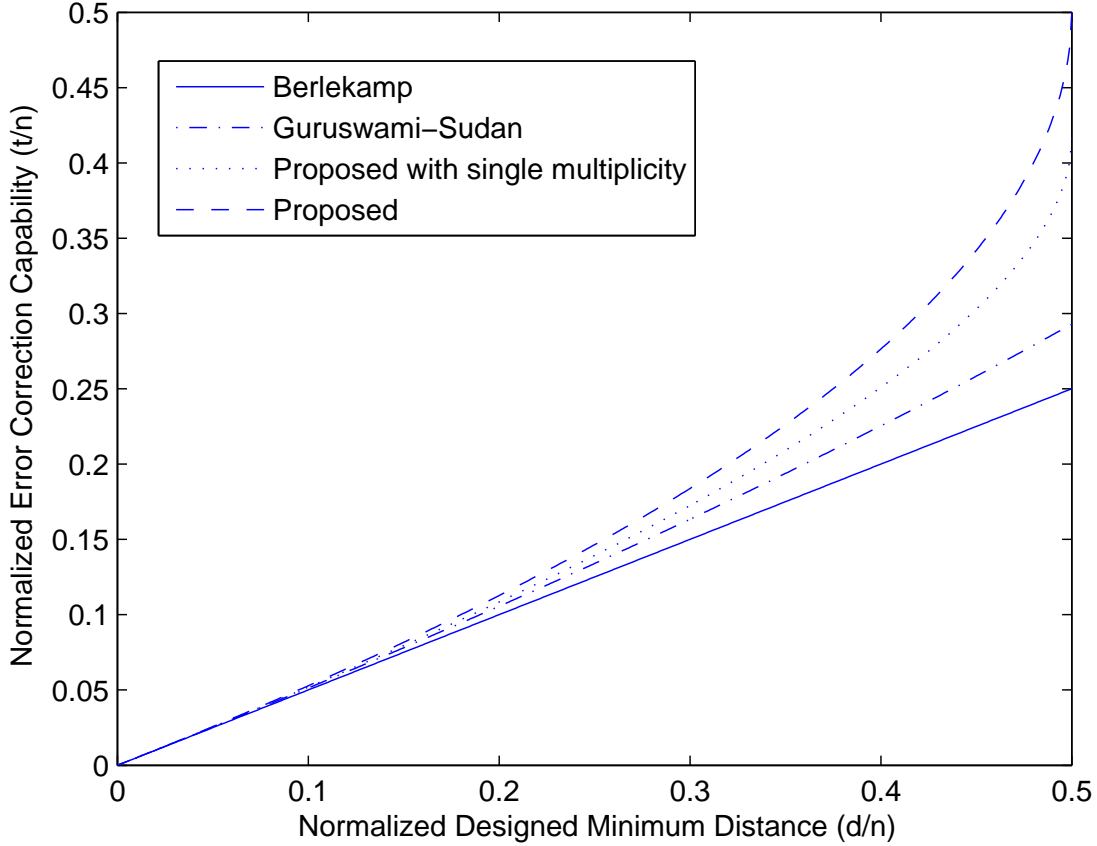


Figure 1: Normalized (list) error correction capability as a function of the normalized designed minimum distance for binary BCH codes.

provided by the channel into the multiplicity matrix which is directly involved in the Guruswami-Sudan algorithm. The resulting algorithm outperforms the Guruswami-Sudan algorithm.

In essence, the Guruswami-Sudan algorithm is a polynomial curve-fitting algorithm that determines all polynomials which passes through at least $\sqrt{n(n-d)}$ points out of n distinct points. Specifically, when given n distinct points $\{(x_i, y_i)\}_{i=0}^{n-1}$, where $[y_0, y_1, \dots, y_{n-1}]$ denotes a received word, a $(1, n-d)$ -weighted degree bivariate polynomial $Q(x, y)$ is constructed to pass through all n points, each with appropriate multiplicity, then $Q(x, y)$ contains all desired polynomials $p(x)$ as its factors in the form of $y - p(x)$. Finally all desired polynomials $p(x)$ are factorized iteratively [9]. The interpolation process can be expedited by utilizing the updating algorithm in [15] with quadratic complexity, whereas straightforward implementation using Gaussian elimination requires cubic complexity.

In this paper, we formulate the list decoding of Reed-Solomon codes as a rational curve-fitting problem utilizing the polynomials constructed by the Berlekamp-Massey algorithm. Specifically, let $\Lambda(x)$ and $B(x)$ be the error locator and correction polynomials, respectively, as obtained from the

Berlekamp-Massey algorithm. List decoding then finds pairs of polynomials $b(x)$ and $\lambda(x)$, such that each pair leads to a valid candidate error locator polynomial $\Lambda'(x) \triangleq \lambda(x) \cdot \Lambda(x) + b(x) \cdot xB(x)$, i.e., all the roots of $\Lambda'(x)$ are distinct and belong to the pre-defined space. We reduce this problem to a rational curve-fitting problem and subsequently present a novel polynomial algorithm comprised of rational interpolation and rational factorization. Using the up-to-date most efficient implementation algorithms [15,18], the proposed algorithm exhibits the complexity $O\left(n^2(\sqrt{n} - \sqrt{k})^8\right)$. The proposed list decoding algorithm exhibits the same LECC $n - \sqrt{n(n-d)}$ as the Guruswami-Sudan algorithm. However, the proposed algorithm requires much lower multiplicity, which dictates the algorithmic complexity, for almost the entire range of code rates. In particular, the proposed algorithm reduces the multiplicity by orders of magnitude for medium-to-high rate codes. Finally, the proposed algorithm utilizes the end results of the Berlekamp-Massey algorithm, in contrast to the decoding algorithms in [21, 23], which directly incorporate syndromes and achieve performance gains over conventional hard-decision decoding only when the code rate $\frac{k}{n} < \frac{1}{3}$. By observing that even iterations in the Berlekamp algorithm are automatically satisfied in decoding binary BCH codes, we present a modified version of the proposed algorithm that exhibits the LECC $\frac{n}{2}(1 - \sqrt{1 - 2D})$, which matches the Johnson bound for binary codes [12]. A comparison to existing state-of-the-art LECCs is depicted in Figure 1.

We also reveal a fundamental property of Reed-Solomon and binary BCH codes, namely, that there exist at most a constant number of codewords, regardless of code length n , with respect to an LECC under arbitrary small fraction away from the Johnson bound. Furthermore, we show that the corresponding Johnson bound can be arbitrarily approximated by the derivative algorithms with quadratic complexity, for both Reed-Solomon and binary BCH codes.

The remainder of the paper is organized as follows. Section II.A briefly introduces the Berlekamp-Massey algorithm for decoding Reed-Solomon codes, and then extends one iteration to correct up to $\lfloor \frac{d}{2} \rfloor$ errors with negligible additional complexity. Section II.B presents a re-formulated Berlekamp algorithm for decoding binary BCH codes, and then extends one iteration to correct up to $\lfloor \frac{d+1}{2} \rfloor$ errors with negligible additional complexity. The proposed list decoding algorithm for Reed-Solomon codes is detailed in Section III. The list decoding problem is formulated in Part A, the rational interpolation process is then described in Part B, followed by the rational factorization in Part C. The algorithmic description and performance assertion are presented in Part D and the computational complexity is characterized in Part E. Finally Part F shows that the LECC limit can be arbitrarily approximated with derivative algorithms with constant multiplicities which exhibit only quadratic complexity. Section IV presents an improved algorithm for decoding binary BCH codes. The paper is concluded with pertinent remarks in Section V.

II. ALGEBRAIC HARD-DECISION DECODING OF REED-SOLOMON AND BCH CODES

A. Berlekamp-Massey Algorithm and its One-Step Extension for Decoding Reed-Solomon Codes

For a (possibly shortened) Reed-Solomon $\mathcal{C}(n, k)$ code over $\text{GF}(q)$, a k -symbol $\mathbf{D} \triangleq [D_{k-1}, D_{k-2}, \dots, D_1, D_0]$ is encoded to an n -symbol codeword $\mathbf{C} \triangleq [C_{n-1}, C_{n-2}, \dots, C_1, C_0]$, or more conveniently, a dataword polynomial $D(x) = D_{k-1}x^{k-1} + D_{k-2}x^{k-2} + \dots + D_1x^1 + D_0$ is encoded to a codeword polynomial $C(x) = C_{n-1}x^{n-1} + C_{n-2}x^{n-2} + \dots + C_1x + C_0$, by means of a generator polynomial

$$G(x) \triangleq \prod_{i=0}^{n-k-1} (x - \alpha^{m_0+i})$$

where α is a primitive element of $\text{GF}(q)$ and m_0 is an arbitrary integer (in this presentation we do not distinguish between a vector $\mathbf{A} = [A_0, A_1, A_2, \dots, A_l]$ and its polynomial representation $A(x) = A_0 + A_1x + A_2x^2 + \dots + A_lx^l$). A polynomial of degree less than n is a codeword polynomial if and only if it is a multiple of the generator polynomial $G(x)$. As can be readily seen, a codeword polynomial $C(x)$ satisfies

$$C(\alpha^i) = 0, \quad i = m_0, m_0 + 1, m_0 + 2, \dots, m_0 + n - k - 1.$$

The minimum Hamming distance of the code is $d = n - k + 1$, an attribute known as *maximally-distance-separable* (cf. [4]).

Let $C(x)$ denote the transmitted codeword polynomial and $R(x)$ the received word polynomial. The decoding objective is to determine the error polynomial $E(x)$ such that $C(x) = R(x) - E(x)$.

In the following we introduce the Berlekamp-Massey algorithm, which provides a foundation for our list decoding algorithms. It begins with the task of error correction by computing syndrome values

$$S_i = R(\alpha^{i+m_0}) = C(\alpha^{i+m_0}) + E(\alpha^{i+m_0}) = E(\alpha^{i+m_0}), \quad i = 0, 1, 2, \dots, n - k - 1.$$

If all $n - k$ syndrome values are zero, then $R(x)$ is a codeword polynomial and thus is presumed that $C(x) = R(x)$, i.e., no errors have occurred. Otherwise, let e denote the (unknown) number of errors, $X_i \in \{\alpha^{-i}\}_{i=0}^{n-1}$, $i = 1, 2, \dots, e$, denote the error locations, and $Y_i \in \text{GF}(q)$, $i = 1, 2, \dots, e$, denote the corresponding error magnitudes.

Define the *syndrome* polynomial

$$S(x) \triangleq S_0 + S_1x + S_2x^2 + \dots + S_{n-k-1}x^{n-k-1}, \quad (1)$$

the *error locator* polynomial

$$\Lambda(x) \triangleq \prod_{i=1}^e (1 - X_i x) = 1 + \Lambda_1x + \Lambda_2x^2 + \dots + \Lambda_e x^e, \quad (2)$$

and the *error evaluator* polynomial

$$\Omega(x) \triangleq \sum_{i=1}^e Y_i X_i^{m_0} \prod_{j=1, j \neq i}^e (1 - X_j x) = \Omega_0 + \Omega_1 x + \Omega_2 x^2 + \dots + \Omega_{e-1} x^{e-1}. \quad (3)$$

The three polynomials satisfy the following *key equation* (cf. [4])

$$\Omega(x) = \Lambda(x)S(x) \pmod{x^{n-k}}. \quad (4)$$

The Berlekamp-Massey algorithm can be used to solve the above key equation, given that the number of errors e does not exceed the error-correction capability $\lfloor \frac{n-k}{2} \rfloor$ (cf. [2, 4]). Below we slightly re-formulate the Berlekamp-Massey algorithm given in [4], so as to facilitate the characterizations afterwards:

Berlekamp-Massey Algorithm

- Input: $\mathbf{S} = [S_0, S_1, S_2, \dots, S_{n-k-1}]$
- Initialization: $\Lambda^{(0)}(x) = 1$, $B^{(0)}(x) = 1$, and $L_\Lambda^{(0)} = 0$, $L_B^{(0)} = 0$
- For $r = 0, 1, 2, \dots, n - k - 1$, do:
 - Compute $\Delta^{(r+1)} = \sum_{i=0}^{L_\Lambda^{(r)}} \Lambda_i^{(r)} \cdot S_{r-i}$
 - Compute $\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x) - \Delta^{(r+1)} \cdot x B^{(r)}(x)$
 - If $\Delta^{(r+1)} \neq 0$ and $2L_\Lambda^{(r)} \leq r$, then
 - * Set $B^{(r+1)}(x) \leftarrow (\Delta^{(r+1)})^{-1} \cdot \Lambda^{(r)}(x)$
 - * Set $L_\Lambda^{(r+1)} \leftarrow L_\Lambda^{(r)} + 1$, $L_B^{(r+1)} \leftarrow L_\Lambda^{(r)}$
 - Else
 - * Set $B^{(r+1)}(x) \leftarrow x B^{(r)}(x)$
 - * Set $L_B^{(r+1)} \leftarrow L_B^{(r)} + 1$, $L_\Lambda^{(r+1)} \leftarrow L_\Lambda^{(r)}$
 - endif
- endfor
- Output: $\Lambda(x)$, $B(x)$, L_Λ , L_B

Note that in the above description, we used superscript “ (r) ” to stand for the r -th iteration and subscript “ i ” the i -th coefficient. L_Λ and L_B denote the length of linear feedback shift register (LFSR) described by $\Lambda(x)$ and $B(x)$, respectively. An LFSR of length L , $a_0 = 1$, a_1, a_2, \dots, a_L , is called to *generate* the sequence $s_0, s_1, s_2, \dots, s_r$ if

$$\sum_{j=0}^L s_{i-j} a_j = 0, \quad i = L, L+1, \dots, r. \quad (5)$$

The essence of the Berlekamp-Massey algorithm is to determine a minimum-length LFSR that *generates* the syndrome sequence $S_0, S_1, S_2, \dots, S_{n-k-1}$ [2, 19]. It is worth mentioning that there may exist multiple minimum-length LFSRs that generate the sequence $S_0, S_1, \dots, S_{n-k-1}$, and $\Lambda(x)$ obtained from the Berlekamp-Massey algorithm is one of them when non-unique. The error locator polynomial $\Lambda(x)$ and the correction polynomial $B(x)$ are characterized by the following lemma.

Lemma 1 *Let $\Lambda(x)$ be the error locator polynomial and $B(x)$ be the correction polynomial, computed by the Berlekamp-Massey algorithm.*

(i). L_Λ and L_B , the length of LFSR described by $\Lambda(x)$ and $B(x)$ respectively, satisfy

$$L_\Lambda + L_B = d - 1. \quad (6)$$

(ii). The degrees of $\Lambda(x)$ and $B(x)$ satisfy

$$\deg(\Lambda(x)) \leq L_\Lambda, \quad \deg(B(x)) \leq L_B. \quad (7)$$

When $\Lambda(x)$ is the true error locator polynomial as defined in (2), $\deg(\Lambda(x)) = L_\Lambda$.

(iii). The polynomials $\Lambda(x)$ and $B(x)$ are coprime, i.e., the two do not share a common factor.

Proof: (i). We show the following more general result

$$L_B^{(r)} + L_\Lambda^{(r)} = r. \quad (8)$$

It follows that in each iteration either $L_\Lambda^{(r+1)} \leftarrow L_\Lambda^{(r)} + 1$, $L_B^{(r+1)} \leftarrow L_B^{(r)}$, or $L_\Lambda^{(r+1)} \leftarrow L_\Lambda^{(r)}$, $L_B^{(r+1)} \leftarrow L_B^{(r)} + 1$, thus their sum increases by 1 in either case.

(ii). We show the first part by induction. When $i = 0$, we have $L_\Lambda^{(0)} = \deg(\Lambda^{(0)}(x)) = 0$ and $L_B^{(0)} = \deg(B^{(0)}(x)) = 0$. Assume that $\deg(\Lambda^{(r)}(x)) \leq L_\Lambda^{(r)}$ and $\deg(B^{(r)}(x)) \leq L_B^{(r)}$ hold for $i = r$. In the case of $\Delta^{(r+1)} \neq 0$ and $2L_\Lambda^{(r)} \leq r$, we have the following iteration

- $\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x) - \Delta^{(r+1)} \cdot xB^{(r)}(x)$
- $B^{(r+1)}(x) \leftarrow (\Delta^{(r+1)})^{-1} \cdot \Lambda^{(r)}(x)$
- $L_\Lambda^{(r+1)} \leftarrow L_\Lambda^{(r)} + 1$, $L_B^{(r+1)} \leftarrow L_B^{(r)}$.

(8), in conjunction with the condition $2L_\Lambda^{(r)} \leq r$, results in

$$L_B^{(r)} = r - L_\Lambda^{(r)} \geq L_\Lambda^{(r)}.$$

Therefore, we obtain

$$\begin{aligned} \deg(\Lambda^{(r+1)}(x)) &= \max\{\deg(\Lambda^{(r)}(x)), \deg(B^{(r)}(x)) + 1\} \leq \max\{L_\Lambda^{(r)}, L_B^{(r)} + 1\} = L_B^{(r)} + 1 = L_\Lambda^{(r+1)}. \\ \deg(B^{(r+1)}(x)) &= \deg(\Lambda^{(r)}(x)) \leq L_\Lambda^{(r)} = L_B^{(r+1)}. \end{aligned}$$

When $\Delta^{(r+1)} = 0$, we have the following update

- $B^{(r+1)}(x) \leftarrow xB^{(r)}(x), \quad L_B^{(r+1)} \leftarrow L_B^{(r)} + 1.$

The conclusion naturally holds. Finally, when $\Delta^{(r+1)} \neq 0$ and $2L_\Lambda^{(r)} > r$, the algorithmic updates follow

- $\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x) - \Delta^{(r+1)} \cdot xB^{(r)}(x)$
- $B^{(r+1)}(x) \leftarrow xB^{(r)}(x), \quad L_B^{(r+1)} \leftarrow L_B^{(r)} + 1.$

(8), in conjunction with the condition $2L_\Lambda^{(r)} > r$, results in

$$L_B^{(r)} = r - L_\Lambda^{(r)} < L_\Lambda^{(r)}.$$

Therefore, we obtain

$$\begin{aligned} \deg(\Lambda^{(r+1)}(x)) &= \max\{\deg(\Lambda^{(r)}(x)), \deg(B^{(r)}(x)) + 1\} \leq \max\{L_\Lambda^{(r)}, L_B^{(r)} + 1\} \leq L_\Lambda^{(r)} = L_\Lambda^{(r+1)} \\ \deg(B^{(r+1)}(x)) &= \deg(B^{(r)}(x)) + 1 \leq L_B^{(r)} + 1 = L_B^{(r+1)}. \end{aligned}$$

We thus have justified the first part of (ii). The second part naturally follows (3), (4) and the definition of generating a sequence in (5).

Part (iii) can be shown through contradiction (cf. [15]). Herein we give an inductive proof. Evidently, when $i = 0$, $\Lambda^{(0)}(x) = 1$ and $B^{(0)}(x) = 1$ are coprime. Assume that $\Lambda^{(r)}(x)$ and $B^{(r)}(x)$ are coprime for $i = r$. For $i = r + 1$, if $\Delta^{(r+1)} \neq 0$ and $2L_\Lambda^{(r)} \leq r$, then the iteration, $\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x) - \Delta^{(r+1)} \cdot xB^{(r)}(x)$ and $B^{(r+1)}(x) \leftarrow (\Delta^{(r+1)})^{-1} \cdot \Lambda^{(r)}(x)$, clearly indicates that $\Lambda^{(r+1)}(x)$ and $B^{(r+1)}(x)$ are coprime, conditioned on that $\Lambda^{(r)}(x)$ and $B^{(r)}(x)$ are coprime; so is the alternative iteration, $\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x) - \Delta^{(r+1)} \cdot xB^{(r)}(x)$ and $B^{(r+1)}(x) \leftarrow xB^{(r)}(x)$. We conclude that $\Lambda(x)$ and $B(x)$ are coprime. $\square\square$

Note the initial cause of $\deg(\Lambda(x)) < L_\Lambda$ is due to the special condition [2]

$$L_\Lambda^{(r)} = L_B^{(r)} + 1 \quad \text{and} \quad \Lambda_l^{(r)} = \Delta^{(r+1)} B_{l-1}^{(r)}, \quad \text{where } l = L_\Lambda^{(r)}.$$

Let $n - k$ be an odd number. Then, the number of errors up to

$$t_0 \triangleq \frac{n - k + 1}{2} = \frac{d}{2} \tag{9}$$

can be corrected by the following simple list decoding algorithm

One-Step-Ahead Berlekamp-Massey Algorithm

1. If $L_\Lambda > t_0$, then declare a decoding failure.
2. If $L_\Lambda < t_0$, then determine all distinct roots in $\{\alpha^{-i}\}_{i=0}^{n-1}$. If the number of (distinct) roots is equal to L_Λ , then apply Forney's formula and return the unique codeword, otherwise declare a decoding failure (which is identical to the normal Berlekamp-Massey algorithm).

3. Evaluate $\Delta_i = \frac{\Lambda(\alpha^{-i})}{\alpha^{-i}B(\alpha^{-i})}$, $i = 0, 1, 2, \dots, n-1$.
4. Group the index sets $\{i_1, i_2, \dots, i_{t_0}\}$ such that Δ_i 's are identical (each set corresponds to the roots of a valid error locator polynomial).
5. Apply Forney's formula to compute error magnitudes with respect to each index set, each resulting in a candidate codeword.

Proof of correctness: We note that $\Lambda(x)$ and $B(x)$ are obtained at the $(n-k)$ -th iteration of the Berlekamp-Massey algorithm. Following the nature of the Berlekamp-Massey algorithm, the additional syndrome S_{n-k} determines all valid error locator polynomial $\Lambda^*(x)$ of degree up to t_0 . More specifically, a valid error locator polynomial $\Lambda^*(x)$ of degree up to t_0 satisfies the form

$$\Lambda^*(x) = \Lambda(x) - \Delta(S_{n-k}) \cdot xB(x) \quad (10)$$

where the discrepancy $\Delta(S_{n-k})$ is a linear function of S_{n-k} ,

$$\Delta(S_{n-k}) = \sum_{i=0}^{L_\Lambda} S_{n-k-i} \Lambda_i.$$

We observe that $L_\Lambda + L_{xB} = n - k + 1$. We easily see that $\Lambda^*(x)$ has degree up to t_0 if and only if $L_{\Lambda^*} = L_\Lambda$, in particular, $\Lambda^*(x) = \Lambda(x)$ if $L_\Lambda < t_0$. This justifies Steps 1 and 2.

Now assume $L_\Lambda = t_0$ and $\Delta_{i_1} = \Delta_{i_2} = \dots = \Delta_{i_{t_0}}$. By letting $\Delta = \Delta_{i_1}$, $\Lambda^*(x)$ has degree t_0 , and at same time, contains t_0 valid roots, $\alpha^{-i_1}, \alpha^{-i_2}, \dots, \alpha^{-i_{t_0}}$, i.e., $\Lambda^*(x)$ is a valid error locator polynomial. On the other hand, if an error locator polynomial $\bar{\Lambda}(x)$ is of degree t_0 and contains t_0 valid roots, $\alpha^{-i_1}, \alpha^{-i_2}, \dots, \alpha^{-i_{t_0}}$. Then, following (10), we have

$$0 = \Lambda(\alpha^{-i_l}) - \Delta(S_{n-k}) \cdot \alpha^{-i_l} B(\alpha^{-i_l}), \quad l = 1, 2, \dots, t_0,$$

indicating $\Delta_{i_1} = \Delta_{i_2} = \dots = \Delta_{i_{t_0}} = \Delta(S_{n-k})$. We thus justify Steps 3, 4, and 5. \square

Remarks: Compared to the approach in [4], where the syndrome S_{n-k} is exhaustively searched throughout the field $\text{GF}(q)$, each time $\Lambda^*(x)$ is produced and examined, the proposed algorithm reduces the computational complexity by a factor of q . Essentially, the proposed algorithm extends one iteration beyond the conventional Berlekamp-Massey algorithm while maintaining the original computational complexity. Further, note that an index (location) can only be classified to one group, thus any two sets of error locator roots are disjoint. As a result, there exist at most $\lfloor \frac{n}{t_0} \rfloor$ distinct codewords at distance t_0 from a received word. An extension of the above method is the $\lfloor \frac{d+1}{2} \rfloor$ decoding algorithm which utilizes the Chien Search to determine the subsequent two unknown discrepancies [6].

Remark: The foregoing one-step-ahead algorithm is essentially a degeneration of the list decoding algorithm to be presented in next section.

B. Berlekamp Algorithm and its One-Step Extension for Decoding BCH Codes

The underlying generator polynomial of a BCH code contains consecutive roots $\alpha, \alpha^2, \dots, \alpha^{2t}$. Note for an underlying binary BCH code, the designed minimum distance d is always odd, which is actually a lower bound of the true minimum distance.

The Berlekamp algorithm is a simplified version of the Berlekamp-Massey algorithm for decoding binary BCH codes by incorporating the special syndrome property

$$S_{2i+1} = S_i^2, \quad i = 0, 1, 2, \dots$$

which yields zero discrepancies at even iterations of the Berlekamp-Massey algorithm (cf. [2]). Below we re-formulate slightly the Berlekamp algorithm described in [2], so as to facilitate the characterizations thereafter.

Berlekamp Algorithm

- Input: $\mathbf{S} = [S_0, S_1, S_2, \dots, S_{d-2}]$
- Initialization: $\Lambda^{(0)}(x) = 1, B^{(-1)}(x) = x^{-1}, L_\Lambda^{(0)} = 0, L_B^{(-1)} = -1$
- For $r = 0, 2, \dots, d-3$, do:
 - Compute $\Delta^{(r+2)} = \sum_{i=0}^{L_\Lambda^{(r)}} \Lambda_i^{(r)} \cdot S_{r-i}$
 - Compute $\Lambda^{(r+2)}(x) = \Lambda^{(r)}(x) - \Delta^{(r+2)} \cdot x^2 B^{(r-1)}(x)$
 - If $\Delta^{(r+2)} \neq 0$ and $2L_\Lambda^{(r)} \leq r$, then
 - * Set $B^{(r+1)}(x) \leftarrow (\Delta^{(r+2)})^{-1} \cdot \Lambda^{(r)}(x)$
 - * Set $L_\Lambda^{(r+2)} \leftarrow L_B^{(r-1)} + 2, L_B^{(r+1)} \leftarrow L_\Lambda^{(r)}$
 - Else
 - * Set $B^{(r+1)}(x) \leftarrow x^2 B^{(r-1)}(x)$
 - * Set $L_B^{(r+1)} \leftarrow L_B^{(r-1)} + 2, L_\Lambda^{(r+2)} \leftarrow L_\Lambda^{(r)}$
- endif
- endfor
- Output: $\Lambda(x), B(x), L_\Lambda, L_B$

The following lemma characterizes the error locator polynomial $\Lambda(x)$ and the correction polynomial $B(x)$ produced by the Berlekamp algorithm.

Lemma 2 *Let $\Lambda(x)$ and $B(x)$ be the error locator and correction polynomials, respectively, computed by the Berlekamp algorithm.*

(i). L_Λ and L_B , the length of AFSR described by $\Lambda(x)$ and $B(x)$ respectively, satisfy

$$L_\Lambda + L_B = d - 2. \tag{11}$$

(ii). The degrees of $\Lambda(x)$ and $B(x)$ satisfy

$$\deg(\Lambda(x)) \leq L_\Lambda, \quad \deg(B(x)) \leq L_B. \quad (12)$$

When $\Lambda(x)$ is the true error locator polynomial as defined in (2), $\deg(\Lambda(x)) = L_\Lambda$.

(iii). The polynomials $\Lambda(x)$ and $B(x)$ are coprime, i.e., the two do not share a common factor.

Similarly, we have the following one-step-ahead algorithm that corrects (in the list decoding sense) up to $\frac{d+1}{2}$ errors at essentially same complexity as the original Berlekamp algorithm. The proof is straightforward.

One-Step-Ahead Berlekamp Algorithm

1. If $L_\Lambda > \frac{d+1}{2}$, then declare a decoding failure.
2. If $L_\Lambda < \frac{d+1}{2}$, then determine all distinct roots in $\{\alpha^{-i}\}_{i=0}^{n-1}$. If the number of (distinct) roots is equal to L_Λ , then return the corresponding unique codeword, otherwise declare a decoding failure (which is identical to the normal Berlekamp algorithm)
3. Evaluate $\Delta_i = \frac{\Lambda(\alpha^{-i})}{\alpha^{-2i}B(\alpha^{-i})}$, $i = 0, 1, 2, \dots, n-1$.
4. Group the index sets $\{i_1, i_2, \dots, i_{L_\Lambda}\}$ such that Δ_i 's are identical (each set corresponds to the roots of a valid error locator polynomial).
5. Flip bits on all indices (locations) of each set obtained in Step 4, each resulting in a candidate codeword.

Remark: The proposed algorithm is superior to the “trick” presented in [5] in which $\frac{d+1}{2}$ error correction capability is achieved only for even-weight subcode by exploiting the parity syndrome of a received word.

III. LIST DECODING ALGORITHM FOR REED-SOLOMON CODES

In this section we present a list decoding algorithm for Reed-Solomon codes that corrects up to $\lceil n-1-\sqrt{n(n-d)} \rceil$ errors, which is identical to that of the Guruswami-Sudan algorithm in [9]. We shall extend the notation $t_0 = \frac{d}{2}$ to allow d to take any integer value, instead of even value as initially defined in (9). We use the terminology “valid” root to indicate a root is in the pre-defined space which, in this context, means $\{1, \alpha^{-1}, \dots, \alpha^{-(n-1)}\}$. We also define a companion polynomial $\bar{\mathcal{Q}}(x, y)$ of a bivariate polynomial $\mathcal{Q}(x, y)$ to be $\bar{\mathcal{Q}}(x, y) \triangleq \mathcal{Q}(x, 1/y)y^{P_y}$ (herein P_y denotes the power of y in $\mathcal{Q}(x, y)$), and $[y^l]\mathcal{Q}(x, y)$ the polynomial of x associated with the term y^l . Specifically,

$$\bar{\mathcal{Q}}(x, y) \triangleq f_{P_y}(x) + f_{P_y-1}(x)y + f_{P_y-2}(x)y^2 + \dots + f_1(x)y^{P_y-1} + f_0(x)y^{P_y} \quad (13)$$

$$[y^l]\mathcal{Q}(x, y) \triangleq f_l(x) \quad (14)$$

if $\mathcal{Q}(x, y)$ is in form of $\mathcal{Q}(x, y) = f_0(x) + f_1(x)y + f_2(x)y^2 + \dots + f_{P_y}(x)y^{P_y}$.

A. Problem Formulation

Lemma 3 *Let $\Lambda^*(x)$ be the true error locator polynomial as defined in (2). Let $\Lambda(x)$ and $B(x)$ be the error locator and correction polynomials, respectively, obtained from the Berlekamp-Massey algorithm. Then, $\Lambda^*(x)$ exhibits the form of*

$$\Lambda^*(x) = \Lambda(x) \cdot \lambda^*(x) + xB(x) \cdot b^*(x), \quad (15)$$

where the polynomials $\lambda^*(x)$ and $b^*(x)$ exhibit the following properties

- (i). $\lambda_0^* = 1$;
- (ii). if $b^*(x) = 0$, then $\lambda^*(x) = 1$;
- (iii). $\lambda^*(x)$ and $b^*(x)$ are coprime;
- (iv). $\deg(\lambda^*(x)) = \deg(\Lambda^*(x)) - L_\Lambda \quad \& \quad \deg(b^*(x)) \leq \deg(\Lambda^*(x)) - L_{xB}$, or $\deg(\lambda^*(x)) \leq \deg(\Lambda^*(x)) - L_\Lambda \quad \& \quad \deg(b^*(x)) = \deg(\Lambda^*(x)) - L_{xB}$;
- (v). if $\deg(\Lambda^*(x)) < n - k$, then $\lambda^*(x)$ and $b^*(x)$ are unique.

Proof: When the number of errors $e \leq \lfloor \frac{n-k}{2} \rfloor$, the above conclusions trivially hold, following the nature of the Berlekamp-Massey algorithm. In the following we consider for the case $e > \lfloor \frac{n-k}{2} \rfloor$. Suppose a genie tells additional syndromes $S_{n-k}, S_{n-k+1}, \dots, S_{2e-1}$, (alternative interpretation is to assume the all-zero codeword is transmitted and thus the additional syndromes are available), the true error locator polynomial $\Lambda^*(x)$ can be obtained by further applying the Berlekamp-Massey algorithm in conjunction with syndromes $S_{n-k}, S_{n-k+1}, \dots, S_{2e-1}$. Thus, (15) holds by the nature of the Berlekamp-Massey algorithm.

(i). Note that $\Lambda_0^*(x) = 1$ by definition (2) and $\Lambda_0(x) = 1$ by the nature of the Berlekamp-Massey iteration, $\Lambda^{(r+1)}(x) = \Lambda^{(r)}(x) - \Delta^{(r+1)}xB^{(r)}(x)$, and by the initial condition $\Lambda^{(0)}(x) = 1$. On the other hand, the constant term of $xB(x) \cdot b^*(x)$ is always zero. Therefore, $\lambda_0^*(x) = 1$.

(ii). If $b^*(x) = 0$, then the corresponding (additional) discrepancies $\Delta^{(n-k)}, \Delta^{(n-k+1)}, \dots, \Delta^{(2e)}$, are all zeros. We thus obtain

$$\Lambda^*(x) = \Lambda^{(2e)}(x) = \Lambda^{(2e-1)}(x) = \dots = \Lambda^{(n-k)}(x) = \Lambda(x),$$

which justifies the property (ii).

(iii). Let $B^*(x)$ be the correction polynomial associated with $\Lambda^*(x)$. It can be easily shown that (cf. [4])

$$\begin{bmatrix} \Lambda^*(x) \\ B^*(x) \end{bmatrix} = \prod_{r=1}^{2e} \begin{bmatrix} 1 & -\Delta^{(r)}x \\ (\Delta^{(r)})^{-1}\delta^{(r)} & (1 - \delta^{(r)})x \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

where $\delta^{(r)}$ denotes a binary value associated with selection of $B^{(r)}(x)$ and is zero when $\Delta^{(r)} = 0$, and likewise,

$$\begin{bmatrix} \Lambda(x) \\ B(x) \end{bmatrix} = \prod_{r=1}^{n-k} \begin{bmatrix} 1 & -\Delta^{(r)}x \\ (\Delta^{(r)})^{-1}\delta^{(r)} & (1 - \delta^{(r)})x \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

The above equalities, in conjunction with (15), indicate that

$$\begin{bmatrix} \lambda^*(x) & xb^*(x) \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix} \cdot \prod_{r=n-k+1}^{2e} \begin{bmatrix} 1 & -\Delta^{(r)}x \\ (\Delta^{(r)})^{-1}\delta^{(r)} & (1 - \delta^{(r)})x \end{bmatrix}.$$

We proceed to show by induction that $\lambda^*(x)$ and $xb^*(x)$ are coprime. When $r = n - k + 1$, we have $\lambda^{(1)}(x) = 1$ and $xb^{(1)}(x) = -\Delta^{(n-k+1)}x$. Clearly $\lambda^{(1)}(x)$ and $xb^{(1)}(x)$ are coprime. Assuming that $\lambda^{(i)}(x)$ and $xb^{(i)}(x)$ (i.e., the case $r = n - k + i$) are coprime, we then have

$$\begin{aligned} \begin{bmatrix} \lambda^{(i+1)}(x) & xb^{(i+1)}(x) \end{bmatrix} &= \begin{bmatrix} \lambda^{(i)}(x) & xb^{(i)}(x) \end{bmatrix} \cdot \begin{bmatrix} 1 & -\Delta^{(r+1)}x \\ (\Delta^{(r+1)})^{-1}\delta^{(r+1)} & (1 - \delta^{(r+1)})x \end{bmatrix} \\ &= \begin{bmatrix} \lambda^{(i)}(x) + (\Delta^{(r+1)})^{-1}\delta^{(r+1)}xb^{(i)}(x) & -\Delta^{(r+1)}x\lambda^{(i)}(x) + (1 - \delta^{(r+1)})x^2b^{(i)}(x) \end{bmatrix}. \end{aligned}$$

When $\delta^{(r+1)} = 0$, we obtain

$$\begin{bmatrix} \lambda^{(i+1)}(x) & xb^{(i+1)}(x) \end{bmatrix} = \begin{bmatrix} \lambda^{(i)}(x) & -\Delta^{(r+1)}x\lambda^{(i)}(x) + x^2b^{(i)}(x) \end{bmatrix}$$

which clearly indicates that $\lambda^{(i+1)}(x)$ and $xb^{(i+1)}(x)$ are coprime. When $\delta^{(r+1)} = 1$, we obtain

$$\begin{bmatrix} \lambda^{(i+1)}(x) & xb^{(i+1)}(x) \end{bmatrix} = \begin{bmatrix} \lambda^{(i)}(x) + (\Delta^{(r+1)})^{-1}xb^{(i)}(x) & -\Delta^{(r+1)}x\lambda^{(i)}(x) \end{bmatrix}$$

which again indicates that $\lambda^{(i+1)}(x)$ and $xb^{(i+1)}(x)$ are coprime. Therefore, $\lambda^*(x)$ is coprime to $xb^*(x)$ and subsequently to $b^*(x)$.

(iv). The results clearly hold when $\Lambda^*(x) = \Lambda(x)$. We next show for the case $\Lambda^*(x) \neq \Lambda(x)$, which indicates that the number of errors $e > \frac{n-k}{2}$. Given the hypothetical $2e - (n - k)$ additional syndromes, $S_{n-k}, S_{n-k+1}, \dots, S_{2e-1}$, $\Lambda^*(x)$ is the minimum-length LFSR to generate the syndrome sequence $S_0, S_1, S_2, \dots, S_{2e-1}$. By Lemma 1, $L_{\Lambda^*} = e = \deg(\Lambda^*(x))$. On the other hand, we observe that $\lambda^*(x)$ and $b^*(x)$ are obtained by further applying the Berlekamp-Massey iterations on top of $\Lambda(x) = \Lambda^{(n-k)}(x)$ and $B(x) = B^{(n-k)}(x)$, whose LFSR lengths are L_{Λ} and L_B , respectively. Therefore, we have

$$\begin{aligned} \deg(\lambda^*(x)) &\leq L_{\Lambda^*} - L_{\Lambda} = \deg(\Lambda^*(x)) - L_{\Lambda} \\ \deg(xb^*(x)) &\leq L_{\Lambda^*} - L_B = \deg(\Lambda^*(x)) - L_B. \end{aligned}$$

Further note that (15) indicates that

$$\deg(\Lambda(x)) + \deg(\lambda^*(x)) = \deg(\Lambda^*(x)) \quad \text{or} \quad \deg(xB(x)) + \deg(b^*(x)) = \deg(\Lambda^*(x)).$$

Without loss of generality, we assume $\deg(\Lambda(x)) + \deg(\lambda^*(x)) = \deg(\Lambda^*(x))$, which immediately yields

$$\deg(\lambda^*(x)) = \deg(\Lambda^*(x)) - \deg(\Lambda(x)) \geq \deg(\Lambda^*(x)) - L_\Lambda,$$

where the inequality follows Lemma 1. We thus obtain

$$\deg(\lambda^*(x)) = \deg(\Lambda^*(x)) - L_\Lambda \quad \text{and} \quad \deg(\Lambda(x)) = L_\Lambda.$$

(v). We prove Part (v) by contradiction. Assume that there is another pair $\lambda(x)$ and $b(x)$ satisfying (15). Then, we have

$$\Lambda^*(x) = \Lambda(x) \cdot \lambda^*(x) + xB(x) \cdot b^*(x) = \Lambda(x) \cdot \lambda(x) + xB(x) \cdot b(x),$$

which immediately indicates

$$\Lambda(x) \cdot (\lambda^*(x) - \lambda(x)) = xB(x) \cdot (b(x) - b^*(x)).$$

Since $\Lambda(x)$ is coprime to $xB(x)$ (by Lemma 1.(iii)), $\Lambda(x)$ divides $b(x) - b^*(x)$. Likewise, $xB(x)$ divides $\lambda^*(x) - \lambda(x)$. (15) indicates that

$$\deg(\Lambda^*(x)) = \max\{\deg(\lambda^*(x)) + \deg(\Lambda(x)), \deg(b^*(x)) + \deg(xB(x))\}$$

Without loss of generality, we assume $\deg(\Lambda^*(x)) = \deg(\lambda^*(x)) + \deg(\Lambda(x))$. Then, $\deg(\Lambda(x)) = L_\Lambda$, following Part (iv). Subsequently,

$$\deg(\Lambda(x)) = n - k - L_B > e - L_B \geq \deg(b^*(x)).$$

Likewise, $\deg(\Lambda(x)) > \deg(b(x))$. Therefore, $\Lambda(x)$ divides $b(x) - b^*(x)$ if and only if $b(x) - b^*(x) = 0$, i.e., $b(x) = b^*(x)$. Part (v) is thus justified. \square

Example 1. Consider a transmitted codeword pertaining to the (15, 5) Reed-Solomon code

$$\mathbf{c} = [0, \alpha^{12}, \alpha^{10}, \alpha^{11}, \alpha^7, \alpha^5, \alpha^{11}, \alpha^{11}, \alpha^6, \alpha^8, \alpha^{14}, \alpha^{11}, \alpha^6, \alpha^2, \alpha]$$

where α denotes a primitive element of $\text{GF}(16)$, and the corresponding received word which has 7 errors

$$\mathbf{r} = [0, \alpha^6, \alpha^{10}, \alpha^5, \alpha^7, \alpha^5, \alpha^5, \alpha^{11}, \alpha, \alpha^2, \alpha^{14}, \alpha^{12}, \alpha^6, \alpha^7, \alpha].$$

The true error locator polynomial is

$$\begin{aligned} \Lambda^*(x) &= (1 - \alpha x)(1 - \alpha^3 x)(1 - \alpha^6 x)(1 - \alpha^8 x)(1 - \alpha^9 x)(1 - \alpha^{11} x)(1 - \alpha^{13} x) \\ &= 1 + \alpha^9 x + \alpha^{11} x^3 + \alpha^5 x^4 + \alpha^{12} x^6 + \alpha^6 x^7. \end{aligned}$$

Applying the Berlekamp-Massey algorithm, we obtain the following error locator and correction polynomials

$$\begin{aligned}\Lambda(x) &= 1 + \alpha^7 x + \alpha^{13} x^2 + \alpha x^3 + \alpha^{13} x^4 + \alpha^4 x^5 \\ B(x) &= \alpha^6 x + \alpha^5 x^2 + \alpha^{10} x^3 + \alpha^3 x^4 + \alpha^{11} x^5.\end{aligned}$$

It can be verified that $\Lambda^*(x)$ satisfies the following decomposition

$$\Lambda^*(x) = \Lambda(x)(1 + \alpha x + \alpha^2 x^2) + xB(x) \cdot \alpha^8.$$

□□

Lemma 4 *Let $\Lambda(x)$ and $B(x)$ be the error locator and correction polynomials, respectively, obtained from the Berlekamp-Massey algorithm. Let the distance threshold $t \geq t_0$, where t_0 is defined in (9).*

- (i). *If the degree of $\Lambda(x)$, $L_\Lambda > t$, then there is no codeword within distance t from the received word.*
- (ii). *If the degree of $xB(x)$, $L_{xB} > t$, then only $\Lambda^*(x) = \Lambda(x)$ may result in a codeword within distance t from the received word.*

Proof: We show (i) by contradiction. Assume there is a codeword within distance t from the received word. Then, the corresponding error locator polynomial $\Lambda^*(x)$, which has degree up to t , generates the syndrome sequence $S_0, S_1, \dots, S_{n-k-1}$. This contradicts the fact that $\Lambda(x)$ represents a minimum-length shift register which generates $S_0, S_1, \dots, S_{n-k-1}$.

(ii). If $\Lambda(x)$ contains exactly L_Λ distinct roots within $\{\alpha^{-i}\}_{i=0}^{n-1}$, then $\Lambda^*(x) = \Lambda(x)$ leads to a codeword with distance $L_\Lambda < t$ from the received word. Assume there is a codeword at distance e from the received word, where $e > L_\Lambda$. Then, $e \geq t_0$. This is because, if $e < t_0$, then $\Lambda^*(x)$ is the unique minimum-length LFSR to generate the syndrome sequence, $S_0, S_1, \dots, S_{n-k-1}$, which obviously conflicts the facts $\Lambda^*(x) \neq \Lambda(x)$. Now assume that the additional syndromes $S_{n-k}, S_{n-k+1}, \dots, S_{2e-1}$ are available and used for further applying the Berlekamp-Massey iterations. Let $\Delta^{(r)}$ ($r > n - k$) be the first nonzero discrepancy, then the error locator polynomial is updated as

$$\Lambda^{(r)}(x) = \Lambda^{(n-k)}(x) - \Delta^{(r)} x^{r-(n-k)} B^{(n-k)}(x),$$

which immediately indicates

$$L_\Lambda^* \geq L_\Lambda^{(r)} = L_B + r - (n - k) \geq L_{xB}.$$

Since $\Lambda^*(x)$ is a true error locator polynomial, $e = L_{\Lambda^*} = \deg(\Lambda^*(x))$. Thus, $\Lambda^*(x)$ has degree at least L_{xB} , i.e., $e > L_{xB}$ and the proof is completed. □□

Example 2. Consider a transmitted codeword pertaining to the (15, 5) Reed-Solomon code

$$\mathbf{c} = [\alpha^2, \alpha^3, \alpha^5, \alpha^9, \alpha^{12}, \alpha^5, \alpha^4, \alpha, \alpha^9, \alpha^{14}, \alpha^9, \alpha^8, \alpha^9, \alpha^{11}, \alpha^{10}]$$

and the corresponding received word

$$\mathbf{r} = [\alpha^2, \alpha^8, \alpha^5, \alpha^9, 1, \alpha^5, \alpha^4, \alpha^{12}, \alpha^2, \alpha^{14}, \alpha^9, \alpha^8, \alpha^{13}, 1, \alpha^{11}]$$

which has 7 errors. The true error locator polynomial is

$$\begin{aligned}\Lambda^*(x) &= (1 - \alpha x)(1 - \alpha^4 x)(1 - \alpha^7 x)(1 - \alpha^8 x)(1 - \alpha^{12} x)(1 - \alpha^{13} x)(1 - \alpha^{14} x) \\ &= 1 + \alpha^5 x + \alpha^4 x^3 + \alpha^{14} x^4 + \alpha^8 x^5 + \alpha^{12} x^6 + \alpha^{14} x^7.\end{aligned}$$

The Berlekamp-Massey algorithm returns the error locator and correction polynomials below

$$\begin{aligned}\Lambda(x) &= 1 + \alpha^9 x + \alpha^5 x^2 + \alpha^6 x^3 + \alpha^9 x^5 + \alpha x^6 + \alpha^4 x^7 \\ B(x) &= \alpha^{11} + \alpha^5 x + \alpha x^2 + \alpha^2 x^3.\end{aligned}$$

Note that $\Lambda(x)$ has degree 7, Lemma 4 asserts no codewords within 6 symbol difference from the received word. $\square\square$

B. Rational Interpolation

Dividing both sides of (15) by $xB(x)$, we obtain

$$\frac{\Lambda^*(x)}{xB(x)} = \frac{\Lambda(x)}{xB(x)} \cdot \lambda^*(x) + b^*(x).$$

Define

$$y_i = -\frac{\Lambda(\alpha^{-i})}{\alpha^{-i}B(\alpha^{-i})}, \quad i = 0, 1, 2, \dots, n-1 \quad (16)$$

where y_i is set to ∞ when $B(\alpha^{-i}) = 0$, whose implication will be explored shortly. Let $\alpha^{-i_1}, \alpha^{-i_2}, \dots, \alpha^{-i_e}$, be all the valid roots of the true error locator polynomial $\Lambda^*(x)$. Then, $y \cdot \lambda^*(x) - b^*(x)$ passes precisely through e points, $(\alpha^{-i_1}, y_{i_1}), (\alpha^{-i_2}, y_{i_2}), \dots, (\alpha^{-i_e}, y_{i_e})$.

Given the set of n distinct points $\{(\alpha^{-i}, y_i)\}_{i=0}^{n-1}$, we are interested in finding rational functions $y(x)$ which pass t ($t \geq t_0$) points, in the sense that $y(\alpha^{-i}) = y_i$. If $y_i = \infty$, then $y(x)$ must contain the pole α^{-i} . This is because, when $B(\alpha^{-i}) = 0$, $\lambda(\alpha^{-i})$ must be zero, due to the fact that $B(x)$ and $\Lambda(x)$ are coprime and thus cannot share the root. This is essentially a rational curve-fitting problem. In [9], a powerful approach which makes use of multiple interpolation was presented to solve the polynomial curve-fitting problem. In essence, it constructs a global bivariate polynomial (curve) $\mathcal{Q}(x, y)$ that passes through all points with certain multiplicity. By its algebraic nature all desired polynomials of the form $y - p(x)$ are its factors [9]. In the following we generalize the approach to the rational domain.

The most efficient known interpolation technique was presented in [18], whose prototype was proposed in [15]. This approach exhibits quadratic complexity, as opposed to the straightforward

Gaussian elimination method which exhibits cubic complexity. We show that the same approach can also be applied to the rational interpolation with appropriate modifications. Firstly, the weight of y is determined differently. Note that we are essentially interested in the form of $y \cdot \lambda(x) - b(x)$. We naturally assign the weight of y to be

$$w \triangleq L_\Lambda - L_{xB}. \quad (17)$$

We denote by $\deg_{1,w}(\mathcal{Q}(x,y))$ the $(1,w)$ -weighted degree of a bivariate polynomial $\mathcal{Q}(x,y)$ (refer to [9, 16] for a detailed description of “weighted degree”). It is worth noting that the weight w may take negative values, beyond the traditional notion. Secondly, passing through the point (α^{-i}, ∞) with multiplicity m has the special meaning, i.e., the companion polynomial $\bar{\mathcal{Q}}(x,y) = \mathcal{Q}(x, 1/y)y^{P_y}$ passes $(\alpha^{-i}, 0)$ at m times. Finally, it is worth clarifying the (unconventional) relation between the power of y in $\mathcal{Q}(x,y)$, denoted by P_y , and the $(1,w)$ -weighted degree of $\mathcal{Q}(x,y)$, denoted by $\deg_{1,w}(\mathcal{Q}(x,y))$. P_y is no longer implicitly $\lfloor \frac{\deg_{1,w}(\mathcal{Q}(x,y))}{w} \rfloor$ as in the case of polynomial interpolation, but a more sophisticated function of $\deg_{1,w}(\mathcal{Q}(x,y))$, as will be characterized in (22) in an optimal setup.

Example 3. (i). For the case presented in Example 1, the weight is set to $w = -1$. The $n = 15$ interpolation points are

$$\begin{array}{ccccccccc} (1, \alpha^6) & (\alpha^{-1}, \alpha^8) & (\alpha^{-2}, \alpha^{12}) & (\alpha^{-3}, \alpha^{13}) & (\alpha^{-4}, \alpha^{11}) & & & & \\ (\alpha^{-5}, \alpha^9) & (\alpha^{-6}, \infty) & (\alpha^{-7}, \alpha^4) & (\alpha^{-8}, \alpha^{12}) & (\alpha^{-9}, \alpha^{13}) & & & & \\ (\alpha^{-10}, \alpha^{13}) & (\alpha^{-11}, \infty) & (\alpha^{-12}, \alpha^6) & (\alpha^{-13}, \alpha^{10}) & (\alpha^{-14}, 1) & & & & \end{array}$$

The following $(1, -1)$ -weighted degree bivariate polynomial passes each of the above points 7 times.

$$\begin{aligned} \mathcal{Q}(x,y) = & \\ y^0. & [\alpha^7 + \alpha^9x + \alpha^{14}x^2 + \alpha^5x^3 + \alpha^2x^4 + \alpha^8x^5 + \alpha^{12}x^6 + \alpha^6x^7 + \alpha x^8 + \alpha^2x^9 + \alpha^9x^{10} + \alpha x^{12} \\ & + \alpha^5x^{13} + \alpha^{12}x^{14} + \alpha^7x^{15}] + \\ y^1. & [\alpha^{11} + \alpha^{11}x + \alpha^{14}x^2 + \alpha^9x^3 + \alpha^{12}x^4 + \alpha^{11}x^5 + \alpha^{11}x^6 + x^7 + \alpha^4x^8 + \alpha^7x^9 + \alpha^9x^{10} + \alpha^{11}x^{11} \\ & + \alpha^8x^{12} + \alpha^{13}x^{13} + \alpha^{14}x^{14} + \alpha^{10}x^{15} + \alpha^2x^{16} + \alpha x^{17}] \\ & + \dots + \dots + \\ y^{15}. & [\alpha^{13} + \alpha^{14}x + \alpha^7x^2 + \alpha^2x^3 + x^4 + \alpha^3x^5 + \alpha^5x^6 + \alpha^6x^7 + \alpha^9x^8 + x^9 + \alpha^2x^{10} + \alpha^{14}x^{11} \\ & + \alpha^{11}x^{12} + x^{13} + \alpha^6x^{14} + \alpha^2x^{15} + \alpha^3x^{16} + \alpha^2x^{17} + \alpha^{11}x^{18} + \alpha^{12}x^{19} + \alpha^2x^{20} + \alpha^5x^{21} \\ & + \alpha^7x^{22} + \alpha^3x^{23} + \alpha^{14}x^{24} + \alpha^{10}x^{25} + \alpha^6x^{26} + \alpha^7x^{27} + \alpha x^{28} + \alpha x^{29} + \alpha^2x^{30}] + \\ y^{16}. & [\alpha^6 + x + \alpha^{12}x^2 + \alpha^{12}x^3 + \alpha^6x^5 + \alpha^7x^6 + \alpha^8x^7 + \alpha x^9 + \alpha^3x^{10} + \alpha^9x^{11} + \alpha^4x^{12} \\ & + \alpha^9x^{13} + \alpha^8x^{14} + \alpha^4x^{15} + \alpha^2x^{16} + \alpha x^{17} + \alpha^{10}x^{18} + \alpha^6x^{19} + \alpha^4x^{20} + \alpha^8x^{21} + \alpha^{14}x^{22} \\ & + \alpha^9x^{23} + \alpha^{11}x^{24} + \alpha^{11}x^{25} + \alpha^9x^{26} + x^{27} + \alpha^6x^{28} + \alpha^{10}x^{29} + \alpha^{14}x^{30} + \alpha^2x^{31}]. \end{aligned}$$

Note the $(1, -1)$ -weighted degree of the above $\mathcal{Q}(x,y)$ is 16, which is beyond the conventional notion of degree.

(ii). For the case given in Example 2, the weight is set to $w = 3$. The $n = 15$ interpolation points are

$$\begin{array}{ccccc} (1, \infty) & (\alpha^{-1}, \alpha^5) & (\alpha^{-2}, \alpha^2) & (\alpha^{-3}, \alpha^9) & (\alpha^{-4}, \alpha^5) \\ (\alpha^{-5}, \alpha^4) & (\alpha^{-6}, \alpha^{11}) & (\alpha^{-7}, \alpha) & (\alpha^{-8}, \alpha^4) & (\alpha^{-9}, \alpha^{13}) \\ (\alpha^{-10}, \alpha^{13}) & (\alpha^{-11}, \alpha) & (\alpha^{-12}, \alpha^{11}) & (\alpha^{-13}, \alpha^3) & (\alpha^{-14}, \alpha^5) \end{array}$$

The following $(1, 3)$ -weighted degree bivariate polynomial passes each of the above points 7 times.

$$\mathcal{Q}(x, y) =$$

$$\begin{aligned} y^0. & [\alpha^4 + \alpha^{14}x + \alpha^2x^2 + \alpha^9x^3 + \alpha^{11}x^4 + \alpha^{14}x^6 + \alpha^{12}x^7 + \alpha^4x^8 + \alpha^8x^9 + \alpha^2x^{10} + \alpha^5x^{11} + \alpha^{11}x^{12} \\ & + \alpha^{14}x^{13} + \alpha^{10}x^{14} + \alpha^6x^{15} + \alpha^3x^{16} + \alpha^{14}x^{17} + \alpha^{13}x^{18} + \alpha^5x^{19} + \alpha^{12}x^{21} + x^{22} + \alpha^3x^{23} + \alpha^{11}x^{24} \\ & + \alpha^8x^{25} + \alpha^8x^{26} + \alpha^{10}x^{28} + \alpha^6x^{31} + \alpha^6x^{32} + \alpha^7x^{33} + \alpha^{13}x^{34} + \alpha^5x^{35} + \alpha x^{36} + \alpha^{12}x^{37} + \alpha^2x^{38} \\ & + x^{39} + \alpha^4x^{40} + \alpha^2x^{41} + \alpha^{10}x^{42} + \alpha^6x^{43} + \alpha^7x^{44} + \alpha^3x^{45} + \alpha^{11}x^{46} + \alpha^5x^{47} + \alpha^3x^{48}] + \\ y^1. & [\alpha^8 + \alpha^7x + \alpha^3x^2 + \alpha^{14}x^3 + \alpha^{10}x^4 + \alpha^9x^5 + \alpha^3x^6 + \alpha^8x^7 + \alpha x^{10} + \alpha^{14}x^{11} + \alpha^6x^{12} + \alpha^4x^{13} \\ & + \alpha^6x^{14} + \alpha^4x^{15} + \alpha^6x^{16} + \alpha^4x^{17} + \alpha^9x^{18} + \alpha^7x^{19} + x^{20} + \alpha^{10}x^{21} + \alpha^{13}x^{22} + \alpha^9x^{23} \\ & + \alpha^9x^{24} + \alpha^5x^{25} + \alpha^{12}x^{26} + \alpha^{10}x^{27} + \alpha^{12}x^{28} + \alpha^4x^{29} + \alpha^{10}x^{30} + \alpha^{11}x^{31} + \alpha^7x^{32} + \alpha^6x^{34} \\ & + \alpha^{10}x^{35} + x^{36} + \alpha^3x^{37} + \alpha^5x^{38} + \alpha^2x^{39} + \alpha^{10}x^{40} + \alpha^3x^{41} + \alpha x^{42} + \alpha^8x^{43} + \alpha^6x^{45}] \\ & + \dots + \dots + \\ y^{12}. & [1 + \alpha^4x^2 + \alpha^4x^3 + \alpha^6x^4 + \alpha^4x^6 + \alpha^{14}x^7 + \alpha^{12}x^8 + \alpha^{11}x^9 + \alpha^{11}x^{10} + \alpha^3x^{11}] + \\ y^{13}. & [\alpha^{11}x + \alpha^{10}x^2 + \alpha^{13}x^3 + \alpha^5x^4 + \alpha^{11}x^5 + \alpha^{10}x^6 + \alpha^{13}x^7 + \alpha^5x^8] \end{aligned}$$

□□

Lemma 5 Let $\mathcal{Q}(x, y)$ be a bivariate polynomial passing through all n points $\{(1, y_0), (\alpha^{-1}, y_1), (\alpha^{-2}, y_2), \dots, (\alpha^{-(n-1)}, y_{n-1})\}$, where y_i is defined in (16), each with multiplicity m . If

$$(t - L_\Lambda)P_y + \deg_{1,w}(\mathcal{Q}(x, y)) < tm, \quad (18)$$

where P_y denotes the power of y in $\mathcal{Q}(x, y)$ and $\deg_{1,w}(\mathcal{Q}(x, y))$ (where w is defined in (17)) denotes the $(1, w)$ -weighted degree of $\mathcal{Q}(x, y)$, then $\mathcal{Q}(x, y)$ contains all factors of the form $y\lambda(x) - b(x)$ which pass through t ($t \geq L_\Lambda$) points.

Proof: Let $y\lambda(x) - b(x)$ be a polynomial passing through t points. Then,

$$g(x) \triangleq \lambda^{P_y}(x) \mathcal{Q}\left(x, \frac{b(x)}{\lambda(x)}\right)$$

is a polynomial and contains at least tm roots, i.e., all roots of $\Lambda^*(x)$ each with multiplicity m . We proceed to show that the degree of $g(x)$ is at most $(t - L_\Lambda)P_y + \deg_{1,w}(\mathcal{Q}(x, y))$. This is true for the starting term without involving y , i.e., $\lambda^{P_y}(x)\mathcal{Q}(x, 0)$, following the fact

$$\deg(\lambda^{P_y}(x)\mathcal{Q}(x, 0)) \leq \deg(\lambda(x))P_y + \deg_{1,w}(\mathcal{Q}(x, y)) \leq (t - L_\Lambda)P_y + \deg_{1,w}(\mathcal{Q}(x, y)),$$

where $\deg(\lambda(x)) \leq (t - L_\Lambda)$ is due to Lemma 3.(iv). It also holds true for the ending term associated with y^{P_y} , i.e., $b^{P_y}(x) \cdot [y^{P_y}] \mathcal{Q}(x, y)$, as follows

$$\begin{aligned} \deg(b^{P_y}(x) \cdot [y^{P_y}] \mathcal{Q}(x, y)) &\leq \deg(b(x))P_y + (\deg_{1,w}(\mathcal{Q}(x, y)) - wP_y) \\ &= \deg_{1,w}(\mathcal{Q}(x, y)) + P_y(\deg(b(x)) + L_{xB} - L_\Lambda) \\ &\leq \deg_{1,w}(\mathcal{Q}(x, y)) + P_y(t - L_\Lambda), \end{aligned}$$

where $\deg(b(x)) + L_{xB} \leq t$ is due to Lemma 3.(iv). Hence, it trivially holds true for the intermediate terms. Finally, the fact

$$\deg(g(x)) \leq (t - L_\Lambda)P_y + \deg_{1,w}(\mathcal{Q}(x, y)) < tm$$

indicates the polynomial $g(x)$ has more roots than its degree. This is possible only with $g(x) \equiv 0$. Therefore, we conclude that $y \cdot \lambda(x) - b(x)$ divides $\mathcal{Q}(x, y)$. $\square\square$

On the other hand, a sufficient condition for $\mathcal{Q}(x, y)$ to pass through all n points $\{(1, y_0), (\alpha^{-1}, y_1), (\alpha^{-2}, y_2), \dots, (\alpha^{-(n-1)}, y_{n-1})\}$, each with multiplicity m , is that the number of coefficients (degrees of freedom) of $\mathcal{Q}(x, y)$ is greater than the number of linear constraints. Note that the degrees of freedom, denoted by N_{free} , is easily seen to be

$$N_{\text{free}} = \sum_{i=0}^{P_y} (\deg_{1,w}(\mathcal{Q}(x, y)) + 1 - iw) = \frac{(2 \deg_{1,w}(\mathcal{Q}(x, y)) + 2 - wP_y)(P_y + 1)}{2} \quad (19)$$

whereas passing through a point with multiplicity m results in $\frac{m(m+1)}{2}$ linearly independent constraints (readers are referred to [9] for detailed description). Thus, the number of linear constraints, denoted by N_{cstr} , is

$$N_{\text{cstr}} = \frac{nm(m+1)}{2}. \quad (20)$$

We proceed to maximize the degrees of freedom subject to fixed number of errors $e = t$ and the fixed multiplicity m and the constraint (18), as follows

$$\begin{aligned} N_{\text{free}} &= \frac{(2 \deg_{1,w}(\mathcal{Q}(x, y)) + 2 - wP_y)(P_y + 1)}{2} \\ &\leq \frac{(2(tm - 1 - (t - L_\Lambda)P_y) + 2 - wP_y)(P_y + 1)}{2} \\ &= (tm - P_y(t - t_0))(P_y + 1) \\ &= -(t - t_0)P_y^2 + P_y(tm - t + t_0) + tm \\ &= -(t - t_0) \left(P_y - \frac{tm - t + t_0}{2t - 2t_0} \right)^2 + \frac{(tm + t - t_0)^2}{4(t - t_0)} \end{aligned} \quad (21)$$

where “=” is achieved in “ \leq ” if and only if

$$\deg_{1,w}(\mathcal{Q}^*(x, y)) = tm - 1 - (t - L_\Lambda)P_y \quad (22)$$

which optimally accommodates the zero constraint (18), and the first equality is due to the fact

$$L_\Lambda - \frac{w}{2} = \frac{L_\Lambda + L_{xB}}{2} = \frac{n - k + 1}{2} = t_0.$$

Clearly, the maximum degrees of freedom is achieved by choosing P_y^* to be the closest integer to $\frac{tm-t+t_0}{2t-2t_0}$, i.e.,

$$P_y^* = \left\lfloor \frac{tm - t + t_0}{2t - 2t_0} + 0.5 \right\rfloor = \left\lfloor \frac{tm}{2t - 2t_0} \right\rfloor. \quad (23)$$

Therefore, the optimal choice of m is the minimum integer that enforces $N_{\text{free}} > N_{\text{cstr}}$, i.e.,

$$-(t - t_0) \left(\left\lfloor \frac{tm}{2t - 2t_0} \right\rfloor - \frac{tm - t + t_0}{2t - 2t_0} \right)^2 + \frac{(tm + t - t_0)^2}{4(t - t_0)} > \frac{nm(m + 1)}{2}. \quad (24)$$

We next present an explicit construction of a valid (but not necessary optimal) multiplicity. Note the maximum degrees of freedom is bounded by

$$\begin{aligned} \max\{N_{\text{free}}\} &= -(t - t_0) \left(\left\lfloor \frac{tm}{2t - 2t_0} \right\rfloor - \frac{tm - t + t_0}{2t - 2t_0} \right)^2 + \frac{(tm + t - t_0)^2}{4(t - t_0)} \\ &\geq -\frac{t - t_0}{4} + \frac{(tm + t - t_0)^2}{4(t - t_0)}. \end{aligned} \quad (25)$$

Hence, to solve for the linear equation system, it suffices to enforce

$$\frac{(tm + t - t_0)^2}{4(t - t_0)} - \frac{t - t_0}{4} > \frac{nm(m + 1)}{2}, \quad (26)$$

which is reduced to

$$m^2(t^2 - 2n(t - t_0)) - 2m(t - t_0)(n - t) > 0. \quad (27)$$

The above inequality holds true for sufficiently large m if and only if

$$t^2 - 2n(t - t_0) > 0$$

which in turn indicates that

$$t < n - \sqrt{n(n - d)} \quad (28)$$

which is equal to that of [9]. When $t < n - \sqrt{n(n - d)}$, it suffices to choose the multiplicity m to be

$$m^* = \left\lfloor \frac{2(t - t_0)(n - t)}{t^2 - 2n(t - t_0)} + 1 \right\rfloor = \left\lfloor \frac{t(2t_0 - t)}{t^2 - 2n(t - t_0)} \right\rfloor. \quad (29)$$

The following derives a lower bound on the optimal multiplicity m_{opt} by relaxing P_y to a real number. More specifically, with P_y being a real number, the constraint $N_{\text{free}} > N_{\text{cstr}}$ simplifies to

$$\begin{aligned} &\frac{(tm + t - t_0)^2}{4(t - t_0)} > \frac{nm(m + 1)}{2} \\ \Leftrightarrow &m^2(t^2 - 2n(t - t_0)) - 2m(t - t_0)(n - t) + (t - t_0)^2 > 0 \end{aligned}$$

which indicates

$$\begin{aligned}
m &> \frac{(t-t_0)(n-t) + \sqrt{(t-t_0)^2(n-t)^2 - (t^2 - 2n(t-t_0))(t-t_0)^2}}{t^2 - 2n(t-t_0)} \\
&= \frac{(t-t_0) \left(n-t + \sqrt{n(n-d)} \right)}{t^2 - 2n(t-t_0)} \\
&= \frac{t-t_0}{n - \sqrt{n(n-d)} - t}.
\end{aligned} \tag{30}$$

We thus conclude that the optimal value of m is within the range

$$\left\lceil 1 + \frac{t-t_0}{n - \sqrt{n(n-d)} - t} \right\rceil \leq m_{\text{opt}} \leq \left\lfloor \frac{t(2t_0-t)}{t^2 - 2n(t-t_0)} \right\rfloor. \tag{31}$$

It is worth noting that the LECC t , the multiplicity m and the y -degree P_y are all irrelevant to the degrees of $\Lambda(x)$ or $B(x)$ individually. We ought to ensure there is no negative freedom terms with the above choice, which is easily verified by the fact that $\deg_{1,w}(\mathcal{Q}(x,y)) + 1 - iw$ is always positive for $i = 0, 1, 2, \dots, P_y$.

We summarize the above discussions into the following lemma.

Lemma 6 *Let t satisfy (28), the multiplicity m be chosen as in (29) and the y -degree P_y of bivariate polynomial $\mathcal{Q}(x,y)$ as in (23). Then, all valid polynomials $\lambda(x) \cdot y - b(x)$ that pass through exactly t points are factors of the minimum $(1, w)$ -weighted (where w is defined in (17)) degree polynomial $\mathcal{Q}(x,y)$ that passes through $\{(\alpha^{-i}, y_i)\}_{i=0}^{n-1}$ (where y_i is defined in (16)), each with multiplicity m .*

However, different values of e require different values of P_y and m . We next show that for a given LECC t , it suffices to choose a unified value P_y and m based upon the LECC t to identify all valid polynomials $\lambda(x) \cdot y - b(x)$ corresponding to up to t errors. Since the degrees of freedom is independent of the actual number of errors, it is only left to show that

$$(e - L_\Lambda)P_y + \deg_{1,w}(\mathcal{Q}(x,y)) = (e + t - 2t_0)P_y + (t - t_0) < em,$$

where $\deg_{1,w}(\mathcal{Q}(x,y))$ satisfies (22) and m satisfies (30). By assumption the above inequality holds when $e = t$, i.e.,

$$\frac{(2t - 2t_0)P_y}{tm - t + t_0} < 1.$$

Therefore, it suffices to show

$$\begin{aligned}
&\frac{(2t - 2t_0)P_y}{tm - t + t_0} \geq \frac{(t + e - 2t_0)P_y}{em - t + t_0} \\
&\Leftrightarrow (t - e)P_y((t - 2t_0)m + (t - t_0)) \leq 0 \\
&\Leftrightarrow m \geq \frac{t - t_0}{2t_0 - t},
\end{aligned}$$

which holds true following (30) and the fact $2t_0 = d > n - \sqrt{n(n-d)}$.

The following theorem wraps up the subsection.

Theorem 1 *Let $\Lambda(x)$ and $B(x)$ be the error locator and correction polynomials, respectively, computed by the Berlekamp-Massey algorithm. For any given t satisfying (28), if we choose the multiplicity m as in (29) and the y -degree P_y of bivariate polynomial $\mathcal{Q}(x, y)$ as in (23), then the minimum $(1, L_\Lambda - L_{xB})$ -weighted degree polynomial $\mathcal{Q}(x, y)$ that passes through each of n distinct points, $\{(\alpha^{-i}, y_i)\}_{i=0}^{n-1}$ (where y_i is defined in (16)), with multiplicity m , contains all factors $\lambda(x)y - b(x)$ which pass through at least L_Λ but at most t out of n points.*

Remarks: The above theorem can be viewed as a generalization of Lemma 5 in [9] from polynomial to rational curve-fitting. Although we do not address this explicitly, the choice of weight of y in $\mathcal{Q}(x, y)$ is optimal in the sense of maximizing the LECC t . It optimally trades off between (18) and (27).

Example 4. For the (15, 5) Reed-Solomon code, we have $n - \sqrt{n(n-d)} = 7.254$. We consider correcting up to $t = 7$ errors. It suffices to choose the multiplicity $m = \lfloor \frac{t(2t_0-t)}{t^2-2n(t-t_0)} \rfloor = 7$. Let the degree of y be chosen as $P_y = \lfloor \frac{tm}{2t-2t_0} \rfloor = 16$, following (23), and the degree of $\mathcal{Q}(x, y)$, $\deg_{1,w}(\mathcal{Q}(x, y))$, as in (22). As a result, the degrees of freedom is $(tm - P_y(t - t_0))(P_y + 1) = 425$ whereas the number of linear constraints is $nm(m + 1)/2 = 420$. The lower bound of m is $\lfloor 1 + \frac{t-t_0}{n-\sqrt{n(n-d)}-t} \rfloor = 6$. When $m = 6$, it can be verified that the maximum degrees of freedom and the number of constraints are both 315. Thus, $m = 6$ is not satisfactory and the optimal choice of multiplicity is $m = 7$. For the received words given in Examples 1 & 2, the minimum (weighted) degree bivariate polynomials $\mathcal{Q}(x, y)$ constructed with $m = 7$ and $P_y = 16$ are presented in Example 3. In contrast, to achieve the same LECC $t = 7$, the Guruswami-Sudan algorithm requires multiplicity $m = 16$ with $P_y = 31$ (recall that P_y is a natural upper bound on the list size). $\square\square$

C. Rational Factorization

In this section we apply rational factorization to obtain $\frac{b(x)}{\lambda(x)}$, following the developments in [21, 28]. However, our particular application is complicated by not knowing *a priori* the degrees of $\lambda(x)$ and $b(x)$.

Define

$$f(x) \triangleq \frac{b(x)}{\lambda(x)} \quad (32)$$

which is well defined, as indicated by Lemma 3 that $b(x)$ and $\lambda(x)$ are coprime. We, thus, only need to determine $f(x)$ to construct $\Lambda^*(x)$. Note that $\lambda_0 = 1$, as indicated by Lemma 3. Consequently, $f(x)$ can be expressed by an infinite-length non-negative power series

$$f(x) = \frac{b(x)}{\lambda(x)} = s_0 + s_1x + s_2x^2 + \dots s_ix^i + \dots$$

When $\mathcal{Q}(x, y)$ contains factors of the form $y - f(x)$, we can obtain the infinite-length power series of the rational function $f(x)$ through making use of the following factorization procedure [21, 28].

Rational Factorization Procedure

0. Initialization: $i = 0$ and $\mathcal{Q}^{(0)}(x, y) \leftarrow \mathcal{Q}(x, y)$.
1. Determine the roots of $\mathcal{Q}^{(i)}(0, y)$.
2. For each root s ,
 - Compute the shifted polynomial: $\hat{\mathcal{Q}}^{(i)}(x, y) \leftarrow \mathcal{Q}^{(i)}(x, s + y)$.
 - Transform and then remove x factors: $\mathcal{Q}^{(i+1)}(x, y) \leftarrow \hat{\mathcal{Q}}^{(i)}(x, xy)/x^a$, where x^a denotes the largest power of x contained as a factor in $\hat{\mathcal{Q}}^{(i)}(x, xy)$.
3. Set $i \leftarrow i + 1$ and repeat Steps 1 and 2 for the derivative $\mathcal{Q}^{(i)}(x, y)$ with respect to each root s .

We proceed to show that a pertinent finite-length, say L_s , power series of $s(x)$ suffices to retrieve $b(x)$ and $\lambda(x)$. Note we have

$$\frac{b(x)}{\lambda(x)} \equiv s(x) \pmod{x^{L_s}}. \quad (33)$$

It can be efficiently solved by the Berlekamp-Massey algorithm given that L_s is large enough, as demonstrated below.

We first assume that the degrees of $b(x)$ and $\lambda(x)$ are known *a priori*. When $\deg(b(x)) < \deg(\lambda(x))$, $\lambda(x)$ is then uniquely determined (upon normalization) by the Berlekamp-Massey algorithm (note that $s(x) \pmod{x^{L_s}}$, $\lambda(x)$, and $b(x)$ correspond to syndrome, error locator, and error evaluator polynomials, respectively), with $L_s = 2 \deg(\lambda(x))$. Note that the condition $\deg(b(x)) < \deg(\lambda(x))$ implies

$$\sum_{j=0}^{\deg(\lambda(x))} \lambda_j s_{i-j} = 0, \quad i = \deg(\lambda(x)), \deg(\lambda(x)) + 1, \dots, 2 \deg(\lambda(x)) - 1.$$

Therefore, the LFSR length described by $\lambda(x)$ is $L_\lambda = \deg(\lambda(x))$.

When $\deg(b(x)) \geq \deg(\lambda(x))$, let

$$\frac{b(x)}{\lambda(x)} = a(x) + \frac{b'(x)}{\lambda(x)}$$

where $a(x)$ is a polynomial of degree $\deg(b(x)) - \deg(\lambda(x))$ and the degree of $b'(x)$ is less than that of $\lambda(x)$. Let $j_1 = \deg(b(x)) - \deg(\lambda(x))$ and $j_2 = \deg(b(x)) + \deg(\lambda(x))$. We observe that $\lambda(x)$ generates the sequence $(s_0 - a_0), \dots, (s_{j_1} - a_{j_1}), s_{j_1+1}, \dots, s_{j_2}$, consequently, its subsequence $s_{j_1+1}, s_{j_1+2}, \dots, s_{j_2}$. Therefore, $\lambda(x)$ can be uniquely determined (upon normalization) using the sequence $s_{j_1+1}, s_{j_1+2}, \dots, s_{j_2}$, and subsequently $b(x)$ is computed by following the equality

$$b(x) = s(x)\lambda(x) \pmod{x^{\deg(b(x))+1}}.$$

Clearly, in this case it suffices to compute the power series expansion of $f(x)$ up to length $L_s = \deg(b(x)) + \deg(\lambda(x)) + 1$.

Because we do not know *a priori* the degrees $\deg(b(x))$ and $\deg(\lambda(x))$, we have to take into account the worst cases. When $\deg(b(x)) = t - L_{xB}$ and $\deg(\lambda(x)) = 0$, the Berlekamp-Massey algorithm has to start from the syndrome $s_{t-L_{xB}+1}$, on the other hand, $\deg(\lambda(x))$ may be as large as $t - L_{\Lambda}$, hence the Berlekamp-Massey algorithm requires $2(t - L_{\Lambda})$ syndromes in the worst case. Therefore we may safely set the length of $s(x)$ to:

$$L_s = t - L_{xB} + 1 + 2(t - L_{\Lambda}) = 3t + 1 - 2t_0 - L_{\Lambda}. \quad (34)$$

We thus obtain $\lambda(x)$ by applying the Berlekamp-Massey algorithm which starts from the syndrome $s_{t-L_{xB}+1}$ and iterates $2(t - L_{\Lambda})$ times. Thereafter, we compute $b(x)$ via

$$b(x) = s(x)\lambda(x) \pmod{x^{t-L_{xB}+1}}. \quad (35)$$

After we have determined the error locator polynomial $\Lambda'(x) = \Lambda(x)\lambda(x) + xB(x)b(x)$, we can subsequently identify all error locations and apply Forney's formula to compute the corresponding error magnitudes [4].

Example 5. (i). For the bivariate polynomial $\mathcal{Q}(x, y)$ constructed in Example 3.(i), applying the proposed rational factorization returns three candidate rational functions.

(1). $\lambda(x) = 1 + \alpha x + \alpha^2 x^2$ and $b(x) = \alpha + \alpha^2 x$. The candidate error locator polynomial is constructed

$$\begin{aligned} \Lambda'(x) &= \lambda(x) \cdot \Lambda(x) + b(x) \cdot xB(x) \\ &= 1 + \alpha^9 x + \alpha^5 x^2 + \alpha^{10} x^3 + \alpha^4 x^4 + \alpha^7 x^5 + x^6 + \alpha^4 x^7 \\ &= (1 - x)(1 - \alpha^3 x)(1 - \alpha^6 x)(1 - \alpha^7 x)(1 - \alpha^{10} x)(1 - \alpha^{11} x)(1 - \alpha^{12} x) \end{aligned}$$

which produces the following candidate codeword

$$\mathbf{c} = [\alpha^2, \alpha^6, \alpha^{10}, \alpha^8, \alpha^7, \alpha^5, \alpha^6, \alpha^{12}, \alpha, \alpha^2, \alpha^3, \alpha^9, \alpha^9, \alpha^7, \alpha].$$

(2). $\lambda(x) = 1 + \alpha^{12} x$ and $b(x) = \alpha^7$. The candidate error locator polynomial is constructed as

$$\Lambda'(x) = 1 + \alpha^2 x^1 + \alpha^4 x^2 + \alpha^5 x^3 + \alpha^2 x^4 + \alpha^4 x^5 + \alpha^{10} x^6$$

which has less than 6 distinct roots in $\text{GF}(16)$ and thus is spurious.

(3). $\lambda(x) = 1 + \alpha x + \alpha^2 x^2$ and $b(x) = \alpha^8$. The corresponding candidate error locator polynomial is

$$\begin{aligned} \Lambda'(x) &= 1 + \alpha^9 x^1 + \alpha^{11} x^3 + \alpha^5 x^4 + \alpha^{12} x^6 + \alpha^6 x^7 \\ &= (1 - \alpha x)(1 - \alpha^3 x)(1 - \alpha^6 x)(1 - \alpha^8 x)(1 - \alpha^9 x)(1 - \alpha^{11} x)(1 - \alpha^{13} x) \end{aligned}$$

which successfully retrieves the transmitted codeword

$$\mathbf{c} = [0, \alpha^{12}, \alpha^{10}, \alpha^{11}, \alpha^7, \alpha^5, \alpha^{11}, \alpha^{11}, \alpha^6, \alpha^8, \alpha^{14}, \alpha^{11}, \alpha^6, \alpha^2, \alpha].$$

(ii). For the bivariate polynomial $\mathcal{Q}(x, y)$ constructed in Example 3.(ii), applying the proposed rational factorization returns one candidate rational function: $\lambda(x) = 1$ and $b(x) = \alpha^{12} + \alpha^{10}x + \alpha^4x^2 + \alpha^7x^3$. Consequently, the candidate error locator polynomial is constructed

$$\begin{aligned}\Lambda'(x) &= \lambda(x) \cdot \Lambda(x) + b(x) \cdot xB(x) \\ &= 1 + \alpha^5x + \alpha^4x^3 + \alpha^{14}x^4 + \alpha^8x^5 + \alpha^{12}x^6 + \alpha^{14}x^7 \\ &= (1 - \alpha x)(1 - \alpha^4x)(1 - \alpha^7x)(1 - \alpha^8x)(1 - \alpha^{12}x)(1 - \alpha^{13}x)(1 - \alpha^{14}x)\end{aligned}$$

which produces the original codeword. It also verifies Lemma 4.(i), that there does not exist codewords within distance 6 from the received word. $\square\square$

D. Algorithmic Description and Performance Assertion

We summarize the complete list decoding algorithm and its characterization as follows.

List Decoding Algorithm for Reed-Solomon Codes

0. Initialization: Input code length n , information dimension k , and LECC t satisfying $t < n - \sqrt{n(n-d)}$. Initialize the multiplicity m based on (29), then greedily optimize it subject to the constraint (24), subsequently choose the y -degree P_y as in (23).
1. Input the received word and compute syndromes.
2. Apply the Berlekamp-Massey algorithm to determine $\Lambda(x)$ and $B(x)$. If $L_\Lambda > t$ then declare a decoding failure.
3. Perform error correction and evaluate $y_i = -\frac{\Lambda(\alpha^{-i})}{\alpha^{-i}B(\alpha^{-i})}$, $i = 0, 1, 2, \dots, n-1$.
4. If $L_{xB} > t$, then return the corresponding unique codeword when $\Lambda(x)$ has L_Λ valid roots, otherwise declare a decoding failure.
5. Apply the rational interpolation procedure to compute a $(1, L_\Lambda - L_{xB})$ -weighted-degree polynomial $\mathcal{Q}(x, y)$ that passes through $\{(\alpha^{-i}, y_i)\}_{i=0}^{n-1}$, each with multiplicity m .
6. Apply the rational factorization process to obtain finite-length power series $s(x) \pmod{x^{L_s}}$ of the rational functions $\frac{b(x)}{\lambda(x)}$.
7. For each finite-length power series $s(x) \pmod{x^{L_s}}$, do:
 - Apply the Berlekamp-Massey algorithm to determine $\lambda(x)$.
 - Compute $b(x) = s(x)\lambda(x) \pmod{x^{t-L_{xB}+1}}$.

- Construct $\Lambda'(x) = \lambda(x) \cdot \Lambda(x) + b(x) \cdot xB(x)$.
- Determine the distinct roots of $\Lambda'(x)$ within $\{\alpha^{-i}\}_{i=0}^{n-1}$.
- Compute error magnitudes using the Forney's formula if the number of distinct roots is equal to the degree $L_{\Lambda'}$.

8. Return the list of codewords which are within distance t from the received word.

Theorem 2 *Let a LECC t satisfy (28). If the multiplicity m is chosen as in (29) and subsequently the y -degree P_y as in (23), then the proposed list decoding algorithm produces all codewords within distances t from a received word.*

Remarks: The proposed list decoding algorithm can be applied to the decoding of generalized Reed-Solomon codes, and error-and-erasure decoding using the method in [8].

Let t_{opt} denote the achievable LECC

$$t_{\text{opt}} \triangleq \lceil n - 1 - \sqrt{n(n-d)} \rceil. \quad (36)$$

Corollary 1 *The number of codewords that lie within (strictly less than) Hamming distance $n - \sqrt{n(n-d)}$ from a received word is $O(n - \sqrt{n(n-d)})$.*

Proof: The number of such codewords is upper bounded by the degree P_y that corresponds to $t = t_{\text{opt}}$. Substituting (29) into (23), we further obtain

$$\begin{aligned} P_y &= O \left(\frac{t_{\text{opt}}}{2t_{\text{opt}} - d} \cdot \frac{t_{\text{opt}}(d - t_{\text{opt}})}{t_{\text{opt}}^2 - nt_{\text{opt}} + nd} \right) \\ &= O \left(\frac{t_{\text{opt}}}{2t_{\text{opt}} - d} \cdot \frac{t_{\text{opt}}(d - t_{\text{opt}})}{(n - \sqrt{n(n-d)} - t_{\text{opt}})(n + \sqrt{n(n-d)} - t_{\text{opt}})} \right) \\ &= O \left(\frac{t_{\text{opt}}}{2t_{\text{opt}} - d} \cdot \frac{t_{\text{opt}}(2t_0 - t_{\text{opt}})}{n + \sqrt{n(n-d)} - t_{\text{opt}}} \right) \\ &= O \left(\frac{n - \sqrt{n(n-d)}}{2n - 2\sqrt{n(n-d)} - d} \cdot \frac{(n - \sqrt{n(n-d)})(d - (n - \sqrt{n(n-d)}))}{n + \sqrt{n(n-d)} - (n + \sqrt{n(n-d)})} \right) \\ &= O \left(n - \sqrt{n(n-d)} \right) \end{aligned}$$

as desired.

We next justify the treatment of the term $n - \sqrt{n(n-d)} - t_{\text{opt}}$ as $O(1)$. Indeed, $n - \sqrt{n(n-d)} - t_{\text{opt}}$ can be arbitrarily small with appropriate choices of n . However, if we replace t_{opt} by

$$t' = \lfloor n - \sqrt{n(n-d)} - 0.5 \rfloor,$$

then we immediately have $0.5 \leq n - \sqrt{n(n-d)} - t' < 1.5$, i.e., $n - \sqrt{n(n-d)} - t' = O(1)$. On the other hand, we have $0 \leq t_{\text{opt}} - t' \leq 1$, in which difference by 1 can be safely ignored in light of asymptotic behavior. $\square\square$

Remarks: In [9], the product term $(n - \sqrt{n(n-d)} - t_{\text{opt}})(n + \sqrt{n(n-d)} - t_{\text{opt}})$ is regarded as constant $O(1)$, which results in a quadratic bound $O(\sqrt{kn^3})$ for list size, which otherwise is a linear term as well. Given a LECC $t < n - \sqrt{n(n-d)}$ for (n, k) Reed-Solomon code, the explicit construction of the Guruswami-Sudan algorithm is given in [9]:

$$P_y = \left\lfloor \frac{1}{n-d} \left((n-t) \left(1 + \left\lfloor \frac{n(n-d) + \sqrt{n^2(n-d)^2 + 4((n-t)^2 - n(n-d))}}{2((n-t)^2 - n(n-d))} \right\rfloor \right) - 1 \right) \right\rfloor$$

whereas the explicit construction of the proposed algorithm shows

$$P_y = \left\lfloor \frac{t}{2t - 2t_0} \cdot \left\lfloor \frac{t(2t_0 - t)}{t^2 - 2n(t - t_0)} \right\rfloor \right\rfloor.$$

Figures 2 and 3 shed some light on the tightness of the list size bound between the two algorithms. Note that due to integer precision, it is possible that under a fixed multiplicity m , larger LECC t causes smaller list size, specifically, let t and t' , $t < t'$ be permissible for the same m , then it holds $\lfloor \frac{tm}{2(t-t_0)} \rfloor \geq \lfloor \frac{t'm}{2(t'-t_0)} \rfloor$. To this end, the minimum value should be chosen in the range where the same multiplicity m applies.

Ignoring the integer precision, the ratio $\frac{P_{y,\text{GS}}}{P_{y,\text{prop}}}$ with respect to the LECC limit $t = n(1 - \sqrt{1-D})$ is expressed as

$$\frac{P_{y,\text{GS}}}{P_{y,\text{prop}}} \Big|_{t=n(1-\sqrt{1-D})} = \frac{1}{1 - \sqrt{1-D}},$$

which reveals that the derivative bound of the proposed algorithm is *universally* tighter than that of the Guruswami-Sudan algorithm on the boundary of the optimal LECC, as illustrated in Figure 4. Finally, we clarify that the above results are consistent with the work in [13, 22], in which the list- l Guruswami-Sudan bound is shown to be tight only when the algorithm degenerates to the classical (list-1) hard-decision decoding.

E. Complexity Analysis

Finally we analyze the computational complexity in terms of field operations, assuming n and k are large enough and the field cardinality $q \leq 2^n$ (which is used to bound the factorization complexity, as discussed in [9]). Following the convention, we will fix the normalized minimum distance $D = d/n$ while letting n (and d) go to infinity. Our particular interest lies in the case of the achievable LECC

t_{opt} which is defined in (36). In this case, the multiplicity chosen in (29) simplifies to

$$\begin{aligned}
m &= O\left(\frac{t_{\text{opt}}(d - t_{\text{opt}})}{t_{\text{opt}}^2 - 2nt_{\text{opt}} + nd}\right) \\
&= O\left(\frac{t_{\text{opt}}(d - t_{\text{opt}})}{(n - \sqrt{n(n-d)} - t_{\text{opt}})(n + \sqrt{n(n-d)} - t_{\text{opt}})}\right) \\
&= O\left(\frac{t_{\text{opt}}(d - t_{\text{opt}})}{n + \sqrt{n(n-d)} - t_{\text{opt}}}\right) \\
&= O\left(\frac{(n - \sqrt{n(n-d)})(d - (n - \sqrt{n(n-d)}))}{2\sqrt{n(n-d)}}\right) \\
&= O\left(\left(\sqrt{n} - \sqrt{n-d}\right)^2\right).
\end{aligned}$$

To the best of our knowledge, the up-to-date most efficient interpolation technique is through the Koetter updating procedure which exhibits complexity $O(N^2)$ with N linear constraints [15, 18]. To achieve optimal performance, the number of constraints is of the order $O(nm^2) = O\left(n(\sqrt{n} - \sqrt{k})^4\right)$. Thus, we have the following characterization regarding the interpolation complexity.

Lemma 7 *The rational interpolation of the proposed list decoding algorithm can be implemented with the complexity of*

$$O\left(n^2(\sqrt{n} - \sqrt{n-d})^8\right)$$

to achieve the LECC $t_{\text{opt}} = \lceil n - 1 - \sqrt{n(n-d)} \rceil$.

We next analyze the complexity of the proposed factorization procedure, following the development in [21]. Note the length of $s(x)$, which is factorized and used in the Berlekamp-Massey algorithm to obtain $\lambda(x)$ and $b(x)$, is bounded by

$$3t - 2t_0 - \deg(\lambda(x)) \leq 3t - 2t_0 - \deg(\lambda(x)) + (t - L_{xB}) = 4(t - t_0).$$

In [17], it is shown that the roots in $\text{GF}(q)$ of a polynomial of degree u can be found in expected time complexity $O(u^2 \cdot \log^2 u \cdot \log q)$. We observe that in each iteration “ (i) ” the degrees of the polynomials $\mathcal{Q}_j^{(i)}(0, y)$ satisfy $\sum_j \deg(\mathcal{Q}_j^{(i)}(0, y)) \leq P_y$ (See Lemma 6.2 in [21]). The root-finding complexity in one iteration is then bounded by

$$\sum_j O(\deg^2(\mathcal{Q}_j^{(i)}(0, y)) \cdot \log^2 \deg(\mathcal{Q}_j^{(i)}(0, y)) \cdot \log q) \leq O(P_y^2 \cdot \log^2 P_y \cdot \log q).$$

Thus, the computational complexity of determining roots of $\mathcal{Q}(0, y)$ in up to $4(t - t_0)$ iterations is bounded by

$$O\left((t - t_0) \cdot P_y^2 \cdot \log^2 P_y \cdot \log q\right).$$

We now analyze the shift operation $\hat{\mathcal{Q}}^{(i)}(x, y) \leftarrow \mathcal{Q}^{(i)}(x, y + s_i)$. Lemma 6.2 in [21] indicates that the step $\mathcal{Q}^{(i+1)}(x, y) \leftarrow \hat{\mathcal{Q}}^{(i)}(x, xy)/x^a$, where x^a denotes the largest power of x which divides $\hat{\mathcal{Q}}^{(i)}(x, xy)$, results in

$$\deg \left(\mathcal{Q}^{(i+1)}(0, y) \right) \leq a \leq h \leq \deg \left(\mathcal{Q}^{(i)}(0, y) \right),$$

where h denotes the multiplicity of the root s_i of $\mathcal{Q}^{(i)}(0, y)$. Hence, the parameter a is non-increasing with iterations going on. Therefore, it suffices, in each iteration, to update the first $4a(t - t_0)$ terms of $[y^l]\mathcal{Q}^{(i)}(x, y)$, $l = 0, 1, 2, \dots, P_y$, in order to determine the first $4(t - t_0)$ terms of $s(x)$. Since each term can be updated with $O(P_y)$ operations, the corresponding complexity in one iteration is

$$\sum_j 4a_j \cdot (t - t_0)P_y^2 \leq \sum_j 4 \deg(\mathcal{Q}_j^{(i)}(0, y)) \cdot (t - t_0)P_y^2 = O((t - t_0)P_y^3).$$

and thus the resulting overall complexity is $O((t - t_0)^2 P_y^3)$.

Finally, since there are at most P_y candidate polynomials $s(x)$, it takes $O(P_y(t - t_0)^2)$ operations to compute the corresponding $\lambda(x)$ and $b(x)$ through the Berlekamp-Massey algorithm. Clearly, the overall complexity of the proposed factorization procedure is dominated by the shift operation with the complexity of

$$O((t - t_0)^2 P_y^3) = O\left(n^{3/2}(\sqrt{n} - \sqrt{n-d})^7\right).$$

Therefore, we characterize the complexity of the factorization procedure as follows

Lemma 8 *Given that the field cardinality is at most 2^n , the rational factorization procedure of the proposed list decoding algorithm can be implemented with the complexity of*

$$O\left(n^{3/2}(\sqrt{n} - \sqrt{n-d})^7\right)$$

to achieve the LECC t_{opt} .

The following theorem summarizes the complexity of the proposed algorithm.

Theorem 3 *Given that the field cardinality is at most 2^n , the proposed list decoding algorithm exhibits the computational complexity in terms of field operations*

$$O\left(n^2(\sqrt{n} - \sqrt{n-d})^8\right) \tag{37}$$

to achieve the LECC $t_{opt} = \lceil n - 1 - \sqrt{n(n-d)} \rceil$.

Remarks: Note that the multiplicity dictates the complexity. It is insightful to also compare the minimum multiplicities between the Guruswami-Sudan algorithm and the proposed algorithm. For an intermediate $\frac{n-k}{2} < t < n - \sqrt{n(n-d)}$, the value of multiplicity given in [9] is, under our notations,

$$m_{GS} = 1 + \left\lfloor \frac{n(n-d) + \sqrt{(n^2(n-d)^2 + 4((n-t)^2 - n(n-d))}}{2(t^2 - 2nt + nd)} \right\rfloor$$

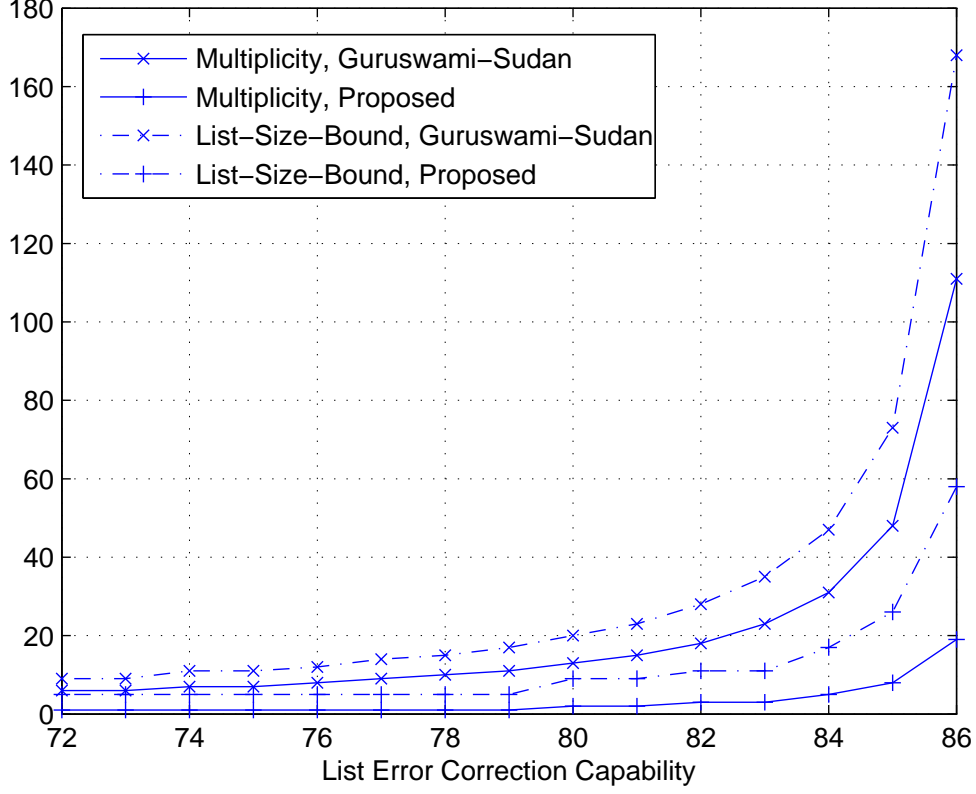


Figure 2: Multiplicity and list-size-bound as functions of list error correction capability for the (255, 112) Reed-Solomon code.

whereas the value of multiplicity of the proposed approach is given in (29), i.e.,

$$m_{\text{prop}} = 1 + \left\lfloor \frac{2(t - t_0)(n - t)}{t^2 - 2nt + nd} \right\rfloor.$$

(recall that the above m_{prop} is a sufficient value but not necessarily the minimum/optimal value, as indicated in (31).)

Figures 2 and 3 plot the above two multiplicity functions for the (255, 112) Reed-Solomon code and the (2047, 1647) Reed-Solomon code, respectively. We observe in Figure 3, where the code rate is 0.8, that to correct extra five erroneous symbols, the proposed algorithm requires multiplicity $m = 1$, whereas the Guruswami-Sudan algorithm requires multiplicity $m = 143$; to achieve the LECC limit of correcting extra 11 errors, the former requires the multiplicity $m = 26$, whereas the latter requires $m = 2197$.

We further consider the ratio $\frac{m_{\text{GS}}}{m_{\text{prop}}}$ with respect to the LECC limit $t = n(1 - \sqrt{1 - D})$, which is expressed as

$$\left. \frac{m_{\text{GS}}}{m_{\text{prop}}} \right|_{t=n(1-\sqrt{1-D})} = \frac{n(n-d)}{2(t-t_0)(n-t)} = \frac{\sqrt{1-D}}{(1-\sqrt{1-D})^2}.$$

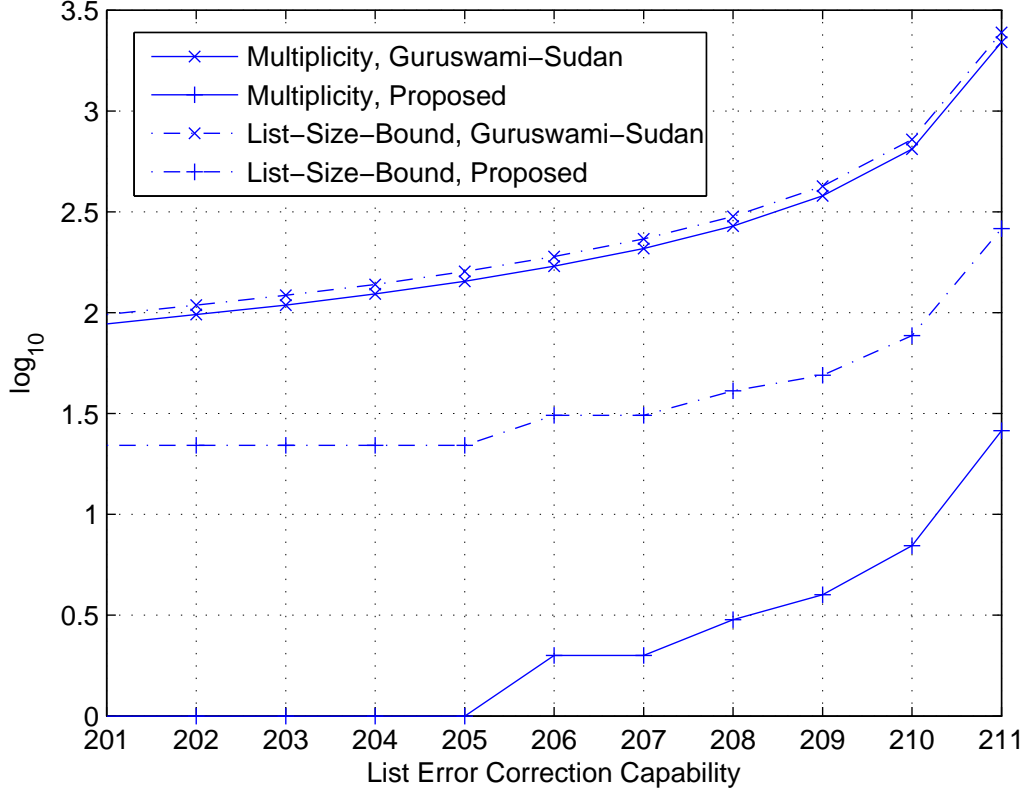


Figure 3: Multiplicity and list-size-bound as functions of list error correction capability for the (2047, 1647) Reed-Solomon code.

(Note the integer precision is not taken into account in the above equality.) It is plotted in Figure 4 as a function of normalized minimum distance D . Evidently, the proposed algorithm reduces the (required) multiplicity (to achieve the optimal LECC) by orders of magnitude.

F. An Alternative Perspective

In the above we have characterized the multiplicity m with respect to a fixed LECC t as in (31), and in particular, its asymptotics with respect to the maximum LECC t_{opt} as in (37). In this subsection, we characterize the LECC with respect to a fixed multiplicity and show that the derivative algorithm has quadratic complexity which is identical to that of the Berlekamp-Massey algorithm. To simplify our analysis we will treat P_y , and subsequently $\deg_{1,w}(\mathcal{Q}(x,y))$, as real numbers. Indeed, this treatment reveals many fundamental insights.

Note that the degrees of freedom is maximized to

$$\max\{N_{\text{free}}\} = \frac{(tm + t - t_0)^2}{4(t - t_0)}$$

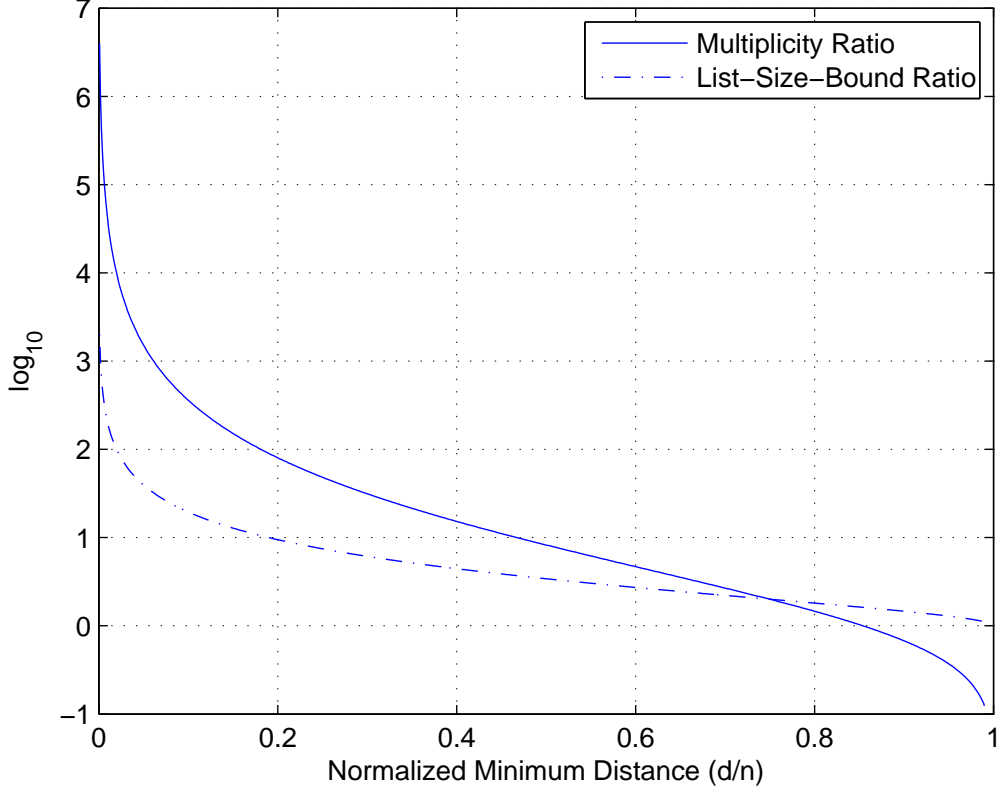


Figure 4: The ratio of multiplicity and list-size-bound of the Guruswami-Sudan algorithm to that of the proposed algorithm in achieving the limit of list error correction capability.

associated with $P_y = \frac{tm-t+t_0}{2(t-t_0)}$. Consequently, the constraint $\max\{N_{\text{free}}\} > N_{\text{cstr}}$ with respect to a fixed multiplicity m is reduced to

$$\frac{(tm - t + t_0)^2}{4(t - t_0)} > \frac{nm(m+1)}{2}, \quad (38)$$

which simplifies to

$$t^2(m+1)^2 - 2t(m+1)(t_0 + mn) + t_0^2 + 2t_0nm(m+1) > 0.$$

Solving, we obtain

$$t < \frac{1}{m+1}t_0 + \frac{m}{m+1} \left(n - \sqrt{n(n-d)} \right). \quad (39)$$

It indicates that, the practical LECC gain can be achieved with very small constant multiplicity, irregardless of the code length n , and particularly, half the LECC gain is achieved with multiplicity $m = 1$.

Recall that the above LECC is achieved with P_y set to

$$\begin{aligned} P_y &= \frac{tm - t + t_0}{2(t - t_0)} \\ &= \frac{D}{(1 - \sqrt{1 - D})^2} + \frac{m - 1}{1 - \sqrt{1 - D}}, \end{aligned} \quad (40)$$

where the second equality is obtained by substituting t with the limit in (39). It indicates that the list size is upper bounded by a constant for a fixed normalized minimum distance D , regardless of code length n . Figure 6 also reveals that the bound increases when the distance D decreases, although the algorithm itself becomes less and less powerful. On the other hand, the algorithmic complexity is quadratic following the step-by-step analysis of the preceding subsection, fundamentally attributed to a constant upper bound on the list size.

The above two-fold facts immediately reveals the following fundamental insights

Theorem 4 *For arbitrarily small $\epsilon > 0$, the list decoding up to the LECC*

$$t = \left\lfloor \epsilon \cdot t_0 + (1 - \epsilon) \cdot (n - \sqrt{n(n - d)}) \right\rfloor \quad (41)$$

can be achieved by the proposed algorithm with multiplicity $m = \lfloor \frac{1}{\epsilon} \rfloor$, whose complexity is quadratic in nature, $O(n^2)$.

Remarks: (i). Even if $m = 1$ the proposed algorithm universally outperforms the conventional hard-decision decoding, as shown in Figure 5, in contrast to the Sudan algorithm, where the improvement is observed only when the code rate $\frac{k}{n} < \frac{1}{3}$ [25]. (ii). The list size of candidate codewords is upper bounded by a constant arbitrarily close to the LECC limit $n - \sqrt{n(n - d)}$. However, the list size may be as large as superpolynomial slightly beyond the LECC limit, as explicitly constructed in [11].

IV. IMPROVED LIST DECODING ALGORITHM FOR BINARY BCH CODES

In this section we investigate the list decoding of the narrow-sense binary BCH codes. It is straightforward to apply the algorithm in the preceding section to the decoding of a binary BCH code and obtain the inherent LECC to be $n(1 - \sqrt{1 - D})$, where D denotes the normalized designed minimum distance. In the following we present an improved list decoding algorithm for the binary BCH codes that achieves the Johnson bound for the binary codes $\frac{n}{2}(1 - \sqrt{1 - 2D})$, which gives a general lower bound on the number of correctable errors using small lists for any code, as a function of the normalized minimum distance of the code [12].

The following lemma identifies a special feature of binary BCH codes. Its proof follows from Lemma 3 and the particularity of the Berlekamp algorithm.

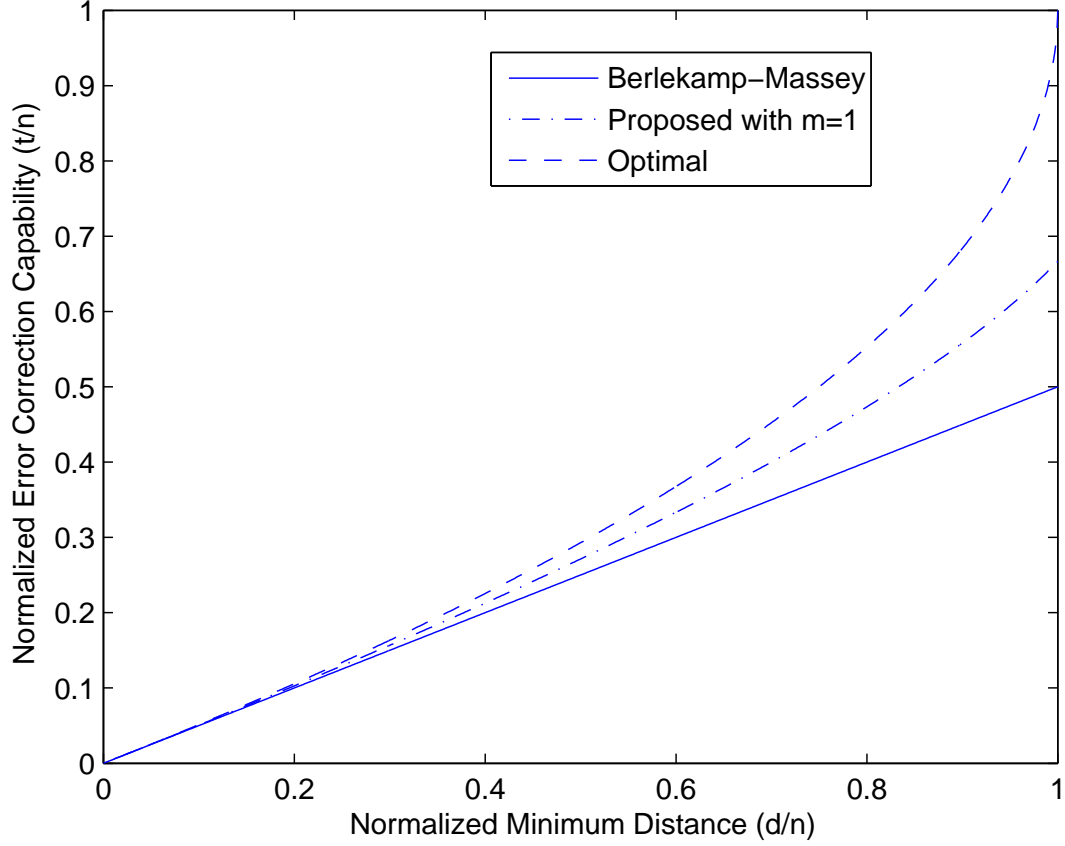


Figure 5: Normalized (list) error correction capability as a function of the normalized minimum distance on decoding Reed-Solomon codes.

Lemma 9 *Let $\Lambda^*(x)$ be the true error locator polynomial as defined in (2). Let $\Lambda(x)$ and $B(x)$ be the error locator and correction polynomials, respectively, obtained from the (re-formulated) Berlekamp algorithm. Then, $\Lambda^*(x)$ exhibits the form of*

$$\Lambda^*(x) = \Lambda(x) \cdot \lambda^*(x^2) + x^2 B(x) \cdot b^*(x^2), \quad (42)$$

where the polynomials $\lambda^*(x)$ and $b^*(x)$ exhibit the following properties:

- (i). $\lambda_0^* = 1$;
- (ii). if $b^*(x) = 0$, then $\lambda^*(x) = 1$;
- (iii). $\lambda^*(x)$ and $b^*(x)$ are coprime;
- (iv). $2 \deg(\lambda^*(x)) = \deg(\Lambda^*(x)) - L_\Lambda$ and $2 \deg(b^*(x)) \leq \deg(\Lambda^*(x)) - L_{x^2 B}$, or $2 \deg(\lambda^*(x)) \leq \deg(\Lambda^*(x)) - L_\Lambda$ and $2 \deg(b^*(x)) = \deg(\Lambda^*(x)) - L_{x^2 B}$;
- (v). if $\deg(\Lambda^*(x)) < d$, then $\lambda^*(x)$ and $b^*(x)$ are unique;

We proceed to incorporate the special form (42) into the rational interpolation process to optimize

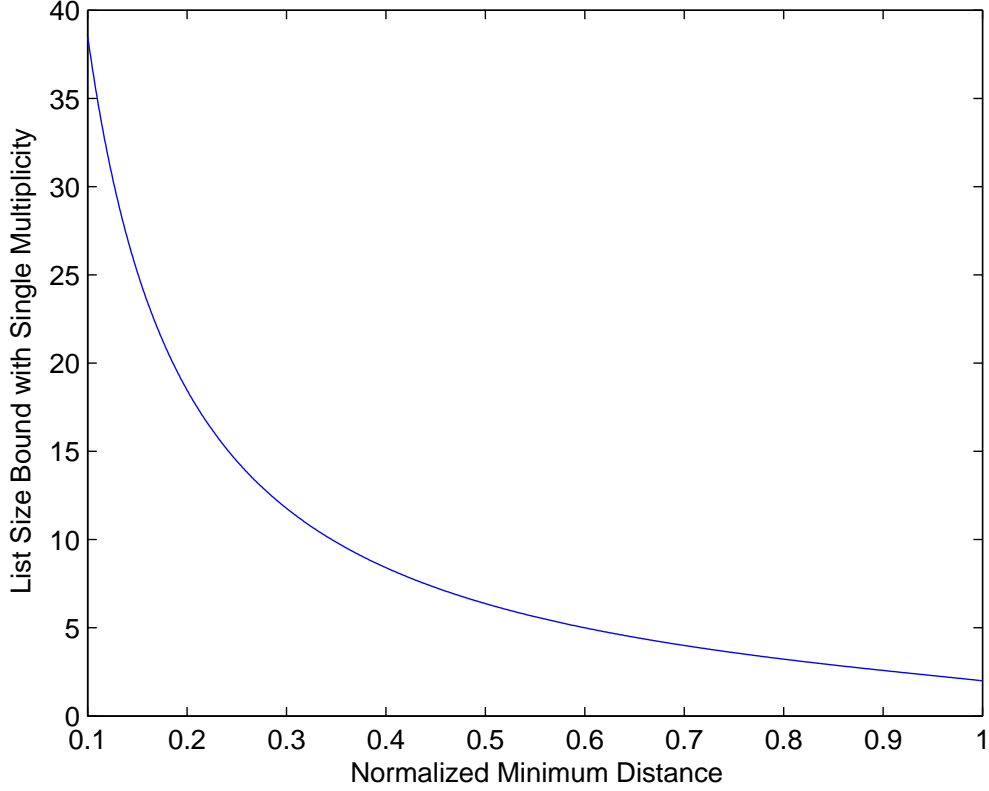


Figure 6: The bound on the list size with multiplicity $m = 1$.

the LECC. Define

$$y_i = -\frac{\Lambda(\alpha^{-i})}{\alpha^{-2i}B(\alpha^{-i})}, \quad i = 0, 1, 2, \dots, n-1. \quad (43)$$

Note we are interested in determining the form of $y\lambda(x^2) - b(x^2)$, so we naturally assign the weight of y to be

$$w = L_\Lambda - L_{x^2B}, \quad (44)$$

which is always odd since $L_\Lambda + L_{x^2B} = d$ is odd.

Lemma 10 *Let $\Lambda(x)$ and $B(x)$ be the error locator and correction polynomials, respectively, obtained from the (re-formulated) Berlekamp algorithm. Let $\mathcal{Q}(x, y)$ be a bivariate polynomial passing through all n points $\{(1, y_0), (\alpha^{-2}, y_1), (\alpha^{-4}, y_2), \dots, (\alpha^{-2(n-1)}, y_{n-1})\}$ (where y_i is defined in (43)), each with multiplicity m . If*

$$(t - L_\Lambda)P_y + \deg_{2,w}(\mathcal{Q}(x, y)) < 2tm, \quad (45)$$

where P_y denotes the power of y in $\mathcal{Q}(x, y)$ and $\deg_{2,w}(\mathcal{Q}(x, y))$ (where w is defined in (44)) denotes the $(2, w)$ -weighted degree of $\mathcal{Q}(x, y)$, then $\mathcal{Q}(x^2, y)$ contains all factors of the form $y\lambda(x^2) - b(x^2)$ which pass through t ($t \geq L_\Lambda$) points.

Proof: Let $y\lambda(x^2) - b(x^2)$ be a polynomial passing through t out of n points, $\{(1, y_0), (\alpha^{-1}, y_1), (\alpha^{-2}, y_2), \dots, (\alpha^{-(n-1)}, y_{n-1})\}$, and (α^{-i}, y_i) be one of points. Let

$$p(x) \triangleq \frac{b((x + \alpha^{-i})^2)}{\lambda((x + \alpha^{-i})^2)} - y_i = \frac{b(x^2 + \alpha^{-2i})}{\lambda(x^2 + \alpha^{-2i})} - y_i,$$

where the second equality is due to the field of characteristic of 2. Then $p(0) = 0$ and moreover $p(x)$ must be in the form of

$$p(x) = x^2 p'(x),$$

where $p'(x)$ contains no pole of zero. We next show that $(x - \alpha^{-i})^{2m}$ divides the following polynomial

$$g(x) \triangleq \lambda^{P_y}(x^2) \mathcal{Q}\left(x^2, \frac{b(x^2)}{\lambda(x^2)}\right)$$

To this end, consider

$$g'(x) \triangleq \lambda^{P_y}(x^2 + \alpha^{-2i}) \cdot \mathcal{Q}^{(i)}(x^2, p(x)),$$

where $\mathcal{Q}^{(i)}(x, y)$ is the shift of $\mathcal{Q}(x, y)$ to (α^{-2i}, y_i) . Consequently,

$$\begin{aligned} g(x) &= \lambda^{P_y}(x^2) \mathcal{Q}^{(i)}\left(x^2 + \alpha^{-2i}, \frac{b(x^2)}{\lambda(x^2)} - y_i\right) \\ &= \lambda^{P_y}(x^2) \mathcal{Q}^{(i)}(x^2 + \alpha^{-2i}, p(x - \alpha^{-i})) = g'(x - \alpha^{-i}). \end{aligned}$$

By construction, $\mathcal{Q}^{(i)}(x, y)$ has weighted degree less than m . Thus, plugging in $p(x) = x^2 p'(x)$, we conclude that $(x - \alpha^{-i})^{2m}$ divides $g(x)$. Therefore, $g(x)$ is a polynomial that contains at least $2tm$ roots, i.e., all roots of $\Lambda^*(x)$ each with multiplicity $2m$. The remaining proof trivially follows that of Lemma 5. $\square\square$

Note that the degrees of freedom in $\mathcal{Q}(x, y)$, given the degrees $\deg_{2,w}(\mathcal{Q}(x, y))$ and P_y , is

$$\begin{aligned} \sum_{i=0}^{P_y} \left(1 + \left\lfloor \frac{\deg_{2,w}(\mathcal{Q}(x, y)) - iw}{2} \right\rfloor\right) &\geq \sum_{i=0}^{P_y} \left(1 + \frac{\deg_{2,w}(\mathcal{Q}(x, y)) - iw - 1}{2}\right) \\ &= \frac{(\deg_{2,w}(\mathcal{Q}(x, y)) + 1 - P_y w/2)(P_y + 1)}{2}. \end{aligned}$$

Define

$$t_0 \triangleq L_\Lambda - w/2 = \frac{L_\Lambda + L_{x^2 B}}{2} = \frac{d}{2}.$$

Note that the above definition is consistent with the case of Reed-Solomon codes.

We next maximize the lower bound of N_{free} , $\frac{(\deg_{2,w}(\mathcal{Q}(x, y)) + 1 - P_y w/2)(P_y + 1)}{2}$, subject to fixed number of errors t , fixed multiplicity m and the zero constraint (45)

$$\begin{aligned} \frac{(\deg_{2,w}(\mathcal{Q}(x, y)) + 1 - P_y w/2)(P_y + 1)}{2} &\leq \frac{1}{2}(2tm - P_y(t - t_0))(P_y + 1) \\ &= -\frac{t - t_0}{2}P_y^2 + \frac{2tm - t + t_0}{2}P_y + tm \\ &= -\frac{t - t_0}{2}\left(P_y - \frac{2tm - t + t_0}{2(t - t_0)}\right)^2 + \frac{(2tm + t - t_0)^2}{8(t - t_0)} \quad (46) \end{aligned}$$

where “=” in the “ \leq ” is achieved if and only if

$$\deg_{2,w}(\mathcal{Q}^*(x, y)) = 2tm - 1 - P_y(t - L_\Lambda). \quad (47)$$

We note that the maximum degrees of freedom is achieved by choosing P_y^* to be the closest integer to $\frac{2tm-t+t_0}{2t-2t_0}$, i.e.,

$$P_y^* = \left\lfloor \frac{2tm - t + t_0}{2t - 2t_0} + 0.5 \right\rfloor = \left\lfloor \frac{tm}{t - t_0} \right\rfloor. \quad (48)$$

Therefore, the maximum degrees of freedom has the following lower bound

$$-\frac{t-t_0}{8} + \frac{(2tm+t-t_0)^2}{8(t-t_0)} = \frac{t^2m^2 + tm(t-t_0)}{2(t-t_0)}. \quad (49)$$

Hence, to solve for the linear equation system, it suffices to enforce

$$\frac{t^2m^2 + tm(t-t_0)}{2(t-t_0)} > \frac{nm(m+1)}{2}, \quad (50)$$

which is equivalent to

$$m^2(t^2 - n(t-t_0)) - m(t-t_0)(n-t) > 0.$$

The above condition holds true for sufficiently large m if and only if

$$t^2 - n(t-t_0) > 0,$$

which indicates the following limit of LECC

$$t < \frac{n - \sqrt{n(n-4t_0)}}{2} = \frac{n - \sqrt{n(n-2d)}}{2}. \quad (51)$$

It is always superior to the inherent LECC $n - \sqrt{n(n-d)}$, following the fact

$$\frac{n - \sqrt{n(n-2d)}}{2} - (n - \sqrt{n(n-d)}) = \frac{t_0\sqrt{n}}{\sqrt{n-d} + \sqrt{n-2d}} - \frac{t_0\sqrt{n}}{\sqrt{n} + \sqrt{n-d}} > 0.$$

When (51) is satisfied, we may choose the multiplicity m to be

$$m^* = \left\lfloor \frac{(t-t_0)(n-t)}{t^2 - n(t-t_0)} + 1 \right\rfloor = \left\lfloor \frac{tt_0}{t^2 - n(t-t_0)} \right\rfloor. \quad (52)$$

The following presents the “lossless” maximization of the degrees of freedom conditioned on the fixed t and m . It can be easily verified that the degrees of freedom is bounded by

$$\begin{aligned} \frac{(\deg_{2,w}(\mathcal{Q}(x, y)) + 1 - P_y w/2)(P_y + 1)}{2} + \frac{P_y}{4} + \frac{1}{2} &\geq \sum_{i=0}^{P_y} \left(1 + \left\lfloor \frac{\deg_{2,w}(\mathcal{Q}(x, y)) - iw}{2} \right\rfloor \right) \\ &\geq \frac{(\deg_{2,w}(\mathcal{Q}(x, y)) + 1 - P_y w/2)(P_y + 1)}{2} + \frac{P_y}{4}. \end{aligned}$$

Due to the integer constraint, it suffices to treat and optimize the degrees of freedom as follows

$$\begin{aligned}
N_{\text{free}} &= \frac{(\deg_{2,w}(\mathcal{Q}(x,y)) + 1 - P_y w/2)(P_y + 1)}{2} + \frac{P_y}{4} \\
&\leq \frac{1}{2}(2tm - P_y(t - t_0))(P_y + 1) + \frac{P_y}{4} \\
&= -\frac{t - t_0}{2} \left(P_y - \frac{2tm - t + t_0 + 0.5}{2(t - t_0)} \right)^2 + \frac{(2tm - t + t_0 + 0.5)^2}{8(t - t_0)} + tm
\end{aligned} \tag{53}$$

where “=” is achieved by choosing $\deg_{2,w}(\mathcal{Q}(x,y))$ as in (47). Evidently, the optimal choice of P_y is as follows,

$$P_y^* = \left\lfloor \frac{2tm - t + t_0 + 0.5}{2(t - t_0)} + 0.5 \right\rfloor = \left\lfloor \frac{tm + 0.25}{t - t_0} \right\rfloor, \tag{54}$$

which is slightly different from (48). Consequently, the optimal multiplicity m is the minimum one that complies with the following constraint

$$-\frac{t - t_0}{2} \left(\left\lfloor \frac{tm + 0.25}{t - t_0} \right\rfloor - \frac{2tm - t + t_0 + 0.5}{2(t - t_0)} \right)^2 + \frac{(2tm - t + t_0 + 0.5)^2}{8(t - t_0)} + tm > \frac{nm(m + 1)}{2}. \tag{55}$$

We are ready to present the complete list decoding algorithm for binary BCH codes as follows.

List Decoding Algorithm for Binary BCH Codes

0. Initialization: Input code length n , minimum distance d , and LECC t satisfying $t < \frac{1}{2} \left(n - \sqrt{n(n - 2d)} \right)$. Initialize the multiplicity m based on (52), then greedily optimize it subject to the constraint (55), subsequently choose the y -degree P_y as in (54).
1. Input the received word and compute syndromes.
2. Apply the Berlekamp algorithm to determine $\Lambda(x)$ and $B(x)$. If $L_\Lambda > t$ then declare a decoding failure.
3. Perform error correction and evaluate $y_i = -\frac{\Lambda(\alpha^{-i})}{\alpha^{-2i}B(\alpha^{-i})}$, $i = 0, 1, 2, \dots, n - 1$.
4. If $L_{x^2B} > t$, then return the corresponding unique codeword when $\Lambda(x)$ has precisely L_Λ valid roots, otherwise declare a decoding failure.
5. Apply the rational interpolation procedure to compute a $(2, L_\Lambda - L_{x^2B})$ -weighted-degree polynomial $\mathcal{Q}(x,y)$ that passes through $\{(\alpha^{-2i}, y_i)\}_{i=0}^{n-1}$, each with multiplicity m .
6. Apply the rational factorization process to obtain finite-length power series $s(x) \pmod{x^{L_s}}$ of the rational functions $\frac{b(x)}{\lambda(x)}$.
7. For each finite-length power series $s(x) \pmod{x^{L_s}}$, do:
 - Apply the Berlekamp-Massey algorithm to determine $\lambda(x)$.

- Compute $b(x) = s(x)\lambda(x) \pmod{x^{t-L_{x^2B}+1}}$.
- Construct $\Lambda'(x) = \lambda(x^2) \cdot \Lambda(x) + b(x^2) \cdot x^2B(x)$.
- Determine the distinct roots of $\Lambda'(x)$ within $\{\alpha^{-i}\}_{i=0}^{n-1}$.

8. Return the list of codewords which are within distance t from the received word.

We summarize the above discussions into the following theorem

Theorem 5 *For a given (n, k, d) BCH code, let the LECC t satisfies (51) and the multiplicity m be chosen as in (52), then the proposed list decoding algorithm produces all codewords up to distance t from a received word.*

Remarks: The LECC limits of the proposed algorithm and the Guruswami-Sudan algorithm are illustrated in Figure 1. The proposed performance matches the Johnson bound for binary codes [12]. The proposed performance also demonstrates the tightness of the improved bound of the list size in [7].

Following the step-by-step complexity analysis in the preceding section, we characterize the complexity of the proposed algorithm as follows

Theorem 6 *Given that the field cardinality is at most 2^n , the proposed list decoding algorithm for (n, k, d) binary BCH codes exhibits the computational complexity in terms of field operations*

$$O\left(n^2(\sqrt{n} - \sqrt{n-2d})^8\right) \quad (56)$$

to achieve the maximum LECC $t_{opt} = \lceil \frac{n - \sqrt{n(n-2d)}}{2} - 1 \rceil$.

We proceed to characterize the proposed list decoding algorithm with respect to a fixed multiplicity m . For the conciseness of analysis, we treat the degrees of freedom to be $\frac{1}{2}(\deg_{2,w}(\mathcal{Q}(x, y)) + 1 - P_y w/2)(P_y + 1)$, and P_y as well as $\deg_{2,w}(\mathcal{Q}(x, y))$ to be real numbers. Following (46), the maximum degrees of freedom is

$$\max\{N_{\text{free}}\} = \frac{(2tm + t - t_0)^2}{8(t - t_0)}$$

which is achieved by choosing $P_y = \frac{2tm - t + t_0}{2(t - t_0)}$. Subsequently, the constraint of more degrees of freedom than the number linear constraints is reduced to

$$\frac{(2tm + t - t_0)^2}{8(t - t_0)} > \frac{nm(m + 1)}{2}$$

which is re-organized in decreasing order of t

$$t^2(2m + 1)^2 - 2t((2m + 1)t_0 + 2m(m + 1)n) + t_0^2 + 4t_0(m + 1)mn > 0.$$

Solving, we obtain the following LECC limit with respect to the fixed m

$$t < \frac{(2m + 1)t_0 + 2m(m + 1)n - 2m\sqrt{n^2(m + 1)^2 - nd(m + 1)(2m + 1)}}{(2m + 1)^2}. \quad (57)$$

It can be shown that

$$\begin{aligned} & \frac{(2m+1)t_0 + 2m(m+1)n - 2m\sqrt{n^2(m+1)^2 - nd(m+1)(2m+1)}}{(2m+1)^2} \\ & > \frac{1}{2m+1} \cdot t_0 + \frac{2m}{2m+1} \cdot \frac{n - \sqrt{n(n-2d)}}{2}. \end{aligned} \quad (58)$$

On the other hand, we have

$$\begin{aligned} P_y &= \frac{2tm - t + t_0}{2(t - t_0)} \\ &\leq \frac{(2m-1) \left(\frac{1}{2m+1}t_0 + \frac{2m}{2m+1} \frac{n - \sqrt{n(n-2d)}}{2} \right) + t_0}{\frac{2}{2m+1}t_0 + \frac{4m}{2m+1} \frac{n - \sqrt{n(n-2d)}}{2} - d} \\ &= \frac{2D}{(1 - \sqrt{1-2D})^2} + \frac{2m-1}{1 - \sqrt{1-2D}}, \end{aligned} \quad (59)$$

which indicates that the list size is upper bounded by a constant with respect to a given normalized minimum distance D . We characterize the above discussions into the following theorem.

Theorem 7 *For arbitrarily small $\epsilon > 0$, the list decoding of the (n, k, d) binary BCH code up to the LECC*

$$t = \left\lfloor \epsilon \cdot t_0 + (1 - \epsilon) \cdot \frac{n - \sqrt{n(n-2d)}}{2} \right\rfloor \quad (60)$$

can be achieved by the proposed algorithm with multiplicity $m = \lfloor \frac{1}{2\epsilon} \rfloor$, whose complexity is quadratic in nature, $O(n^2)$.

Example 6. Consider the (63, 18, 21) BCH code. Its conventional error correction capability is 10. The achievable LECC of the Guruswami-Sudan algorithm is $\lceil 63 - \sqrt{63 \times 42} - 1 \rceil = 11$, whereas that of the proposed algorithm is $\lceil \frac{63 - \sqrt{63 \times 21}}{2} - 1 \rceil = 13$. To correct up to $t = 13$ errors, the multiplicity and the degree of y are set as follows. Let the multiplicity be $m = \lfloor \frac{13 \times 10.5}{13^2 - 63 \times (13 - 11.5)} \rfloor = 11$, following (52), which turns out to be optimal. Let the y degree be $P_y = \lfloor \frac{13 \times 11 + 0.25}{13 - 10.5} \rfloor = 57$, following (48), which also turns out to be optimal. We next walk through a simulation example to illustrate the algorithmic procedure. Let the prototype codeword be

$$\begin{aligned} \mathbf{c} = [& 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, \\ & 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0] \end{aligned}$$

and the received word be

$$\begin{aligned} \mathbf{r} = [& 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, \\ & 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0] \end{aligned}$$

which has 13 erroneous bits. The true error locator polynomial is

$$\begin{aligned}\Lambda^*(x) &= (1 - \alpha^4 x)(1 - \alpha^7 x)(1 - \alpha^{16} x)(1 - \alpha^{17} x)(1 - \alpha^{19} x)(1 - \alpha^{20} x)(1 - \alpha^{33} x)(1 - \alpha^{45} x) \\ &\quad (1 - \alpha^{46} x)(1 - \alpha^{47} x)(1 - \alpha^{50} x)(1 - \alpha^{53} x)(1 - \alpha^{60} x) \\ &= 1 + \alpha^{29} x + \alpha^{23} x^2 + \alpha^{37} x^3 + \alpha^{61} x^4 + \alpha^{42} x^5 + \alpha^{28} x^6 + \alpha^{15} x^7 + \alpha^{35} x^8 + \alpha^{40} x^9 \\ &\quad + \alpha^{35} x^{10} + \alpha x^{11} + \alpha^{12} x^{12} + \alpha^{39} x^{13}.\end{aligned}$$

The Berlekamp algorithm outputs the following pair of error locator and correction polynomials

$$\begin{aligned}\Lambda(x) &= 1 + \alpha^{29} x + \alpha^{16} x^2 + \alpha^5 x^3 + \alpha^2 x^4 + \alpha^{47} x^5 + \alpha^6 x^6 + \alpha^{49} x^7 + \alpha^{27} x^8 + \alpha^{19} x^9 + \alpha^{34} x^{10} \\ B(x) &= \alpha^2 + \alpha^{31} x + \alpha^{23} x^2 + \alpha^{50} x^3 + \alpha^{51} x^4 + \alpha^{42} x^5 + \alpha^{53} x^6 + \alpha^{53} x^7 + \alpha^{32} x^8 + \alpha^{15} x^9.\end{aligned}$$

In this case, the weight of y is set to $w = L_\Lambda - L_{x^2 B} = -1$. The following lists the $n = 63$ interpolation points $\{(\alpha^{-2i}, \frac{\Lambda(\alpha^{-i})}{\alpha^{-2i} B(\alpha^{-i})})\}_{i=0}^{62}$:

$(1, \alpha^{57})$	$(\alpha^{-2}, \alpha^{19})$	$(\alpha^{-4}, \alpha^{54})$	$(\alpha^{-6}, \alpha^{35})$	$(\alpha^{-8}, \alpha^{13})$	$(\alpha^{-10}, \alpha^{38})$	$(\alpha^{-12}, \alpha^{48})$	$(\alpha^{-14}, \alpha^{46})$
$(\alpha^{-16}, \alpha^{27})$	$(\alpha^{-18}, \alpha^{49})$	$(\alpha^{-20}, \alpha^{52})$	$(\alpha^{-22}, \alpha^{58})$	$(\alpha^{-24}, \alpha^{14})$	$(\alpha^{-26}, \alpha^{34})$	$(\alpha^{-28}, \alpha^{45})$	$(\alpha^{-30}, \alpha^{31})$
(α^{-32}, α^5)	$(\alpha^{-34}, \alpha^{44})$	$(\alpha^{-36}, \alpha^{45})$	(α^{-38}, α^3)	$(\alpha^{-40}, \alpha^{15})$	$(\alpha^{-42}, \alpha^{61})$	$(\alpha^{-44}, \alpha^{62})$	$(\alpha^{-46}, \alpha^{30})$
$(\alpha^{-48}, \alpha^{31})$	(α^{-50}, ∞)	$(\alpha^{-52}, \alpha^{49})$	(α^{-54}, α^8)	$(\alpha^{-56}, \alpha^{35})$	$(\alpha^{-58}, \alpha^{41})$	$(\alpha^{-60}, \alpha^{57})$	(α^{-62}, α^4)
$(\alpha^{-1}, \alpha^{45})$	$(\alpha^{-3}, \alpha^{17})$	$(\alpha^{-5}, \alpha^{34})$	(α^{-7}, α^5)	$(\alpha^{-9}, \alpha^{16})$	$(\alpha^{-11}, \alpha^{62})$	(α^{-13}, α^7)	$(\alpha^{-15}, \alpha^{47})$
$(\alpha^{-17}, \alpha^{26})$	(α^{-19}, α^8)	(α^{-21}, ∞)	$(\alpha^{-23}, \alpha^{12})$	$(\alpha^{-25}, \alpha^{54})$	$(\alpha^{-27}, \alpha^{38})$	$(\alpha^{-29}, \alpha^{11})$	$(\alpha^{-31}, \alpha^{25})$
$(\alpha^{-33}, \alpha^{62})$	(α^{-35}, α^8)	$(\alpha^{-37}, \alpha^{59})$	(α^{-39}, α^8)	$(\alpha^{-41}, \alpha^{30})$	$(\alpha^{-43}, \alpha^{53})$	$(\alpha^{-45}, \alpha^{12})$	$(\alpha^{-47}, \alpha^{47})$
$(\alpha^{-49}, \alpha^{44})$	$(\alpha^{-51}, \alpha^{15})$	$(\alpha^{-53}, \alpha^{49})$	$(\alpha^{-55}, \alpha^{26})$	$(\alpha^{-57}, \alpha^{57})$	(α^{-59}, α^5)	$(\alpha^{-61}, \alpha^{48})$	

Rational factorization returns three candidate rational functions.

(1). $\lambda(x) = 1 + \alpha^{15} x$ and $b(x) = \alpha^{31} + \alpha^{24} x$. The candidate error locator polynomial is constructed as

$$\begin{aligned}\Lambda'(x) &= \lambda(x^2) \cdot \Lambda(x) + b(x^2) \cdot x^2 B(x) \\ &= 1 + \alpha^{29} x + \alpha^{59} x^2 + \alpha^{38} x^3 + \alpha^2 x^4 + \alpha^{46} x^5 + \alpha^{52} x^6 + \alpha^{62} x^7 + \alpha^{39} x^8 + \alpha^{14} x^9 + \alpha^5 x^{10} \\ &\quad + \alpha^{59} x^{11} + \alpha^6 x^{12} + \alpha^{37} x^{13} \\ &= (1 - \alpha^4 x)(1 - \alpha^7 x)(1 - \alpha^{16} x)(1 - \alpha^{17} x)(1 - \alpha^{19} x)(1 - \alpha^{20} x)(1 - \alpha^{33} x)(1 - \alpha^{45} x)(1 - \alpha^{46} x) \\ &\quad (1 - \alpha^{47} x)(1 - \alpha^{50} x)(1 - \alpha^{53} x)(1 - \alpha^{60} x)\end{aligned}$$

which properly retrieves the prototype codeword.

(2). $\lambda(x) = 1 + \alpha^{52} x$ and $b(x) = \alpha^{55}$. The candidate error locator polynomial is constructed as

$$\begin{aligned}\Lambda'(x) &= 1 + \alpha^{29} x + \alpha^{29} x^2 + \alpha^7 x^3 + \alpha^8 x^4 + \alpha^{11} x^5 + \alpha^{11} x^6 + \alpha^{15} x^7 + \alpha^{15} x^8 + \alpha^{28} x^9 \\ &\quad + \alpha^{17} x^{10} + \alpha^{13} x^{11} + \alpha^{23} x^{12} \\ &= (1 - \alpha^{10} x)(1 - \alpha^{12} x)(1 - \alpha^{19} x)(1 - \alpha^{24} x)(1 - \alpha^{28} x)(1 - \alpha^{31} x)(1 - \alpha^{33} x)(1 - \alpha^{36} x) \\ &\quad (1 - \alpha^{48} x)(1 - \alpha^{49} x)(1 - \alpha^{52} x)(1 - \alpha^{59} x)\end{aligned}$$

which yields an alternative candidate codeword with only 12-bit difference from the received word.

$$\mathbf{c} = [1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, \\ 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0].$$

(3). $\lambda(x) = 1 + \alpha^{32}x + \alpha^{19}x^2$ and $b(x) = \alpha^{45} + \alpha^{28}x$. The corresponding candidate error locator polynomial is

$$\Lambda'(x) = 1 + \alpha^{29}x + \alpha^{59}x^2 + \alpha^{38}x^3 + \alpha^2x^4 + \alpha^{46}x^5 + \alpha^{52}x^6 + \alpha^{62}x^7 + \alpha^{39}x^8 + \alpha^{14}x^9 \\ + \alpha^5x^{10} + \alpha^{59}x^{11} + \alpha^6x^{12} + \alpha^{37}x^{13} + \alpha^{53}x^{14}$$

which does not contain precisely 14 distinct nonzero roots in $\text{GF}(64)$ and thus is spurious. $\square\square$

V. CONCLUDING REMARKS

Although the proposed list decoding algorithms are presented in the context of the list decoding of Reed-Solomon and BCH codes, their core is a rational curve-fitting algorithm, which may be viewed as an extension of the polynomial curve-fitting algorithm proposed in [9].

Following the strategy of soft reliability transformation in the Koetter-Vardy algorithm [16], the proposed hard-decision list decoding algorithm can be extended in a straightforward manner to algebraic soft-decision decoding using weighted multiplicity array (in one-dimension), where weight is proportional to symbol error probability for (generalized) Reed-Solomon codes, or bit error probability for binary BCH codes. However, we may only obtain a one-dimensional multiplicity array due to the preprocessing of the Berlekamp-Massey algorithm, instead of a two-dimensional multiplicity matrix as in the Koetter-Vardy algorithm [16].

It is shown that the number of codewords is less than n for list decoding up to the Johnson bound for binary codes [1, 7, 10], whereas we have disclosed that the list size for decoding up to the Johnson bound is bounded by $O(t_{\text{opt}})$ for the particular class (nonbinary) Reed-Solomon and binary BCH codes. We thus conjecture that n is the universal bound for list decoding up to the Johnson bound for any code.

Our developments are based on the notion of one-to-one correspondence along the two different definitions/interpretations of Reed-Solomon codes. Following the notion as well as the fact that BCH codes are subfield subcodes of Reed-Solomon codes [24], we conjecture that the Guruswami-Sudan algorithm can be modified in a way to achieve the Johnson bound for binary BCH codes.

Acknowledgements

The author would like to thank Dr. Shih-Ming Shih, Prof. Paul Siegel, Prof. Ralf Koetter, Prof. Jorn Justesen, Prof. Sergey Bezzateev, Prof. Ronny Roth, and specially Prof. Vladimir Sidorenko, for many constructive comments on improving the quality as well as the presentation of the manuscript.

References

- [1] E. Agrell, A. Vardy, and K. Zeger, “Upper bounds for constant-weight codes,” *IEEE Trans. Inform. Theory*, vol. 46, pp. 2373–2395, Nov. 2000.
- [2] E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
- [3] ———, “Bounded distance +1 soft-decision Reed-Solomon decoding,” *IEEE Trans. Inform. Theory*, vol. 42, pp. 704–720, May 1996.
- [4] R. E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University Press, Cambridge, UK, 2003.
- [5] I. M. Duursma and R. Koetter, “Error-locating pairs for cyclic codes,” *IEEE Trans. Inform. Theory*, vol. 40, pp. 1108–1121, July 1994.
- [6] S. Egorov, G. Markarian, and K. Pickavance, “A modified Blahut algorithm for decoding Reed-Solomon codes beyond half the minimum distance,” *IEEE Trans. Communications*, vol. 52, pp. 2052–2056, Dec. 2004.
- [7] P. Elias, “Error-correcting codes for list decoding,” *IEEE Trans. Inform. Theory*, vol. 37, pp. 5–12, Jan. 1991.
- [8] G. D. Forney, Jr., “On decoding BCH codes,” *IEEE Trans. Inform. Theory*, vol. 11, pp. 549–557, Oct. 1965.
- [9] V. Guruswami and M. Sudan, “Improved decoding of Reed-Solomon codes and algebraic-geometry codes,” *IEEE Trans. Inform. Theory*, vol. 45, no. 6, pp. 1757–1767, Sept. 1999.
- [10] ———, “Extensions to the Johnson bound,” unpublished manuscript, Dec. 2000. Available at <http://people.csail.mit.edu/madhu/papers.html>.
- [11] V. Guruswami and A. Rudra, “Limits to list decoding Reed-Solomon codes,” *IEEE Trans. Inform. Theory*, vol. 52, pp. 3642–3649, Aug. 2006.
- [12] S. M. Johnson, “A new upper bound for error-correcting codes,” *IEEE Trans. Inform. Theory*, vol. 8, pp. 203–207, Apr. 1962.
- [13] J. Justesen and T. Hoholdt, “Bounds on list decoding of MDS codes,” *IEEE Trans. Inform. Theory*, vol. 47, pp. 1604–1609, May 2001.
- [14] N. Kamiya, “On algebraic soft-decision decoding algorithms for BCH codes,” *IEEE Trans. Inform. Theory*, vol. 47, pp. 45–58, Jan. 2001.

- [15] R. Koetter, “Fast generalized minimum-distance decoding of algebraic-geometry and Reed-Solomon codes,” *IEEE Trans. Inform. Theory*, vol. 42, pp. 721–737, May 1996.
- [16] R. Koetter and A. Vardy, “Algebraic soft-decision decoding of Reed-Solomon codes,” *IEEE Trans. Inform. Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.
- [17] R. Lidl and H. Niederreiter, *Finite Fields*, Reading, MA: Addison-Wesley, 1983.
- [18] J. Ma, P. Trifonov, and A. Vardy, “Divide-and-conquer interpolation for list decoding of Reed-Solomon codes,” *Proc. IEEE Inform. Theory Workshop*, Chicago, p. 386, 2004.
- [19] J. L. Massey, “Shift-register synthesis and BCH decoding,” *IEEE Trans. Inform. Theory*, vol. 15, pp. 122–127, Jan. 1969.
- [20] R. J. McEliece, “The Guruswami-Sudan decoding algorithm for Reed-Solomon codes,” JPL progress report: 42–153, Apr. 2003. Available at http://tmo.jpl.nasa.gov/progress_report/42-153/153F.pdf
- [21] R. Roth and G. Ruckenstein, “Efficient decoding of Reed-Solomon codes beyond half the minimum distance,” *IEEE Trans. Inform. Theory*, vol. 46, no. 1, pp. 246–257, Jan. 2000.
- [22] G. Ruckenstein and R. M. Roth, “Bounds on the list-decoding radius of Reed-Solomon codes,” *SIAM J. Discrete Math*, vol. 17, no. 2, pp. 171–195, 2003.
- [23] G. Schmidt, V. Sidorenko, and M. Bossert, “Decoding Reed-Solomon codes beyond half the minimum distance using shift-register synthesis,” *Proc. IEEE Int. Symp. Inform. Theory*, Seattle, pp. 459–463, 2006.
- [24] H. Stichtenoth, *Algebraic Function Fields and Codes*, Springer-Verlag, Berlin, Germany, 1993.
- [25] M. Sudan, “Decoding of Reed-Solomon codes beyond the error-correction bound,” *J. Complexity*, vol. 13, no. 1, pp. 180–193, 1997.
- [26] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, “A method for solving key equation for decoding Goppa codes,” *Inform. and Control*, vol. 27, pp. 87–99, Feb. 1975.
- [27] L. R. Welch and E. R. Berlekamp, “Error correction for algebraic block codes,” U.S. patent no. 4,633,470, Dec. 30, 1986.
- [28] X.-W. Wu and P. H. Siegel, “Efficient root-finding algorithm with application to list decoding of algebraic-geometric codes,” *IEEE Trans. Inform. Theory*, vol. 47, pp. 2579–2587, Sept. 2001.