TABLE III
WEIGHT DISTRIBUTION (A_i) OF THE CODE D_1

weight i	0	13	14	15	16	17	18
A_i	1	328	852	400	850	7484	13296
weight i	19	20	21	22	23	24	25
A_i	4296	45604	57996	13776	14924	73464	67936
weight i	26	27	28	29	30	31	32
A_i	11312	8888	32944	21836	2896	1629	3916
weight i	33	34	35	36	37	38	39
A_i	1904	72	28	100	20	16	4

linear code C_0 is shown in Table II. The code C_1 has been constructed by a search procedure similar to that of [2]. However, in each step of the procedure, binary nonlinear codes are constructed from binary linear codes by the Y2 construction [3, pp. 592–593], a simple generalization of the Nordstrom–Robinson code construction [3, pp. 73–74].

II. A BINARY $(49, 393216, 13)$ CODE

By Litsyn [4], the value $A(49, 13)$ is shown to be at least $2^{18} = 262144$. We improve this bound by presenting a new binary $(49, 393216, 13)$ code.

The new binary $(49, 393216, 13)$ code D_1 is given by the union of a linear $[49, 16, 15]$ code D_0 with six of its cosets. The linear code D_0 is generated by the canonical matrix at the bottom of the previous page.

The cosets are given by the coset leaders of minimal weight (w_0 is the zero vector),

$$w_1 = 1000000000001000110011110101001100000000001000000$$
$$w_2 = 1000100001000101000110000110001000000000000101010$$
$$w_3 = 0000000000000100010101101011011100000000000010001$$
$$w_4 = 1011000010011000010001000000100100000000001011000$$
$$w_5 = 1011010010001000001100100001010000000000000010001$$

The code D_1 has the dual distance $d' = 5$ and its weight distribution is given in Table III. The code D_1 has been constructed by the same search procedure as the code C_1.

REFERENCES

[1] E. Agrell, A. Vardy, and K. Zeger, "A table of upper bounds for binary codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 3004–3006, Nov. 2001.

[2] T. Verhoeff, "An updated table of minimum-distance bounds for binary linear codes," *IEEE Trans. Inf. Theory*, vol. IT-33, pp. 665–680, Sept. 1987.

[3] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1998.

[4] S. Litsyn *et al.*, "Table of best known binary codes," in *Handbook of Coding Theory*, V. Pless *et al.*, Eds. Amsterdam, The Netherlands: North-Holland, 1998.

[5] K. Elssel, "Konstruktion nichtlinearer Binärcodes aus linearen Binärcodes und deren Anwendung in der Kryptographie," Project Work, TU Hamburg-Harburg, Hamburg, Germany, 2002.

[6] N. S. Babu and K. H. Zimmermann, "A short introduction to quaternary codes," TU Hamburg-Harburg, Hamburg, Germany, Res. Rep. 99.3, 1999.

A Root-Finding Algorithm for List Decoding of Reed–Muller Codes

Xin-Wen Wu, *Member, IEEE*, Margreta Kuijper, *Member, IEEE*, and Parampalli Udaya, *Member, IEEE*

***Abstract*—Let $\boldsymbol{F}_q[\boldsymbol{X_1}, \ldots, \boldsymbol{X_m}]$ denote the set of polynomials over $\boldsymbol{F}_q$ in $\boldsymbol{m}$ variables, and $\boldsymbol{F}_q[\boldsymbol{X_1}, \ldots, \boldsymbol{X_m}]_{\leq u}$ denote the subset that consists of the polynomials of total degree at most $\boldsymbol{u}$. Let $\boldsymbol{H(T)}$ be a nontrivial polynomial in $\boldsymbol{T}$ with coefficients in $\boldsymbol{F}_q[\boldsymbol{X_1}, \ldots, \boldsymbol{X_m}]$. A crucial step in interpolation-based list decoding of $\boldsymbol{q}$-ary Reed–Muller (RM) codes is finding the roots of $\boldsymbol{H(T)}$ in $\boldsymbol{F}_q[\boldsymbol{X_1}, \ldots, \boldsymbol{X_m}]_{\leq u}$. In this correspondence, we present an efficient root-finding algorithm, which finds all the roots of $\boldsymbol{H(T)}$ in $\boldsymbol{F}_q[\boldsymbol{X_1}, \ldots, \boldsymbol{X_m}]_{\leq u}$. The algorithm can be used to speed up the list decoding of RM codes.**

***Index Terms*—$\boldsymbol{q}$-ary Reed–Muller (RM) codes, list decoding, Reed–Solomon (RS) codes, root-finding algorithm.**

I. INTRODUCTION

List decoding [5], [7], [12], [13] is an important decoding method which makes it possible to recover information in the presence of errors beyond the traditional error-correction bound. List decoding is also used in Koetter–Vardy's algebraic soft-decision decoding of Reed–Solomon (RS) codes [8], as well as Lee-metric decoding of RS codes [15]. A crucial step in list decoding is finding the roots of a polynomial with coefficients being polynomials or rational functions over a finite field. These roots are then used to reconstruct the codewords that are candidates for the transmitted codeword. Efficient root-finding algorithms for list decoding of RS and algebraic-geometric (AG) codes were given in [1], [3], [6], [11], [16]. In this correspondence, building on the ideas in [14], we present an efficient root-finding algorithm for list decoding of Reed–Muller (RM) codes.

RM codes are a generalization of RS codes. Denote by $\boldsymbol{F}_q$ the finite field of q elements. Let $\boldsymbol{F}_q[X_1, \ldots, X_m]$ be the ring of polynomials in m variables with coefficients in $\boldsymbol{F}_q$. The degree of a monomial $X_1^{a_1} \cdots X_m^{a_m}$ is defined as $\sum_{i=1}^{m} a_i$, and the degree of a polynomial f, denoted by $\deg(f)$, is the maximal degree of its terms. Let $P_1, \ldots, P_n$ be an enumeration of the points of $\boldsymbol{F}_q^m$, the m-dimensional vector space

Manuscript received June 17, 2004; revised November 23, 2004. This work was supported by the Australian Research Council.

X.-W. Wu and M. Kuijper are with the Department of Electrical and Electronic Engineering, University of Melbourne, VIC 3010, Australia (e-mail: x.wu@ee.mu.oz.au; m.kuijper@ee.mu.oz.au).

P. Udaya is with the Department of Computer Science and Software Engineering, University of Melbourne, VIC 3010, Australia (e-mail: udaya@cs.mu.oz.au).

Communicated by R. J. McEliece, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2004.842765

over $\boldsymbol{F}_q$, where $n = q^m$. The q-ary *RM code* $\mathcal{RM}_q(u, m)$ of order u in m variables is defined as

$$\mathcal{RM}_q(u,m) = \{(f(P_1),\ldots,f(P_n)) \mid f \in \boldsymbol{F}_q[X_1,\ldots,X_m], \text{ and } \deg(f) \le u\}.$$

Note that when $m = 1$, the code $\mathcal{RM}_q(u,1)$ is an RS code. $\mathcal{RM}_q(u,m)$ is an $[n,k]$ code, where $n = q^m$ and $k = \binom{u+m}{m}$.

A generalization of the list decoding of RS codes [5] was given in [10] for RM codes. Just like the list decoders of RS and AG codes [5], the list decoder of RM codes [10] completes two main computing tasks. First, given a received vector $\boldsymbol{y} = (y_1,\ldots,y_n)$, the list decoder computes a nontrivial polynomial $H(T)$ in T with coefficients in $\boldsymbol{F}_q[X_1,\ldots,X_m]$. The polynomial $H(T)$ should have every point (P_i, y_i) for $i = 1,\ldots,n$ as its zero of multiplicity ≥ 1. This is a weighted interpolation problem that can be reduced to the problem of solving a system of homogeneous linear equations and can thus be solved with low complexity [5], [9]. Next, given the polynomial $H(T)$, the list decoder finds all the polynomials $f = f(X_1,\ldots,X_m)$ with $\deg(f(X_1,\ldots,X_m)) \le u$, such that $H(f)$ is identically zero. Finally, for these roots $f(X_1,\ldots,X_m)$, the list decoder outputs a list consisting of all the codewords $(f(P_1),\ldots,f(P_n))$, which are candidates for the transmitted codewords.

To find these roots of $H(T)$, a classical method is to factorize $H(T)$ completely. However, the complete factorization is not efficient for the purpose of list decoding of RM codes. In this correspondence, we present an efficient root-finding algorithm which can be used to speed up list decoding of RM codes. The algorithm is designed for solving the following problem.

The Root-Finding Problem: Given nonnegative integers s and u, and a nonzero polynomial

$$H(T) = \sum_{i=0}^{s} h_i(X_1,\ldots,X_m)T^i$$

where

$$h_i(X_1,\ldots,X_m) \in \boldsymbol{F}_q[X_1,\ldots,X_m]$$

find all the polynomials $f = f(X_1,\ldots,X_m) \in \boldsymbol{F}_q[X_1,\ldots,X_m]$ with $\deg(f(X_1,\ldots,X_m)) \le u$ such that $H(f)$ is identically 0.

Among the root-finding algorithms for list decoders of RS and AG codes in the literature [1], [3], [6], [11], [16], the ones in [11] and [16] are simple and have low time complexity. For RS codes, the list decoder needs to find roots of the form $\alpha_0 + \alpha_1 x + \cdots + \alpha_{k-1}x^{k-1}$ of a polynomial $H(T)$ with coefficients in $\boldsymbol{F}_q[x]$, where $k = u + 1$ is the dimension of $\boldsymbol{F}_q[x]_{\le u}$, the set of polynomials in x with degree at most u. This problem is a special case, i.e., $m = 1$, of the above root-finding problem. In [11], an efficient algorithm is proposed that finds the coefficients $\alpha_0, \alpha_1, \ldots, \alpha_{k-1}$ recursively, employing the following properties: 1) the basis elements $1, x, \ldots, x^{k-1}, \ldots$ of the space $\boldsymbol{F}_q[x]$ have a unique common zero at 0 (here 0 is considered a zero of the basis element 1 of multiplicity 0) , and 2) for $i \le j$, x^j/x^i is still a basis element. For AG codes, the list decoder needs to find roots of the form $\alpha_0 + \alpha_1\varphi_1 + \cdots + \alpha_{k-1}\varphi_{k-1}$ of a polynomial $H(T)$, where k is the dimension of a space of rational functions $L(\rho P)$, and the φ_i's are the basis elements of this space, and the coefficients of $H(T)$ are also in this space. The basis elements of the space $L(\rho P)$ have properties that are similar to properties 1) and 2), namely, 1)* $\varphi_0, \varphi_1, \ldots, \varphi_{k-1}$ have a unique common pole at the point P, and 2) * for every $i \le j$, φ_j/φ_i is still an element in $L(\rho P)$. Employing properties 1)* and 2)*, the algorithm in [16] finds the coefficients $\alpha_{k-1}, \ldots, \alpha_1, \alpha_0$ recursively. It is clear that in the RM code case, we have to deal with the space $\boldsymbol{F}_q[X_1,\ldots,X_m]$. The basis elements of this space have neither the properties 1) and 2) nor the properties 1)* and 2)*. Thus, the algorithms in [11] and [16] do not immediately generalize to the list-decoding of RM codes.

In [14], using the ordering associated with an order domain (see [4] for the definition and properties of order domain) instead of the above-mentioned properties, an algorithm for finding roots of a polynomial with coefficients in an order domain is presented. However, no strict proof of the correctness of the algorithm was given there. Because the algorithm suggests that the output size is exponential in the input size, it is important to prove that the root-finding procedure (including checking the output) has low complexity. In [14], this type of efficiency has not been proved.

Below, we will present a simple and efficient algorithm that solves the root-finding problem for list decoding of RM codes. For this we use the graded lexicographical ordering of monomials and modify the algorithm in [14]. We will also prove the correctness of the algorithm. As we will see in Section III, the output of the algorithm may contain some polynomials that are not roots of the input polynomial $H(T)$. As a result, we need to check the elements of the output to get the exact set of roots. After proving a key lemma (Lemma 3.2), we show that the size of the output is at most s, where s is the degree of $H(T)$. This is the most difficult part of the present work. Using this lemma, we prove that for the purpose of list decoding of $[n,k]$ RM codes, the complexity of our root-finding algorithm is $O(kn)$.

One of the features of our algorithm is that it can be used to find all the roots (not only those with degree less than or equal to a specified integer u) in $\boldsymbol{F}_q[X_1,\ldots,X_m]$ of $H(T) = h_0 + h_1T + \cdots + h_sT^s$, that is, the algorithm can find all the linear factors of $H(T)$. As we will see in Section III, if we set the input parameter u large enough, namely,

$$u = \lceil \max\{(\deg(h_i) - \deg(h_s))/(s-i) \mid i = 0,\ldots,s-1\} \rceil$$

then the algorithm finds all the roots of $H(T)$.

This correspondence is organized as follows. In Section II, we present the algorithm and give an example. In Section III, we prove the correctness of the algorithm, and then estimate the output size of the algorithm and evaluate the time complexity. In Section IV, we give the proof of a key lemma (Lemma 3.2), and prove the result on time complexity. Finally, we present our conclusions.

II. The Algorithm

Before presenting our root-finding algorithm, we first give some properties of multivariable polynomials. Denote by $\mathbb{N}_0$ and $\mathbb{N}_0^m$ the sets of nonnegative integers and m-tuples of nonnegative integers, respectively. Note that every monomial $X_1^{a_1}\cdots X_m^{a_m}$ in $\boldsymbol{F}_q[X_1,\ldots,X_m]$ uniquely corresponds to an element $(a_1,\ldots,a_m)$ in $\mathbb{N}_0^m$.

Denote by $<_o$ the graded lexicographical ordering on $\mathbb{N}_0^m$, which is defined as $(a_1,\ldots,a_m) <_o (b_1,\ldots,b_m)$, provided that one of the following two conditions holds:

i) $\sum_{i=1}^m a_i < \sum_{i=1}^m b_i$;

ii) $\sum_{i=1}^m a_i = \sum_{i=1}^m b_i$, and for some $v \in \{1,2,\ldots,m\}$, $a_v < b_v$, $a_{v+1} = b_{v+1},\ldots,a_m = b_m$ (here, if $v = m$ we view $a_i = b_i = 0$ for $i > v$).

For example, when $m = 2$, in $\mathbb{N}_0^2$ under the graded lexicographical ordering

$$(0,0) <_o (1,0) <_o (0,1) <_o (2,0) <_o (1,1) <_o (0,2) <_o \cdots.$$

This gives an ordering of monomials in $\boldsymbol{F}_q[X_1, \ldots, X_m]$, that is,

$$X_1^{a_1} \cdots X_m^{a_m} <_o X_1^{b_1} \cdots X_m^{b_m}$$

if and only if $(a_1, \ldots, a_m) <_o (b_1, \ldots, b_m)$. Now, for a polynomial

$$f = \sum \alpha_{a_1 \ldots a_m} X_1^{a_1} \cdots X_m^{a_m}$$

we call a term $\alpha_{a_1 \ldots a_m} X_1^{a_1} \cdots X_m^{a_m}$ the *leading term*, if for any $\alpha_{b_1 \ldots b_m} \neq 0$, $X_1^{b_1} \cdots X_m^{b_m} <_o X_1^{a_1} \cdots X_m^{a_m}$, which is denoted by $LT(f)$. The coefficient $\alpha_{a_1 \ldots a_m}$ of the leading term is called the *leading coefficient*, which is denoted by $LC(f)$. It is clear that the degree of f is equal to the degree of its leading term.

Let $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$ denote the set of polynomials

$$f(X_1, \ldots, X_m) \in \boldsymbol{F}_q[X_1, \ldots, X_m]$$

with

$$\deg(f(X_1, \ldots, X_m)) \leq u.$$

It can be proved that $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$ is a linear space over $\boldsymbol{F}_q$ and has dimension $k = \binom{u+m}{m}$.

Let $\{\varphi_0, \ldots, \varphi_{k-1}\}$ be a basis of $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$, where $\varphi_0 = 1$ and for $1 \leq j \leq k-1$, φ_j is some monomial $X_1^{a_1} \cdots X_m^{a_m}$ with $a_1 + \cdots + a_m \leq u$. We assume that $\varphi_0 <_o \varphi_1 <_o \cdots <_o \varphi_{k-1}$. This means that for every $j \in \{0, \ldots, k-2\}$, either $\deg(\varphi_j) < \deg(\varphi_{j+1})$, or φ_j and φ_{j+1} can be written as

$$\varphi_j = X_1^{a_1} \cdots X_v^{a_v} X_{v+1}^{a_{v+1}} \cdots X_m^{a_m}$$

and

$$\varphi_{j+1} = X_1^{a_1} \cdots X_v^{a_v - 1} X_{v+1}^{a_{v+1}+1} \cdots X_m^{a_m}$$

for some $1 \leq v \leq m$. Note that φ_j and φ_{j+1} are only differ in the exponents of X_v and X_{v+1}. We will use these facts in the sequel.

For the basis $\{\varphi_0, \ldots, \varphi_{k-1}\}$, every polynomial

$$f \in \boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$$

can be written uniquely as

$$f = f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1}$$

where $f_0, \ldots, f_{k-1}$ are elements of $\boldsymbol{F}_q$. For convenience, in the following algorithm we use the array $[f_{k-1}, \ldots, f_1, f_0]$ of elements in $\boldsymbol{F}_q$ to represent the polynomial $f = f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1}$.

Algorithm 2.1

INPUT: A nonzero polynomial $H(T) = h_0 + h_1 T + \cdots + h_s T^s$ where $h_j \in \boldsymbol{F}_q[X_1, \ldots, X_m]$, and a basis $\{\varphi_0, \ldots, \varphi_{k-1}\}$ of $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$.

OUTPUT: A list that contains all the roots of $H(T)$ in $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$.

Step 1: Set $i := 0$. Set $H_0(T) := H(T)$.

Step 2: Substitute $T = z\varphi_{k-i-1}$ into $H_i(T)$, where z denotes an undetermined element in $\boldsymbol{F}_q$. $H_i(z\varphi_{k-i-1})$ is a polynomial in the variables $X_1, \ldots, X_m$. Compute the leading coefficient of $H_i(z\varphi_{k-i-1})$, which is denoted by $g_i(z)$, i.e.,

$$g_i(z) = LC(H_i(z\varphi_{k-i-1})).$$

Clearly, $g_i(z)$ is a polynomial in z with coefficients in $\boldsymbol{F}_q$.

Step 3: Find all the roots of $g_i(z)$.

Step 4: For each of the distinct roots, α_{k-i-1}, of the polynomial $g_i(z)$,

if $i < k-1$, set

$$H_{i+1}(T) := H_i(T + \alpha_{k-i-1}\varphi_{k-i-1}),$$

set $i \longleftarrow i+1$, and return to Step 2.

Otherwise, go to Step 5.

Step 5: Output all arrays $[\alpha_{k-1}, \ldots, \alpha_1, \alpha_0]$.

Now, we give an example to illustrate the above algorithm.

Example 1: Given the following polynomial:

$$H(T) = T^3 - (xy + y + x + 2)T^2 + (xy^2 + 2xy + 2y + x + 1)T - (xy^2 + xy + y)$$

we want to find all the roots of $H(T)$ in $\boldsymbol{F}_7[x, y]_{\leq 2}$. The space $\boldsymbol{F}_7[x, y]_{\leq 2}$ has a basis $1, x, y, x^2, xy, y^2$, where $1 <_o x <_o y <_o x^2 <_o xy <_o y^2$. We will use the algorithm to find the coefficients $f_0, \ldots, f_5$ of the roots $f = f_0 + f_1 x + f_2 y + f_3 x^2 + f_4 xy + f_5 y^2$ recursively.

First by Step 2, we get $g_0(z) = z^3$. The root of $g_0(z)$ is $\alpha_5 = 0$.

From $\alpha_5 = 0$ and $H_0(T) = H(T)$, by Step 4, we get $H_1(T) = H(T)$. By Step 2, we have $g_1(z) = z^2(z-1)$. The roots of $g_1(z)$ are: $\alpha_4 = 0$ or 1.

For $\alpha_5 = 0$ and $\alpha_4 = 0$, by Steps 4 and 2, we get $g_2(z) = -z^2$. The root of $g_2(z)$ is $\alpha_3 = 0$.

For $\alpha_5 = 0$ and $\alpha_4 = 1$, by Steps 4 and 2, we get $g_2(z) = z$. The root of $g_2(z)$ is $\alpha_3 = 0$.

Now, for $\alpha_5 = 0$, $\alpha_4 = 0$ and $\alpha_3 = 0$, by Steps 4 and 2, we have $g_3(z) = -z(z-1)$. By Step 3, we have $\alpha_2 = 0$ or 1.

For $\alpha_5 = 0$, $\alpha_4 = 1$ and $\alpha_3 = 0$, by Steps 4 and 2, we have $g_3(z) = z$. By Step 3, we have $\alpha_2 = 0$.

For $\alpha_5 = 0$, $\alpha_4 = 0$, $\alpha_3 = 0$, and $\alpha_2 = 0$, by Steps 4 and 2, we have $g_4(z) = z$. By Step 3, we have $\alpha_1 = 0$.

For $\alpha_5 = 0$, $\alpha_4 = 0$, $\alpha_3 = 0$, and $\alpha_2 = 1$, by Steps 4 and 2, we have $g_4(z) = -z$. By Step 3, we have $\alpha_1 = 0$.

For $\alpha_5 = 0$, $\alpha_4 = 1$, $\alpha_3 = 0$, and $\alpha_2 = 0$ by Steps 4 and 2, we have $g_4(z) = z - 1$. By Step 3, we have $\alpha_1 = 1$.

For $\alpha_5 = 0$, $\alpha_4 = 0$, $\alpha_3 = 0$, $\alpha_2 = 0$, and $\alpha_1 = 0$, by Steps 4 and 2, we have $g_5(z) = z - 1$. By Step 3, we have $\alpha_0 = 1$.

For $\alpha_5 = 0$, $\alpha_4 = 0$, $\alpha_3 = 0$, $\alpha_2 = 1$, and $\alpha_1 = 0$, by Steps 4 and 2, we have $g_5(z) = -z$. By Step 3, we have $\alpha_0 = 0$.

For $\alpha_5 = 0$, $\alpha_4 = 1$, $\alpha_3 = 0$, $\alpha_2 = 0$ and $\alpha_1 = 1$, by Steps 4 and 2, we have $g_5(z) = z - 1$. By Step 3, we have $\alpha_0 = 1$.

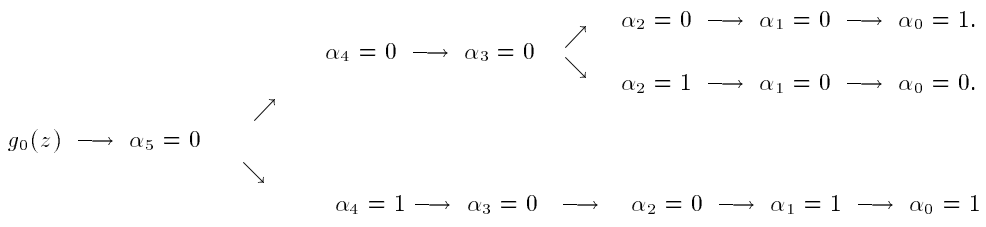
The above root-finding procedure is represented in graph form at the top of the following page.

The algorithm returns three arrays

$$[0, 0, 0, 0, 0, 1], \quad [0, 0, 0, 1, 0, 0], \quad [0, 1, 0, 0, 1, 1].$$

They correspond to three polynomials: 1, y, and $1 + x + xy$. It is easy to check that each of these polynomials is a root of $H(T)$. As $\deg_T(H(T)) = 3$, $H(T)$ can be factorized as $H(T) = (T-1)(T-y)(T-xy-x-1)$.

Remark 2.1: In the next section, we prove that every root of $H(T)$ in $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$ is contained in the output list of the algorithm. However, this does not guarantee that every element of the output list is a root of $H(T)$. For example, suppose $H(T) = (1 + x + xy) + (xy)T$



and we want to find the roots of $H(T)$ in $\boldsymbol{F}_2[x, y]$ of degree 0. For this case, it can be easily verified that the output of the algorithm is 1. But 1 is not a root of $H(T)$, since $H(T)$ has no roots of degree 0. Therefore, to get the exact set of roots, we need to check the elements of the output list.

To show the efficiency of the whole root-finding procedure, it is very important to prove that the size of the output list is bounded from above by a polynomial in the input size. In Theorem 3.3, we show that the output list has size at most s, where s is the degree of $H(T)$. The proof of Theorem 3.3 is based on a key lemma (Lemma 3.2).

III. Correctness and Complexity of the Algorithm

In this section, we prove the correctness of the algorithm and evaluate its time complexity.

Theorem 3.1: Let $H(T) = h_0 + h_1T + \cdots + h_sT^s$ be a nonzero polynomial with coefficients in $\boldsymbol{F}_q[X_1, \ldots, X_m]$. Then, Algorithm 2.1 returns a list that contains all the roots in $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$ of $H(T)$.

Proof: Let

$$f = f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1} \in \boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$$

be any root of $H(T) = h_0 + h_1T + \cdots + h_sT^s$. We will prove that the coefficients $f_{k-1}, \ldots, f_0$ of f are found recursively by the algorithm. Thus, f is in the output of the algorithm. Since f is a root of $H(T)$, i.e., $H(f) \equiv 0$, we have that

$$H(f) = h_0 + h_1(f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1}) + \cdots\cdots + \\ + h_s(f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1})^s$$

is the zero polynomial in $\boldsymbol{F}_q[X_1, \ldots, X_m]$, i.e., all the coefficients of $H(f)$ coincide with the zero element in $\boldsymbol{F}_q$. In particular, the leading coefficient of $H(f)$ is zero, i.e., $LC(H(f)) = 0$. Since $1 <_o \varphi_1 <_o \cdots <_o \varphi_{k-1}$, this leading coefficient equals

$$\begin{aligned} LC(H(f)) &= LC(h_0 + h_1(f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1}) + \cdots\cdots \\ &\quad + h_s(f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1})^s) \\ &= LC(h_0 + h_1 \cdot f_{k-1}\varphi_{k-1} + \cdots\cdots + h_s \cdot f_{k-1}^s\varphi_{k-1}^s) \\ &= LC(H(f_{k-1}\varphi_{k-1})). \end{aligned}$$

Consider $g_0(z) := LC(H_0(z\varphi_{k-1}))$, where z denotes an undetermined element in $\boldsymbol{F}_q$ and $H_0(T) := H(T)$. This is a polynomial in z with coefficients in $\boldsymbol{F}_q$ and degree $\leq s$. Because of the assumption that f is a root of $H(T)$, f_{k-1} is a root of $g_0(z)$. It is therefore found in Step 3 of the algorithm.

For $i = 0, \ldots, k-2$, from the definition of H_{i+1} in Step 4 of the algorithm, we have

$$\begin{aligned} &H_{i+1}(f_0 + f_1\varphi_1 + \cdots + f_{k-i-2}\varphi_{k-i-2}) \\ &\quad = H_i(f_0 + f_1\varphi_1 + \cdots + f_{k-i-1}\varphi_{k-i-1}) \\ &\quad\quad \cdots\cdots \\ &\quad = H_0(f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1}) \\ &\quad = H(f_0 + f_1\varphi_1 + \cdots + f_{k-1}\varphi_{k-1}) \\ &\quad \equiv 0. \end{aligned}$$

Here H_{i+1} has been defined from $H_i(T)$ and f_{k-i-1}, whereas $H_i(T)$ has been defined from $H_{i-1}(T)$ and f_{k-i}, and so on. Now repeating the above argument it is easily shown that

$$LC(H_{i+1}(f_{k-i-2}\varphi_{k-i-2})) = 0$$

i.e., f_{k-i-2} is a root of $g_{i+1}(z) = LC(H_{i+1}(z\varphi_{k-i-2}))$. As a result, f_{k-i-2} is found in Step 3.

We conclude that $f_{k-1}, f_{k-2}, \ldots, f_0$ are found by the algorithm. □

We now give a key lemma, which will be proved in the next section. The lemma is used in Theorem 3.3 below to estimate the output size of the algorithm, as well as evaluate the complexity of the algorithm.

Lemma 3.2: Let $P(T) = p_0 + p_1T + \cdots + p_sT^s$ be a nonzero polynomial with $p_j \in \boldsymbol{F}_q[X_1, \ldots, X_m]$ for $j = 0, 1, \ldots, s$. Let $\varphi_0, \varphi_1, \ldots, \varphi_d$ be a basis of the space $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq w}$. Suppose $\alpha \in \boldsymbol{F}_q$ is a root of multiplicity r, where r is a positive integer, of the polynomial equation

$$g(z) := LC(P(z\varphi_d)) = 0$$

where z denotes an undetermined element in $\boldsymbol{F}_q$, and the leading coefficient of $P(z\varphi_d)$ is computed with respect to the variables $X_1, \ldots, X_m$. Now define $\tilde{P}(T)$ and $\tilde{g}(z)$ from $P(T)$ and α as

$$\tilde{P}(T) := P(T + \alpha\varphi_d), \qquad \tilde{g}(z) := LC(\tilde{P}(z\varphi_{d-1})).$$

Then, $\tilde{g}(z)$ is a polynomial in z of degree $\leq r$.

Theorem 3.3: Let $H(T) = h_0 + h_1T + \cdots + h_sT^s$ be a nonzero polynomial with coefficients in $\boldsymbol{F}_q[X_1, \ldots, X_m]$. Then, the output list of Algorithm 2.1 contains at most s elements.

Proof: The algorithm is designed for finding the roots of $H(T)$ in $\boldsymbol{F}_q[X_1, \ldots, X_m]_{\leq u}$, which is a k-dimensional space. We will use mathematical induction on k to prove this theorem. As we have seen, $g_0(z)$ is a polynomial in z with coefficients in $\boldsymbol{F}_q$ and degree $\leq s$. As a result, $g_0(z)$ has at most s roots (counting the multiplicities) α_{k-1}, and we get at most s arrays $[\alpha_{k-1}]$ of length 1.

Now, suppose the polynomial $g_{i-1}(z)$ has $t \leq s$ roots α_{k-i}, denoted by $\alpha_{k-i}^{(1)}, \ldots, \alpha_{k-i}^{(t)}$, such that every $\alpha_{k-i}^{(j)}$ is a root of multiplicity r_j, where $r_1 + \cdots + r_t \leq s$. Also, we suppose that from these roots we get at most $t \leq s$ arrays $[\alpha_{k-1}, \ldots, \alpha_{k-i}]$ of length i. From Steps 4 and 2, for each root $\alpha_{k-i}^{(j)}$, we construct an $H_i(T)$ and the corresponding $g_i(z)$. By Lemma 3.2, $g_i(z)$ is a polynomial in z of degree $\leq r_j$. Hence, $g_i(z)$ has at most r_j roots α_{k-i-1}. Thus, in total, we can get at most $r_1 + \cdots + r_t \leq s$ arrays $[\alpha_{k-1}, \ldots, \alpha_{k-i}, \alpha_{k-i-1}]$ of length $i + 1$.

By induction, we conclude that we have at most s arrays $[\alpha_{k-1}, \alpha_{k-2}, \ldots, \alpha_0]$. This means that the size of the output list of the algorithm is at most s. □

Before we evaluate the time complexity of Algorithm 2.1, we give some notations and assumptions to the size of the input of the algorithm. For $H(T) = h_0 + h_1T + \cdots + h_sT^s$ and $\{\varphi_0, \ldots, \varphi_{k-1}\}$, we denote $L := \max\{\deg(h_i) + i \cdot \deg(\varphi_{k-1}) \mid i = 0, \ldots, s\}$. Then

$$\deg(h_i) \leq L - i \cdot \deg(\varphi_{k-1}), \qquad i = 0, \ldots, s. \tag{1}$$

Consider the vector space $\boldsymbol{F}_q[X_1,\ldots,X_m]_{\le L}$ that consists of all polynomials of degree $\le L$. Let $K = \binom{L+m}{m}$ be the dimension of $\boldsymbol{F}_q[X_1,\ldots,X_m]_{\le L}$. Then every polynomial in $X_1,\ldots,X_m$ of degree at most L has at most K terms. Now, we have the following result on the time complexity of the algorithm, which we will prove in the next section.

Theorem 3.4: The time complexity of Algorithm 2.1 is $O(s^2kK)$.

Note that in practical list decoding, s is a designed constant parameter, k is the code dimension and less than or equal to the code length n. Also, K is bounded from above by a linear polynomial in n (for example, in the case of list decoding of RS codes, $K \le n$). Thus, for the purpose of list decoding of RM codes the time complexity of our algorithm is $O(kn)$, which is the same as the complexity of the root-finding algorithm for RS codes in [11].

Remark 3.1:

1) Obviously, a special version (i.e., when $m = 1$) of our algorithm gives a root-finding procedure for RS list decoding.
2) Although our algorithm is designed for finding the roots of $H(T)$ in $\boldsymbol{F}_q[X_1,\ldots,X_m]_{\le u}$, by setting the input parameter u large enough, the algorithm can be used to find all the roots of $H(T)$, as is shown in the following proposition.

Proposition 3.5: Suppose

$$H(T) = h_0 + h_1T + \cdots + h_sT^s$$

where $h_0, h_1, \ldots, h_s \in \boldsymbol{F}_q[X_1,\ldots,X_m]$ and $h_s \neq 0$. Let u be given as

$$u = \lceil \max\{(\deg(h_i) - \deg(h_s))/(s-i) \mid i = 0,\ldots,s-1\}\rceil.$$

Then, any root of $H(T)$ in $\boldsymbol{F}_q[X_1,\ldots,X_m]$ has degree at most u. Therefore, if we set the input parameter u as above, then Algorithm 2.1 returns all the roots of $H(T)$.

Proof: Suppose f is a polynomial in $\boldsymbol{F}_q[X_1,\ldots,X_m]$ with

$$\deg(f) > \left\lceil \max\left\{ \frac{\deg(h_i) - \deg(h_s)}{s-i} \;\middle|\; i = 0,\ldots,s-1 \right\}\right\rceil.$$

Consider

$$H(f) = h_0 + h_1f + \cdots + h_sf^s.$$

We have that for $i = 0, 1, \ldots, s-1$

$$\begin{aligned}\deg(h_sf^s) - \deg(h_if^i) &= \deg(h_s) - \deg(h_i) + (s-i)\deg(f)\\ &> \deg(h_s) - \deg(h_i) + \deg(h_i) - \deg(h_s)\\ &= 0\end{aligned}$$

that is, $\deg(h_sf^s) > \deg(h_if^i)$. Here, the inequality above is obtained from the assumption $\deg(f) > (\deg(h_i) - \deg(h_s))/(s-i)$ for all $i = 0,\ldots,s-1$. This implies that $H(f)$ can not be the zero polynomial. □

IV. Proofs of Lemma 3.2 and Theorem 3.4

Proof of Lemma 3.2: First we write out $\tilde{g}(z)$. We have

$$\begin{aligned}\tilde{P}(T) &= P(T + \alpha\varphi_d)\\ &= p_0 + p_1(T+\alpha\varphi_d) + p_2(T+\alpha\varphi_d)^2\\ &\quad + \cdots + p_s(T+\alpha\varphi_d)^s.\end{aligned}$$

Therefore,

$$\begin{aligned}\tilde{g}(z) &= LC(\tilde{P}(z\varphi_{d-1}))\\ &= LC(p_0 + p_1\cdot(\alpha\varphi_d + z\varphi_{d-1})\\ &\quad + p_2\cdot(\alpha^2\varphi_d^2 + 2\alpha z\varphi_d\varphi_{d-1} + z^2\varphi_{d-1}^2)\\ &\quad + \cdots + p_s\cdot(\alpha^s\varphi_d^s + \binom{s}{1}\alpha^{s-1}z\varphi_d^{s-1}\varphi_{d-1}\\ &\quad + \cdots + z^s\varphi_{d-1}^s)).\end{aligned}$$

Since α is a constant in $\boldsymbol{F}_q$, while z is an undetermined element in $\boldsymbol{F}_q$, $\tilde{g}(z)$ is a polynomial in z with coefficients in $\boldsymbol{F}_q$ and degree at most s. We want to prove that the degree of $\tilde{g}(z)$ is at most r, provided that α is a root of $g(z)$ of multiplicity r.

When $\alpha \neq 0$, consider the terms

$$(\alpha^2\varphi_d^2 + 2\alpha z\varphi_d\varphi_{d-1} + z^2\varphi_{d-1}^2),\ldots$$

and

$$(\alpha^s\varphi_d^s + \binom{s}{1}\alpha^{s-1}z\varphi_d^{s-1}\varphi_{d-1} + \binom{s}{2}\alpha^{s-2}z^2\varphi_d^{s-2}\varphi_{d-1}^2 + \cdots + z^s\varphi_{d-1}^s).$$

Since $\varphi_{d-1}^2 <_o \varphi_d\varphi_{d-1}$, the term $z^2\varphi_{d-1}^2$ does not contribute to the leading term of $\tilde{P}(z\varphi_{d-1})$. Similarly, since

$$\varphi_d^{s-2}\varphi_{d-1}^2 <_o \varphi_d^{s-1}\varphi_{d-1}, \ldots, \varphi_{d-1}^s <_o \varphi_d^{s-1}\varphi_{d-1}$$

the terms

$$\binom{s}{2}\alpha^{s-2}z^2\varphi_d^{s-2}\varphi_{d-1}^2, \ldots, z^s\varphi_{d-1}^s$$

do not contribute to the leading term of $\tilde{P}(z\varphi_{d-1})$. Thus,

$$\tilde{g}(z) = LC(p_0 + p_1\cdot(\alpha\varphi_d + z\varphi_{d-1}) + p_2\cdot(\alpha^2\varphi_d^2 + 2\alpha z\varphi_d\varphi_{d-1}) + \cdots + p_s\cdot(\alpha^s\varphi_d^s + \binom{s}{1}\alpha^{s-1}z\varphi_d^{s-1}\varphi_{d-1})).$$

It is clear that $\tilde{g}(z)$ is a polynomial in z of degree $\le 1 \le r$. Therefore, the assertion is true.

When $\alpha = 0$, we have

$$\tilde{g}(z) = LC(p_0 + p_1z\varphi_{d-1} + p_2z^2\varphi_{d-1}^2 + \cdots + p_sz^s\varphi_{d-1}^s). \quad (2)$$

Let us look at

$$g(z) = LC(p_0 + p_1\varphi_dz + \cdots + p_r\varphi_d^rz^r + p_{r+1}\varphi_d^{r+1}z^{r+1} + \cdots + p_s\varphi_d^sz^s).$$

In the following, we assume that $p_0, p_1, \ldots, p_s$ are monomials without loss of generality (since, otherwise, we can only consider their leading terms). Since 0 is a root of $g(z)$ of multiplicity r, for $j = 0,\ldots,r-1$, $p_j\varphi_d^jz^j$ does not contribute to the leading term, while $p_r\varphi_d^rz^r$ contributes to the leading term. Thus, under the ordering $<_o$, we have $p_r\varphi_d^r >_o p_0$, …, and $p_r\varphi_d^r >_o p_{r-1}\varphi_d^{r-1}$, and

$$p_r\varphi_d^r \ge_o \begin{cases} p_{r+1}\varphi_d^{r+1},\\ p_{r+2}\varphi_d^{r+2},\\ \vdots\\ p_s\varphi_d^s. \end{cases} \quad (3)$$

To prove that (2) is a polynomial in z of degree $\le r$, it is sufficient to prove the following:

$$p_r\varphi_{d-1}^r >_o \begin{cases} p_{r+1}\varphi_{d-1}^{r+1},\\ p_{r+2}\varphi_{d-1}^{r+2},\\ \vdots\\ p_s\varphi_{d-1}^s. \end{cases} \quad (4)$$

From the first inequality of (3), we have

$$\deg(p_r) + r\cdot\deg(\varphi_d) \ge \deg(p_{r+1}) + (r+1)\cdot\deg(\varphi_d).$$

From this inequality and $\deg(\varphi_d) \geq \deg(\varphi_{d-1})$ we have

$$\deg(p_r)+r\cdot\deg(\varphi_{d-1}) \geq \deg(p_{r+1}) + (r+1)\cdot\deg(\varphi_{d-1}).$$

If either

$$\deg(p_r) + r\cdot\deg(\varphi_d) > \deg(p_{r+1}) + (r+1)\cdot\deg(\varphi_d)$$

or

$$\deg(\varphi_d) > \deg(\varphi_{d-1})$$

holds, we have

$$\deg(p_r) + r\cdot\deg(\varphi_{d-1}) > \deg(p_{r+1}) + (r+1)\cdot\deg(\varphi_{d-1}).$$

This implies that the first inequality in (4) is true. Otherwise, φ_{d-1} and φ_d have the same degree and can be written as follows (see Section II):

$$\varphi_{d-1} = X_1^{a_1}\cdots X_v^{a_v}X_{v+1}^{a_{v+1}}\cdots X_m^{a_m}$$

and

$$\varphi_d = X_1^{a_1}\cdots X_v^{a_v-1}X_{v+1}^{a_{v+1}+1}\cdots X_m^{a_m}.$$

Suppose $p_r = \beta X_1^{c_1}\cdots X_m^{c_m}$ and $p_{r+1} = \gamma X_1^{d_1}\cdots X_m^{d_m}$, where $\beta,\gamma \in \boldsymbol{F}_q$. The first inequality in (3) is equivalent to the following:

$$\begin{aligned}(&c_1 + ra_1,\ldots,c_v + r(a_v-1), c_{v+1} + r(a_{v+1}+1),\\ &\quad c_{v+2} + ra_{v+2},\ldots,c_m + ra_m)\\ &\geq_o (d_1 + (r+1)a_1,\ldots,d_v + (r+1)(a_v-1),\\ &\quad d_{v+1} + (r+1)(a_{v+1}+1),\\ &\quad d_{v+2} + (r+1)a_{v+2},\ldots,d_m + (r+1)a_m) \qquad (5)\end{aligned}$$

where the ith components in the left-hand and right-hand vectors are the exponents of X_i in $p_r\varphi_d^r$ and $p_{r+1}\varphi_d^{r+1}$, respectively. The first inequality in (4) is equivalent to

$$\begin{aligned}(&c_1 + ra_1,\ldots,c_v + ra_v, c_{v+1} + ra_{v+1},\\ &\quad c_{v+2} + ra_{v+2},\ldots,c_m + ra_m)\\ &>_o (d_1 + (r+1)a_1,\ldots,d_v + (r+1)a_v,\\ &\quad d_{v+1} + (r+1)a_{v+1},\\ &\quad d_{v+2} + (r+1)a_{v+2},\ldots,d_m + (r+1)a_m). \qquad (6)\end{aligned}$$

Suppose (5) is true. We are going to prove (6). Comparing the left-hand vectors of (5) and (6), they have the same last $(m-v-1)$ components. Also, the right-hand vectors of (5) and (6) have the same last $(m-v-1)$ components. If there exists $j \geq v+2$ such that $c_j + ra_j > d_j + (r+1)a_j$ and

$$\begin{aligned}c_{j+1} + ra_{j+1} &= d_{j+1} + (r+1)a_{j+1},\\ &\vdots\\ c_m + ra_m &= d_m + (r+1)a_m\end{aligned}$$

then we are done. Otherwise,

$$\begin{aligned}c_j + ra_j &= d_j + (r+1)a_j\\ c_{j+1} + ra_{j+1} &= d_{j+1} + (r+1)a_{j+1},\\ &\vdots\\ c_m + ra_m &= d_m + (r+1)a_m\end{aligned}$$

and from (5) and the definition of $<_o$, we have

$$c_{v+1} + r(a_{v+1}+1) \geq d_{v+1} + (r+1)(a_{v+1}+1).$$

This inequality implies $c_{v+1} + ra_{v+1} > d_{v+1} + (r+1)a_{v+1}$. This implies that (6) is true.

Therefore, we have proved the first inequality of (4) making use of the first inequality of (3). In the same way, we can prove the other inequalities of (4) using (3). □

Proof of Theorem 3.4: From the algorithm, we see that for $i = 1,\ldots,k-1$, we need to construct

$$H_i(T) := H_{i-1}(T + \alpha_{k-i}\varphi_{k-i}).$$

Assume that $H_i(T) = h_0^{(i)} + h_1^{(i)}T + \cdots + h_s^{(i)}T^s$. Using mathematical induction on k, we will prove that for $i = 1,\ldots,k-1$

$$\deg(h_j^{(i)}) \leq L - j\cdot\deg(\varphi_{k-1}), \qquad j = 0,\ldots,s. \qquad (7)$$

Then by the definition of K prior to Theorem 3.4, every $h_j^{(i)}$ contains at most K terms.

We have

$$H_i(T) = H_{i-1}(T + \alpha_{k-i}\varphi_{k-i}) = h_0^{(i)} + h_1^{(i)}T + \cdots + h_s^{(i)}T^s.$$

So

$$\begin{aligned}h_0^{(i)} &= h_0^{(i-1)} + \alpha_{k-i}\cdot\varphi_{k-i}h_1^{(i-1)} + \cdots\\ &\quad + \alpha_{k-i}^s\cdot\varphi_{k-i}^s h_s^{(i-1)}\\ h_1^{(i)} &= h_1^{(i-1)} + 2\alpha_{k-i}\cdot\varphi_{k-i}h_2^{(i-1)} + \cdots\\ &\quad + \binom{s}{1}\alpha_{k-i}^{s-1}\cdot\varphi_{k-i}^{s-1}h_s^{(i-1)}\\ h_2^{(i)} &= h_2^{(i-1)} + 3\alpha_{k-i}\cdot\varphi_{k-i}h_3^{(i-1)} + \cdots\\ &\quad + \binom{s}{2}\alpha_{k-i}^{s-2}\cdot\varphi_{k-i}^{s-2}h_s^{(i-1)}\\ &\vdots\\ h_s^{(i)} &= h_s^{(i-1)} \qquad (8)\end{aligned}$$

Therefore, we have

$$\begin{aligned}\deg(h_j^{(i)}) \leq \max\{&\deg(h_j^{(i-1)}),\ \deg(h_{j+1}^{(i-1)}) + \deg(\varphi_{k-i}),\ldots,\\ &\deg(h_s^{(i-1)}) + (s-j)\cdot\deg(\varphi_{k-i})\}. \qquad (9)\end{aligned}$$

For $i = 1$, we have

$$\begin{aligned}\deg(h_j^{(1)}) \leq \max\{&\deg(h_j),\ \deg(h_{j+1}) + \deg(\varphi_{k-1}),\ldots,\\ &\deg(h_s) + (s-j)\cdot\deg(\varphi_{k-1})\}.\end{aligned}$$

From (1), $\deg(h_j) \leq L - j\cdot\deg(\varphi_{k-1})$, for $j = 0,\ldots,s$. We have that $\deg(h_j^{(1)}) \leq L - j\cdot\deg(\varphi_{k-1})$. Now, suppose

$$\deg(h_j^{(i-1)}) \leq L - j\cdot\deg(\varphi_{k-1}).$$

By (9), we then have

$$\deg(h_j^{(i)}) \leq L - j\cdot\deg(\varphi_{k-1}).$$

So, by induction, we have (7). This means that for $i = 0,\ldots,k-1$ and $j = 0,\ldots,s$, $h_j^{(i)}$ is of degree at most L and thus contains at most K terms.

Now, consider Steps 4 and 2. Since for $i = 0,\ldots,k-1$ and $j = 0,\ldots,s$, $h_j^{(i)}$ has at most K terms, Step 4 requires s^2K operations and Step 2 requires sK operations over $\boldsymbol{F}_q$. Thus, Steps 4 and 2 contribute $O(ks^2K)$ field operations over $\boldsymbol{F}_q$ to the overall complexity. Next, consider the complexity of Step 3. It is well known that the roots in $\boldsymbol{F}_q$ of a polynomial of degree s can be found in expected time complexity $O((s\log^2 s)\cdot(\log\log s)\cdot\log q)$ (see [2], [11], [16]). Therefore, Step 3 contributes $O(k((s\log^2 s)\cdot(\log\log s)\cdot\log q))$ field operations over $\boldsymbol{F}_q$ to the overall complexity. As the dominant term between $O(ks^2K)$ and

$O(k((s\log^2 s)\cdot(\log\log s)\cdot\log q))$ is $O(ks^2K)$, the time complexity of the algorithm is $O(ks^2K)$. □

V. Conclusion

In this correspondence, we have proposed an efficient algorithm that finds all the roots in $\boldsymbol{F}_q[X_1,\ldots,X_m]_{\le u}$ of a polynomial $H(T)$ with coefficients in $\boldsymbol{F}_q[X_1,\ldots,X_m]$. Setting the input parameter u large enough, our algorithm can find all the linear factors of $H(T)$ with a complexity of $O(kn)$. A special version of our algorithm (i.e., when $m=1$) gives a root-finding procedure for RS list decoding, with the same complexity as the algorithm in [11]. Our algorithm can be used to speed up the list decoding of the q-ary RM codes.

Acknowledgment

The authors would like to thank the anonymous referees whose thoughtful criticisms and valuable suggestions helped to improve the quality of this correspondence.

References

[1] D. Augot and L. Pecquet, "A Hensel listing to replace factorization in list-decoding of algebraic-geometric and Reed-Solomon codes," *IEEE Trans. Inf. Theory*, vol. 46, no. 7, pp. 2605–2614, Nov. 2000.

[2] M. Ben-Or, "Probabilistic algorithms in finite fields," in *Proc. 22nd Annu. IEEE Symp. Foundations of Computer Science*, Nashville, TN, Oct. 1981, pp. 394–398.

[3] S.-H. Gao and M. Shokrollahi, "Computing roots of polynomials over function fields of curves," in *Coding Theory and Cryptography*, D. Joyner, Ed. New York: Springer-Verlag, 1999, pp. 114–228.

[4] O. Geil and R. Pellikaan, "On the structure of order domains," *Finite Fields Their Applic.*, vol. 8, pp. 369–396, 2002.

[5] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometry codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 7, pp. 1757–1767, Nov. 1999.

[6] T. Høholdt and R. Nielsen, "Decoding Hermitian codes with Sudan's algorithm," in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (Lecture Notes in Computer Science)*, M. Fossorier, H. Imei, S. Lin, and A. Pole, Eds. Berlin, Germany: Springer-Verlag, 1999, vol. 1719, pp. 260–270.

[7] R. Nielsen and T. Høholdt, "Decoding Reed-Solomon codes beyond half the minimum distance," in *Coding Theory, Cryptography and Related Areas*, J. Buchmann, T. Høholdt, T. Stichtenoth, and H. Tapia-Recillas, Eds. Berlin, Germany: Springer-Verlag, 2000, pp. 221–236.

[8] R. Koetter and A. Vardy, "Algebraic soft-decision decoding of Reed–Solomon codes," *IEEE Trans. Inf. Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.

[9] M. Kuijper and J. W. Polderman, "Reed–Solomon list decoding from a system-theoretic perspective," *IEEE Trans. Inf. Theory*, vol. 50, no. 2, pp. 259–271, Feb. 2004.

[10] R. Pellikaan and X.-W. Wu, "List decoding of q-ary Reed–Muller codes," *IEEE Trans. Inf. Theory*, vol. 50, no. 4, pp. 679–682, Apr. 2004.

[11] R. Roth and G. Ruckenstein, "Efficient decoding of Reed–Solomon codes beyond half the minimum distance," *IEEE Trans. Inf. Theory*, vol. 46, no. 1, pp. 246–257, Jan. 2000.

[12] M. Shokrollahi and H. Wasserman, "List decoding of algebraic-geometric codes," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 432–437, Mar. 1999.

[13] M. Sudan, "Decoding of Reed-Solomon codes beyond the error-correction bound," *J. Complexity*, vol. 13, pp. 180–193, 1997.

[14] X.-W. Wu, "An algorithm for finding the roots of the polynomials over order domains," in *Proc. IEEE Int. Symp. Information Theory*, Lausanne, Switzerland, Jun./Jul. 2002, p. 202.

[15] X.-W. Wu, M. Kuijper, and P. Udaya, "Lee-metric decoding of BCH and Reed-Solomon codes," *Electron. Lett.*, vol. 39, no. 21, pp. 1522–1524, 2003.

[16] X.-W. Wu and P. H. Siegel, "Efficient root-finding algorithm with applications to list decoding of algebraic-geometric codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 6, pp. 2579–2587, Sep. 2001.

A Simple Algorithm for Decoding Reed–Solomon Codes and its Relation to the Welch–Berlekamp Algorithm

Sergei V. Fedorenko, *Member, IEEE*

***Abstract*—A simple and natural Gao algorithm for decoding algebraic codes is described. Its relation to the Welch–Berlekamp and Euclidean algorithms is given.**

***Index Terms*—Decoding algorithm, decoding Reed–Solomon (RS) codes, Euclidean algorithm, key equation, RS codes, remainder decoding, Welch–Berlekamp algorithm.**

I. Introduction

In the recent paper, Gao [1] described a simple and natural algorithm for decoding algebraic codes in the class of algorithms decoding up to the designed error-correcting capability. The asymptotic complexity of this algorithm coincides with the complexity of the best Reed–Solomon decoding algorithms, and the description is the simplest for known algorithms descriptions. In this correspondence, the Gao algorithm's relation to the Welch–Berlekamp [2] and Euclidean algorithms [3], [4] is given.

II. Definitions and Notations

Let us define the (n,k,d) Reed–Solomon (RS) code over GF(q) with length $n=q-1$, number of information symbols k, designed distance $d=n-k+1$, q is prime power.

The RS code generator polynomial is

$$g(x)=\prod_{i=b}^{b+d-2}(x-\alpha^i)$$

where α is a primitive element of GF(q), b is any natural number.

The received vector is represented as a polynomial

$$R(x)=\sum_{i=0}^{n-1}r_ix^i=C(x)+E(x)=\sum_{i=0}^{n-1}c_ix^i+\sum_{i=0}^{n-1}e_ix^i$$

where $C(x)$ is the codeword, $E(x)$ is the error vector.

The ith error in the error vector $E(x)$ has a locator

$$Z_i\in\{\alpha^0,\alpha^1,\alpha^2,\ldots,\alpha^{n-1}\}$$

and an error value $Y_i\in \mathrm{GF}(q)\backslash 0$.

The error locator polynomial is

$$W(x)=\prod_{i=1}^{t}(x-Z_i)$$

Manuscript received April 15, 2004; revised August 25, 2004.

The author is with the Department of Distributed Computing and Networking, St. Petersburg State Polytechnic University, 194021 St.Petersburg, Russia (e-mail: sfedorenko@ieee.org).

Communicated by R. J. McEliece, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2004.842738