Performance of Reed–Solomon codes using the Guruswami–Sudan algorithm with improved interpolation efficiency

L. Chen, R.A. Carrasco and E.G. Chester

Abstract: List decoding is a novel method for decoding Reed–Solomon (RS) codes that generates a list of candidate transmitted messages instead of one unique message as with conventional algebraic decoding, making it possible to correct more errors. The Guruswami–Sudan (GS) algorithm is the most efficient list decoding algorithm for RS codes. Until recently only a few papers in the literature suggested practical methods to implement the key steps (interpolation and factorisation) of the GS algorithm that make the list decoding of RS codes feasible. However, the algorithm's high decoding complexity is unsolved and a novel complexity-reduced modification to improve its efficiency is presented. A detailed explanation of the GS algorithm with the complexity-reduced modification is given with simulation results of RS codes for different list decoding parameters over the AWGN and Rayleigh fading channels. A complexity analysis is presented comparing the GS algorithm with our modified GS algorithm, showing the modification can reduce complexity significantly in low error weight situations. Simulation results using the modified GS algorithm show larger coding gains for RS codes with lower code rates, with more significant gains being achieved over the Rayleigh fading channels.

1 Introduction

The idea of list decoding block codes was introduced by Elias [1] and Wozencraft [2] independently in the 1950s. In 1997, Sudan [3] introduced this idea to decode low-rate (n, k) RS codes beyond the half-distance boundary $\lfloor (n-k-1)/2 \rfloor$, where n is the code length and k is the message length. Later, Guruswami and Sudan [5, 6] improved the algorithm to decode RS codes of nearly any code rate beyond this boundary. However, this algorithm was still impractical to implement until Kotter and Vardy [7–9] and Roth and Ruckenstein [10] presented low-complexity implementation methods for the key steps of the GS algorithm: interpolation and factorisation. In 2003, McEliece [11] gave an explicit tutorial discussion of the algorithm.

Traditional algebraic decoding algorithms for RS codes generate a unique decoded codeword. These algorithms include the Berlekamp–Massey algorithm [12] and Euclid's algorithm [13, 14] and are very efficient in terms of running time. However, they are unable to correct any number of errors greater than $\lfloor (n-k-1)/2 \rfloor$ limiting the performance of RS codes over deeply corruptive channels. The GS algorithm for RS codes removes this limitation by finding a list of possible transmitted messages with decoding considered to be successful as long as the transmitted message is included in the list. The correct transmitted message is chosen by re-encoding the list of candidate messages and selecting the codeword with minimum distance to the received word. According to Guruswami and Sudan [5, 6], the GS algorithm improves the error correction capability significantly for low-rate ($<1/3$) RS codes. However, for higher rate codes this algorithm can still improve the error correction capability but with a less significant improvement.

Kotter and Vardy [8] presented an algebraic soft-decision list decoder for RS codes. This algorithm is based on the interpolation theorem used in the GS algorithm. In addition, the reliability information of each symbol in the received word is used. According to [15], this decoder has higher complexity for low error weights and weaker error dependence compared with the GS algorithm, but has slightly lower complexity for high error weights. Therefore overall the GS algorithm is still one of the most efficient list decoding algorithms for RS codes.

The GS algorithm has not been assessed by many researchers due to its high decoding complexity and it also requires a good understanding of mathematics. However, its greater error-correction capability makes it is a potential alternative decoding algorithm for RS codes and can be extended to the family of algebraic–geometric codes [5, 7], which lead to wider applications. This paper describes the principle of the GS algorithm for RS codes from an algebraic–geometric point of view. Addressed towards improving the algorithm's decoding efficiency, a novel complexity-reduced modification to the original algorithm is presented and a detailed complexity analysis is given. Using this modified GS algorithm, simulation results are presented showing the performance of this algorithm for different rate RS codes over the AWGN and Rayleigh fading channels. In the literature there are very few simulation results that have been published.

doi:10.1049/iet-com:20060057

Paper first received 21st October 2005 and in revised form 24th April 2006

The authors are with the School of Electrical, Electronic and Computer Engineering, University of Newcastle-upon-Tyne, NE1 7RU, UK

E-mail: r.carrasco@newcastle.ac.uk

2 Overview of Guruswami–Sudan algorithm

Before explaning the GS algorithm we first denote some commonly used symbols in this paper

F_q – finite field with q elements;
$F_q[x]$ – ring of polynomials with coefficients from F_q and variable x;
$F_q[x^w]$ – ring of polynomials from $F_q[x]$ with x degree $\leq w$;
$F_q[x, y]$ – ring of bivariate polynomials with coefficients from F_q and variables x and y.

2.1 Encoding Reed–Solomon codes

If $f(x)$ is a subspace of $F_q[x^{k-1}]$, the generation of a (n, k) RS code can be described as evaluating $f(x)$ at a set of points $x_0, x_1, \ldots, x_{n-1} \in F_q$

$$(c_0, c_1, \ldots, c_{n-1}) = (f(x_0), f(x_1), \ldots, f(x_{n-1})) \quad (1)$$

where $f(x)$ can be described as a linear combination of functions $1, x, \ldots, x^{k-1}$ with its coefficients in the finite field $f_0, f_1, \ldots, f_{k-1} \in F_q$

$$f(x) = f_0 + f_1 x + \cdots + f_{k-1} x^{k-1} \quad (2)$$

The coefficients $f_0, f_1, \ldots, f_{k-1}$ are regarded as the transmitted message.

2.2 Brief description of Guruswami–Sudan algorithm

The GS algorithm includes two key steps: interpolation and factorisation.

Interpolation: If the received word is $y = (y_0, y_1, \ldots, y_{n-1})$, then combining with the finite-field elements used in encoding $(x_0, x_1, \ldots, x_{n-1})$, one can form n points as $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$. The task of interpolation is to construct a bivariate polynomial

$$Q(x, y) = \sum_{i,j} q_{ij} x^i y^j \quad (3)$$

which has a zero of order m over these n points and with minimal $(1, k-1)$-weight degree, which is explained subsequently and $q_{ij} \in F_q$ is the coefficient of $x^i y^j$. This polynomial intersects the n points m times and the value of m is also called multiplicity in this paper.

Factorisation: After the bivariate polynomial $Q(x, y)$ has been found, we must factorise $Q(x, y)$ to find the list L of polynomials $p(x)$ given by

$$L = \{p(x): (y - p(x)) | Q(x, y) \quad \text{and} \quad \deg(p(x)) < k\} \quad (4)$$

All the polynomials in L have the possibility of being the transmitted message $f(x)$. The one with minimum distance to the received word after re-encoding is chosen by the decoder.

2.3 Decoding parameters

To understand the GS algorithm it is necessary to introduce a few decoding parameters prior to the detailed description.

Defining the (u, v)-weight degree of monomial $x^i y^j$ as

$$\text{w-deg}_{u,v}(x^i y^j) = iu + jv \quad (5)$$

a sequence of bivariate monomials can be arranged by their weight degrees. To decode a (n, k) RS code by the GS algorithm, the $(1, k-1)$-reverse lexicographic ($(1, k-1)$-revlex) order is used. Under $(1, k-1)$-revlex order [16, 23]

$$x^{i_1} y^{j_1} < x^{i_2} y^{j_2}, \text{ if}$$

$$\text{w-deg}_{1,k-1}(x^{i_1} y^{j_1}) < \text{w-deg}_{1,k-1}(x^{i_2} y^{j_2}),$$

$$\text{or w-deg}_{1,k-1}(x^{i_1} y^{j_1}) = \text{w-deg}_{1,k-1}(x^{i_2} y^{j_2}) \text{ and } i_1 > i_2$$

For example, to decode a (7, 5) RS code, (1, 4)-revlex order is used. The generation of this order is shown by Table 1. The entries E_{ij} in Tables 1 and 2 represent the (1, 4)-weight degree and (1, 4)-revlex order of monomials M with x degree i and y degree j, respectively. Applying (5) with $u = 1$ and $v = 4$, one can generate the (1, 4)-weight degree of monomials M shown by Table 1. Based on Table 1 and applying the revlex order rule, one can generate the (1, 4)-revlex order of monomials M shown by Table 2 and denoted as ord(M). From Table 2, it is easy to determine that under (1, 4)-revlex order, $x^4 < x^2 y < y^2$, since $\text{ord}(x^4) = 4$, $\text{ord}(x^2 y) = 9$ and $\text{ord}(y^2) = 14$.

Based on the monomials' degree and order definition we define the weight degree of a nonzero bivariate polynomial as the weight degree of its leading monomial M_L. Any nonzero bivariate polynomial $Q(x, y)$ can be written as

$$Q(x, y) = a_0 M_0 + a_1 M_1 + \cdots + a_L M_L, \text{ with}$$
$$M_0 < M_1 < \cdots < M_L, a_0, a_1, \ldots, a_L \in F_q \text{ and } a_L \neq 0 \quad (6)$$

The $(1, k-1)$-weight degree of $Q(x, y)$ can be defined as

$$\text{w-deg}_{1,k-1}(Q(x, y)) = \text{w-deg}_{1,k-1}(M_L) \quad (7)$$

L is called the leading order, lod, of polynomial $Q(x, y)$, defined as

$$\text{lod}(Q(x, y)) = \text{ord}(M_L) = L \quad (8)$$

Therefore any two nonzero polynomials $Q(x, y)$ and $H(x, y)$ ($Q, H \in F_q[x, y]$) can be compared with respect to their

Table 1: (1, 4)-weight degree

i	0	1	2	3	4	5	6	7	8	9	10	11	12	...
j														
0	0	1	2	3	4	5	6	7	8	9	10	11	12	...
1	4	5	6	7	8	9	10	11	12	...				
2	8	9	10	11	12	...								
3	12	...												
⋮	⋮													

Table 2: (1, 4)-revlex order

i	0	1	2	3	4	5	6	7	8	9	10	11	12	...
j														
0	0	1	2	3	4	6	8	10	12	15	18	21	24	...
1	5	7	9	11	13	16	19	22	25	...				
2	14	17	20	23	26	...								
3	27	...												
⋮	⋮													

leading order

$$Q(x, y) \leq H(x, y), \text{ if } \text{lod}(Q(x, y)) \leq \text{lod}(H(x, y)) \quad (9)$$

$S_x(N)$ and $S_y(N)$ are denoted as the highest degree of x and y under the $(1, k-1)$-revlex order such that

$$S_x(N) = \max\{i: \text{ord}(x^i y^0) \leq N\} \quad (10)$$

$$S_y(N) = \max\{j: \text{ord}(x^0 y^j) \leq N\} \quad (11)$$

where N is any nonnegative integer. It is interesting that under $(1, k-1)$-revlex order $x^i y^0$ is the minimal monomial with weight degree i. Therefore the $(1, k-1)$-weight degree of any nonzero bivariate polynomial defined in (6) with leading order L can be determined as

$$\text{w-deg}_{1,k-1}(Q(x, y)) = S_x(L) \quad (12)$$

The error-correction capability t_m and the maximum number of candidate messages l_m in the output list with respect to a certain multiplicity m ($m \geq 1$) of the GS algorithm can be stated as [11]

$$t_m = n - 1 - \left\lfloor \frac{S_x(C)}{m} \right\rfloor \quad (13)$$

$$l_m = S_y(C) \quad (14)$$

where

$$C = n\binom{m+1}{2} \quad (15)$$

These parameters are proven in Section 4. t_m and l_m grow monotonically with multiplicity m [11]

$$t_{m_1} < t_{m_2} < \cdots < t_{m_x} < \cdots < t_{m_{GS}} \quad (16)$$

$$l_{m_1} < l_{m_2} < \cdots < l_{m_x} < \cdots < l_{m_{GS}} \quad (17)$$

where $m_1 < m_2 < \cdots < m_x < \cdots < m_{GS}$ and $t_{m_{GS}}$ is the upper error-correcting bound of the GS algorithm, defined as [6]

$$t_{m_{GS}} = n - 1 - \left\lfloor \sqrt{(k-1)n} \right\rfloor \quad (18)$$

$t_{m_{GS}}$ is greater or equal to the half distance boundary $\lfloor (n-k-1)/2 \rfloor$ and approaches to it asymptotically with code-rate k/n increases. According to the GS algorithm analysis in [5], GS algorithm's decoding capability merges with the conventional algebraic decoding algorithm at about $k/n = 0.9$. Note that the generalised minimum distance (GMD) decoding algorithm [18] performance does not depend on the code-rate and it can always outperform the conventional decoding algorithm with small coding gain. Simulations results in [8] show that the GS algorithm can outperform the GMD algorithm in relatively low code-rate situations. However, as code rate increases, the GS algorithm's performance will approach to the conventional decoding algorithm, and the GMD algorithm can slightly outperform the GS algorithm.

Now two examples are given to illustrate how t_m and l_m grow with multiplicity m with the GS algorithm. Notice that those m listed in the following examples are the minimal values need to correct the corresponding number of errors t_m.

Example 1: To decode RS(63, 15) defined over F_{64}, with code rate 0.238 ($<1/3$), we obtain

m	1	2	4	6	$26 = m_{GS}$
t_m	27	30	31	32	33
l_m	2	4	8	13	55

Example 2: To decode RS(63, 31) defined over F_{64}, with code rate 0.492 ($>1/3$), we obtain

m	1	3	5	$13 = m_{GS}$
t_m	16	17	18	19
l_m	1	4	7	19

3 Interpolation

The interpolation theorem is explained from the algebraic geometric point of view, followed by a detailed description of Kotter's interpolation algorithm and a novel modification which improves its efficiency.

3.1 Interpolation theorem

For RS codes, $1, x, \ldots, x^a$ are the rational functions that have increasing pole order [19] over the point of infinity p_∞ of a projective curve. The interpolated polynomial can generally be written as

$$Q(x, y) = \sum_{a,b \in N} q_{ab} x^a y^b \quad (19)$$

where $q_{ab} \in F_q$ is the coefficient of $x^a y^b$. Functions $1, (1-x_i), \ldots, (1-x_i)^u$ are the rational functions that have increasing zero order over the finite-field element x_i used in encoding, and the received word $y_i \in F_q$. The interpolated polynomial with respect to point (x_i, y_i) can also be written as

$$Q(x, y) = \sum_{u,v \in N} q_{uv}^{(x_i, y_i)} (x - x_i)^u (y - y_i)^v \quad (20)$$

where $q_{uv}^{(x_i, y_i)} \in F_q$ is the coefficient of $(x - x_i)^u (y - y_i)^v$. If $q_{uv}^{(x_i, y_i)} = 0$ for $u + v < m$, $Q(x, y)$ has a zero of multiplicity m over (x_i, y_i).

Notice that

$$x^a = (x - x_i + x_i)^a = \sum_{a \geq u} \binom{a}{u} x_i^{a-u} (x - x_i)^u \quad (21)$$

and

$$y^b = (y - y_i + y_i)^b = \sum_{b \geq v} \binom{b}{v} y_i^{b-v} (y - y_i)^v \tag{22}$$

Substituting (21) and (22) into (19),

$$\begin{aligned} Q(x,y) &= \sum_{a,b} q_{ab} \sum_{a \geq u} \binom{a}{u} x_i^{a-u} (x - x_i)^u \sum_{b \geq v} \binom{b}{v} y_i^{b-v} (y - y_i)^v \\ &= \sum_{u,v} \sum_{a \geq u, b \geq v} q_{ab} \binom{a}{u} \binom{b}{v} x_i^{a-u} y_i^{b-v} (x - x_i)^u (y - y_i)^v \end{aligned} \tag{23}$$

Therefore from (20)

$$q_{uv}^{(x_i,y_i)} = \sum_{a \geq u, b \geq v} q_{ab} \binom{a}{u} \binom{b}{v} x_i^{a-u} y_i^{b-v} \tag{24}$$

This is the (u, v)-Hasse derivative evaluation on the point (x_i, y_i) of the polynomial $Q(x, y)$ defined by (19) [17, 20, 21]. Using $D(Q)$ to denote the Hasse derivative evaluation of $Q(x, y)$, (24) can be denoted as

$$D_{uv} Q(x_i, y_i) = \sum_{a \geq u, b \geq v} q_{ab} \binom{a}{u} \binom{b}{v} x_i^{a-u} y_i^{b-v} \tag{25}$$

Therefore the interpolation of the GS algorithm can be generalised as: Find a minimal $(1, k-1)$-weight degree polynomial $Q(x, y)$ that satisfies

$$\begin{aligned} Q(x,y) = \min\{Q(x,y) \in F_q[x,y] | D_{uv} Q(x_i, y_i) = 0 \\ \text{for } i = 0, \ldots, n-1 \text{ and } u + v < m (u, v \in N)\} \end{aligned} \tag{26}$$

3.2 Kotter's algorithm

Kotter [6–8] suggested an efficient polynomial reconstruction algorithm to find the polynomial defined by (26), called Kotter's algorithm. It is an iterative modification algorithm based on the following two properties of the Hasse derivative [17, 20].

Property 1: Linear functional of Hasse derivative
If $H, Q \in F_q[x, y]$, c_1 and $c_2 \in F_q$, then

$$D(c_1 H + c_2 Q) = c_1 D(H) + c_2 D(Q) \tag{27}$$

Property 2: Bilinear Hasse derivative
If $H, Q \in F_q[x, y]$, then

$$[H, Q]_D = HD(Q) - QD(H) \tag{28}$$

If the Hasse derivative evaluation of $D(Q) = d_1$ and $D(H) = d_2$ ($d_1, d_2 \neq 0$), based on Property 1 it is obvious to conclude that the Hasse derivative evaluation of (28) is zero, denoted as

$$D([H, Q]_D) = 0 \tag{29}$$

If $\text{lod}(H) > \text{lod}(Q)$, the new constructed polynomial from (28) has leading order $\text{lod}(H)$. Therefore by performing the bilinear Hasse derivative over two polynomials both of which have nonzero evaluations, one can reconstruct a polynomial which has zero Hasse derivative evaluation. Based on this principle, Kotter's algorithm is to iteratively modify a set of polynomials through all n points and with every possible (u, v) pair under each point.

With multiplicity m, there are $\binom{m+1}{2}$ pairs of (u, v), which are arranged as: $(u, v) = (0, 0), (0, 1), \ldots, (0, m-1), (1, 0), (1, 1), \ldots, (1, m-2), \ldots, (m-1, 0)$. Therefore when decoding a (n, k) RS code with multiplicity m, there are $C = n\binom{m+1}{2}$ iterative modifications under Kotter's algorithm in order to construct a polynomial defined by (26). If i denotes the index of points ($i = 0, 1, \ldots, n-1$), r denotes the index of (u, v) pairs ($r = 0, 1, \ldots, \binom{m+1}{2} - 1$), the index of the iterative modification i_k can be written as: $i_k = i\binom{m+1}{2} + r$.

At the beginning of Kotter's algorithm, a group of polynomials are initialised as

$$G_0 = \{g_{0,j} = y^j, j = 0, 1, \ldots, l_m\} \tag{30}$$

where l_m is the maximal number of messages in the output list defined by (14). If M_L denotes the leading monomial of g, it is important to point out that

$$g_{0,j} = \min\{g(x,y) \in F_q[x,y] | \deg_y(M_L) = j\} \tag{31}$$

Under i_k modification, each polynomial in group G_{i_k} is tested by (25) using

$$\Delta_j = D_{i_k}(g_{i_k,j}) \tag{32}$$

Those polynomials with $\Delta_j = 0$ do not need to be modified. However, those polynomials with $\Delta_j \neq 0$ need to be modified based on (28). To construct a group of polynomials which satisfy

$$g_{i_k+1,j} = \min \left\{ \begin{array}{c} g(x,y) \in F_q[x,y] | D_{i_k}(g_{i_k+1,j}) = 0, \\ D_{i_k-1}(g_{i_k+1,j}) = 0, \ldots, D_0(g_{i_k+1,j}) = 0 \\ \text{and } \deg_y(M_L) = j \end{array} \right\} \tag{33}$$

we choose the minimal polynomial among those polynomials with $\Delta_j \neq 0$, denote its index as j^* and record it as g^*

$$j^* = \text{index}(\min\{g_{i_k,j} | \Delta_j \neq 0\}) \tag{34}$$

$$g^* = g_{i_k,j^*} \tag{35}$$

For those polynomials with $\Delta_j \neq 0$ but $j \neq j^*$, modify them by (28) without the leading order increasing

$$g_{i_k+1,j} = [g_{i_k,j}, g^*]_{D_{i_k}} \tag{36}$$

Based on (29) we know that $D_{i_k}(g_{i_k+1,j}) = 0$. As $\text{lod}(g_{i_k,j*}) > \text{lod}(g^*)$, therefore $\text{lod}(g_{i_k+1,j}) = \text{lod}(g_{i_k,j})$. For g^* itself, we modify it by (28) with the leading order increasing

$$g_{i_k+1,j^*} = [xg^*, g^*]_{D_{i_k}} \tag{37}$$

$\Delta_{j^*} = D_{i_k}(g^*) \neq 0$ and so as $D_{i_k}(xg^*) \neq 0$, therefore $D_{i_k}(g_{i_k+1,j^*}) = 0$. As $\text{lod}(xg^*) > \text{lod}(g^*)$, $\text{lod}(g_{i_k+1,j^*}) = \text{lod}(xg^*) > \text{lod}(g_{i_k,j^*})$. Therefore whenever (37) is performed, $\text{lod}(g_{i_k+1,j}) > \text{lod}(g_{i_k,j})$. After C iterative modifications the minimal polynomial in G_C is the interpolated polynomial that satisfies (26), and it is chosen to be factorised in the next step

$$Q(x,y) = \min\{g_{C,j} | g_{C,j} \in G_C\} \tag{38}$$

3.3 Complexity-reduced modification

Based on this analysis, when decoding a (n, k) RS code with multiplicity m, $l_m + 1$ bivariate polynomials are being interactively modified over C steps in which Hasse derivative evaluation and bilinear Hasse derivative modification are being performed. This is responsible for the GS algorithm's high decoding complexity. Therefore reducing the complexity of interpolation is essential to improve the algorithm's efficiency.

The leading order of the polynomial group G_{i_k} is defined as the minimal leading order among the group's polynomials

$$\mathrm{lod}(G_{i_k}) = \min\{\mathrm{lod}(g_{i_k,j}) | g_{i_k,j} \in G_{i_k}\} \tag{39}$$

Based on initialisation defined in (30), the leading order of polynomial group G_0 is $\mathrm{lod}(G_0) = \mathrm{lod}(g_{0,0}) = 0$. In the i_k modification, if no polynomial needs to be modified, the polynomial group is unchanged, $\mathrm{lod}(G_{i_k+1}) = \mathrm{lod}(G_{i_k})$. Once a polynomial needs to be modified, (37) must be used. If M_L is the leading monomial of g^*, we have

$$\mathrm{lod}(xg^*) = \mathrm{lod}(g^*) + \left\lfloor \frac{\deg_x g^*}{k-1} \right\rfloor + \deg_y(M_L) + 1 \tag{40}$$

and $\mathrm{lod}(G_{i_k})$ will be increased if g^* is the minimal polynomial in the group G_{i_k}. The leading order increase guarantees that in the i_k iterative step, the leading order of the polynomials group G_{i_k} is always less than or equal to i_k

$$\mathrm{lod}(G_{i_k}) \le i_k \tag{41}$$

Based on (41), after C iterative steps

$$\mathrm{lod}(G_C) \le C \tag{42}$$

From (38) we know that only the minimal polynomial is chosen from the polynomial group G_C as $Q(x, y) = \{g_{c,j} | g_{c,j} \in G_c \text{ and } \mathrm{lod}(g_{c,j}) = \mathrm{lod}(G_c)\}$, therefore

$$\mathrm{lod}(Q(x, y)) \le C \tag{43}$$

which means the interpolated polynomial $Q(x, y)$ has leading order less than or equal to C. Those polynomials with leading order over C will not be candidates to be $Q(x, y)$. Therefore during the iterative process, we can modify the group of polynomials by eliminating those with leading order over C as

$$G_{i_k} = \{g_{i_k,j} | \mathrm{lod}(g_{i_k,j}) \le C\} \tag{44}$$

We now prove this modification will not affect the final result. In i_k iterative step, if there is a polynomial $g_{i_k,j}$ with $\mathrm{lod}(g_{i_k,j}) > C$, it may be modified either by (36) or (37) which will result in its leading order being unchanged or increased. Therefore at the end $\mathrm{lod}(g_{c,j}) > C$ and based on (43) it cannot be $Q(x, y)$. However, if $g_{i_k,j}$ is the minimal polynomial defined by (34), this implies that those polynomials with leading order less than C do not need to be modified. If $g_{i_k,j}$ is not the minimal polynomial defined by (34), $g_{i_k,j}$ will not be chosen to perform bilinear Hasse derivative (36) with other polynomials. Therefore $Q(x, y)$ has no information introduced from $g_{i_k,j}$ since $\mathrm{lod}(g_{i_k,j}) > C$. As a result, eliminating the polynomials with leading order over C will not affect the final outcome.

This modification can reduce some unnecessary computation in terms of avoiding Hasse derivative evaluation (32) and bilinear Hasse derivative modification (36) (37) of polynomials with leading order over C. Based on the preceding analysis, the modified interpolation process can be summarised as

(i) Initialise a group of polynomials by (30), set $i_k = 0$;
(ii) Modify the polynomial group by (44);
(iii) Perform Hasse derivative evaluation (32) for each polynomial in the group;
(iv) If all the polynomials' Hasse derivative evaluation are zero, go to (vii);
(v) Find the minimal polynomial defined by (34), (35);
(vi) For the minimal polynomial, modify it by (37); For the other polynomials with nonzero Hasse derivative evaluation, modify them by (36);
(vii) $i_k = i_k + 1$;
(viii) If $i_k = C$, stop the process and choose $Q(x, y)$ defined by (38); else go to (ii).

Here an example is given showing how the modified algorithm affects the iterative process.

Example 3: Decode the (7, 2) RS code defined over F_8 with multiplicity $m = 2$. As $C = 7\binom{3}{1} = 21$, based on (13) (14) we have $t_2 = 3$ and $l_2 = 5$. The transmitted codeword is generated by evaluating the message polynomial $f(x)$ over the set of points $x = (1, \alpha, \alpha^3, \alpha^2, \alpha^6, \alpha^4, \alpha^5)$ and the corresponding received word is $y = (\alpha^5, \alpha^3, \alpha^4, 0, \alpha^6, \alpha^2, \alpha^2)$, where α is a primitive element in F_8 and is a root of the primitive polynomial $x^3 + x + 1 = 0$. Construct a bivariate polynomial that has a zero of multiplicity $m = 2$ over the n points $(x_i, y_i)|_{i=0}^{n-1}$.

At the beginning, six polynomials are initialised as $g_{0,0} = 1$, $g_{0,1} = y$, $g_{0,2} = y^2$, $g_{0,3} = y^3$, $g_{0,4} = y^4$ and $g_{0,5} = y^5$.

The whole iterative process with respect to the polynomials' leading order is shown in Table 3. From Table 3 we can see that the modified algorithm starts to take action at $i_k = 10$ when there is polynomial with leading order over 21 and eliminating those polynomials will not affect the final outcome. At the end, both the original and modified GS algorithm produce the same result: $Q(x, y) = \min\{G_{21}\} = g_{21,2} = 1 + \alpha^4 x^2 + \alpha^2 x^4 + y^2 (\alpha^5 + \alpha^4 x^2)$. From this example we see that more computation can be reduced if the modified algorithm starts to take action at earlier steps. A detailed complexity analysis of this modified algorithm is presented in Section 5.

4 Factorisation

4.1 Factorisation theorem

Let $\Lambda(p, Q)$ denote the number of points that satisfy $Q(x_i, p(x_i)) = 0$

$$\Lambda(p, Q) = |\{i : Q(x_i, p(x_i)) = 0, \quad (i = 0, 1, \ldots, n-1)\}| \tag{45}$$

Suppose the bivariate polynomial $Q(x, y)$ has a zero of multiplicity m over the $\Lambda(p, Q)$ points, $p(x) \in F_q[x^{k-1}]$ and $Q(x, y) \in F_q[x, y]$. If

$$m\Lambda(p, Q) > \text{w-deg}_{1,k-1}(Q(x, y))$$

then $(y - p(x)) | Q(x, y)$.

Lemma 1: $Q(x, p(x)) = 0$ if and only if $(y - p(x)) | Q(x, y)$ [10].

From Lemma 1, one see that to factorise $Q(x, y)$ is equivalent to find the y roots of it. As $Q(x, y)$ is the interpolated polynomial from Kotter's algorithm, according to (43)

Table 3: Interpolation process of Example 3

i_k	0	1	2	3	4	5	6	7	8	9			
lod($g_{i_k,0}$)	0	1	1	3	6	6	10	15	15	21			
lod($g_{i_k,1}$)	2	2	4	4	4	7	7	7	11	11			
lod($g_{i_k,2}$)	5	5	5	5	5	5	5	5	5	5			
lod($g_{i_k,3}$)	9	9	9	9	9	9	9	9	9	9			
lod($g_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14			
lod($g_{i_k,5}$)	20	20	20	20	20	20	20	20	20	20			
lod(G_{i_k})	0	1	1	3	4	5	5	5	5	5			
	i_k	10	11	12	13	14	15	16	17	18	19	20	21
originalGS →	lod($g_{i_k,0}$)	28	28	36	45	45	55	55	55	55	66	66	78
	lod($g_{i_k,1}$)	11	16	16	16	22	22	22	22	22	22	29	29
	lod($g_{i_k,2}$)	5	5	5	5	5	5	8	8	12	12	12	12
	lod($g_{i_k,3}$)	9	9	9	9	9	9	9	13	13	13	13	13
	lod($g_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14	14	14
	lod($g_{i_k,5}$)	20	20	20	20	20	20	20	20	20	20	20	20
	lod(G_{i_k})	5	5	5	5	5	5	8	8	12	12	12	12
	i_k	10	11	12	13	14	15	16	17	18	19	20	21
modifiedGS →	lod($g_{i_k,0}$)	–	–	–	–	–	–	–	–	–	–	–	–
	lod($g_{i_k,1}$)	11	16	16	16	–	–	–	–	–	–	–	–
	lod($g_{i_k,2}$)	5	5	5	5	5	5	8	8	12	12	12	12
	lod($g_{i_k,3}$)	9	9	9	9	9	9	9	13	13	13	13	13
	lod($g_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14	14	14
	lod($g_{i_k,5}$)	20	20	20	20	20	20	20	20	20	20	20	20
	lod(G_{i_k})	5	5	5	5	5	5	8	8	12	12	12	12

Note: – means corresponding polynomial is eliminated. Highlighted area means corresponding polynomial is chosen as $Q(x, y)$

with $\mathrm{lod}(Q) \le C$. Based on (12), $\text{w-deg}_{1,k-1}(Q(x,y)) \le S_x(C)$. If $m\Lambda(p, Q) \ge S_x(C)$, then $m\Lambda(p, Q) \ge \text{w-deg}_{1,k-1}(Q(x, y))$. Based on the factorisation theorem, if $\Lambda(p, Q) \ge 1 + \lfloor S_x(C)/m \rfloor$, candidate message polynomial $p(x)$ can be found out by factorising $Q(x, y)$. As $\Lambda(p, Q)$ represents the number of points that satisfy $Q(x_i, p(x_i)) = 0$, or equivalently $y_i = p(x_i)$ where $p(x)$ is the candidate transmitted message polynomial, those points that do not satisfy this equation are where the errors locate. Therefore the error correction capability of the GS algorithm is $t_m = n - \lfloor S_x(C)/m \rfloor - 1$ which is defined by (13). Under $(1, k-1)$-revlex order x^0y^j is the maximal monomial with weight degree $(k-1)j$. In polynomial $Q(x, y)$, there should not be any monomials with y-degree over $S_y(C)$, otherwise $\mathrm{lod}(Q) > C$. As a result, $\max\{\deg_y Q(x, y)\} \le S_y(C)$. From Lemma 1 we know that the messages in the output list are the y roots of $Q(x, y)$, and the number of y roots of $Q(x, y)$ should not exceed its y-degree, therefore the maximal number of candidate messages in the output list is $l_m = S_y(C)$ which is defined by (14).

4.2 Roth–Ruckenstein's algorithm

In 2000, Roth and Ruckenstein [10] introduced an efficient algorithm for factorising these bivariate polynomials, called Roth-Ruckenstein's algorithm. Each $p(x) \in F_q[x^{k-1}]$ can be expressed in the form of

$$p(x) = p_0 + p_1 x + \cdots + p_{k-1}x^{k-1} \tag{46}$$

where $p_0, p_1, \ldots, p_{k-1} \in F_q$. To find the polynomials $p(x)$, we must determine their coefficients $p_0, p_1, \ldots, p_{k-1}$. The idea of Roth-Ruckenstein's algorithm is to sequentially deduce $p_0, p_1, \ldots, p_{k-1}$ one at a time. For any bivariate polynomial, if h is the highest degree such that $x^h | Q(x, y)$, we can define

$$Q^*(x, y) = \frac{Q(x, y)}{x^h} \tag{47}$$

Denoting $p_0 = p(x)$ and $Q_0(x, y) = Q^*(x, y)$, where $Q(x, y)$ is the new interpolated polynomial (38), we define the sequential polynomials $p_s(x)$ and $Q_s(x, y)$, where $s \ge 1$, as

$$p_s(x) = \frac{p_{s-1}(x) - p_{s-1}(0)}{x} = p_s + \cdots + p_{k-1}x^{k-1-s}, (s \le k-1) \tag{48}$$

$$Q_s(x, y) = Q^*_{s-1}(x, xy + p_{s-1}) \tag{49}$$

Lemma 2: In this sequential deduction with $p_s(x)$ and $Q_s(x, y)$ defined by (48) and (49), when $s \ge 1$, $(y - p(x)) | Q(x, y)$ if and only if $(y - p_s(x)) | Q_s(x, y)$ [11].

This means that if polynomial $p_s(x)$ is a y root of $Q_s(x, y)$, we can trace back to find the coefficients $p_{s-1}, \ldots, p_1, p_0$ to reconstruct the polynomial $p(x)$, which is the y root of polynomial $Q(x, y)$. The first coefficient p_0 can be determined by finding the roots of $Q_0(0, y) = 0$. If we assume that $Q(x, p(x)) = 0$, then based on Lemma 2, $p_0(x)$ should satisfy $Q_0(x, p_0(x)) = 0$. When $x = 0$, $Q_0(0, p_0(0)) = 0$. According to (46), $p_0(0) = p_0$, therefore p_0 is the root of $Q_0(0, y) = 0$. By finding the roots of $Q_0(0, y) = 0$, a number of different p_0 can be determined. For each p_0, we deduce further to find the rest of p_s $(s = 1, \ldots, k-1)$ based on the sequential transformation (48) and (49).

Assume that after $s-1$ deductions, polynomial $p_{s-1}(x)$ is the y root of $Q_{s-1}(x, y)$. Based on (48), $p_{s-1}(0) = p_{s-1}$ and a number of p_{s-1} can be determined by finding the roots of

$Q_{s-1}(0, y) = 0$. For each p_{s-1}, we deduce to find p_s. Based on the assumption and Lemma 1, $(y - p_{s-1}(x))|Q_{s-1}(x, y)$. If we define $y = xy + p_{s-1}$, then $(xy + p_{s-1} - p_{s-1}(x))|Q_{s-1}(x, xy + p_{s-1})$. Based on (48), $xy + p_{s-1} - p_{s-1}(x) = xy - xp_s(x)$. As $Q_s(x, y) = Q^*_{s-1}(x, xy + p_{s-1})$, $(xy - xp_s(x))|Q_{s-1}(x, xy + p_{s-1})$, and $(y - p_s(x))|Q_s(x, y)$. Therefore p_s can again be determined by finding the roots of $Q_s(0, y) = 0$. This root finding algorithm can be explained as a tree growing process, which is shown in Fig. 1. There can be an exponential number of routes for choosing coefficients p_s $(s = 0, 1, \ldots, k - 1)$ to construct $p(x)$. However, the intended $p(x)$ should satisfy: $\deg(p(x)) < k$ and $(y - p(x))|Q(x, y)$. Based on (48), when $s = k$, $p_k(x) = 0$. Therefore if $Q_k(x, 0) = 0$, or equivalently $Q_k(x, p_k(x)) = 0$, $(y - p_k(x))|Q_k(x, y)$. According to Lemma 2, $(y - p(x))|Q(x, y)$ and $p(x)$ is found.

Based on this analysis, the factorisation process can be summarised as

(i) Initialise $Q_0(x, y) = Q^*(x, y)$, $s = 0$;
(ii) Find roots p_s of $Q_s(0, y) = 0$;
(iii) For each p_s, perform Q transformation (49) to calculate $Q_{s+1}(x, y)$;
(iv) $s = s + 1$;
(v) If $s < k$, go to (ii); if $s = k$ and $Q_s(x, 0) \neq 0$, stop this deduction route; if $s = k$ and $Q_s(x, 0) = 0$, trace the deduction route to find $p_{s-1}, \ldots, p_1, p_0$.

5 Complexity analysis

The GS algorithm's high decoding complexity is mainly caused by interpolation. Compared with this, the factorisation complexity cost is insignificant. This Section analyses the computational complexity (finite-field arithmetic operations) for the original and modified algorithm.

It is difficult to analyse the computational complexity precisely because the length (number of coefficients) of the group of interpolated polynomials varies in different situations. Define the number of coefficients of the polynomial $Q(x, y)$ as its interpolation cost by

$$\gamma = |\{q_{ij} = \text{coeff}(Q(x, y)) \text{ and } q_{ij} \neq 0\}| \tag{50}$$

[17] has stated that the interpolation cost is error dependent

$$\gamma(e) \leq \frac{\Omega^2}{2(k-1)} + \frac{\Omega}{2} + \frac{\Phi(k - \Phi - 1)}{2(k-1)} + m + 1 \tag{51}$$

where $\Omega = em + (k - 1)m$, $\Phi = \Omega \bmod (k - 1)$ and e is the error weight. According to Section 3.3, we know that the interpolated polynomial $Q(x, y)$ has leading order less than or equal to C, therefore its interpolation cost is less than or equal to $C + 1$. Regarding the interpolation process as solving a system of homogeneous linear equations by gaussian elimination and assuming those polynomials have the same interpolation cost as $C + 1$, we predict the GS algorithm's computational complexity. Interpolating the group of polynomials with interpolation cost $C + 1$ by C iterative steps can be regarded as operating on a matrix of size $C \times (C + 1)$ by gaussian elimination and its computational complexity is approximately

$$\frac{2}{3}(C + 1)^3 \tag{52}$$

However, in most of the situations $\gamma(e) \leq C + 1$, which means some elements in the row of the matrix are not used and the row operation is not fully performed. Therefore in most cases (52) is an upper bound for the GS algorithm's computational complexity. As the interpolation cost grows with the error weight, so does the computational complexity. Based on (52), Tables 4 and 5 predict the computational complexity for decoding RS(63, 15) and RS(63, 31) both of which were first introduced in Examples 1 and 2.

In computer simulations, we have measured the computational complexity for the two codes, shown in Fig. 2. Comparing the measurements with Table 4 and 5, in most cases (52) is a computational complexity upper bound for the GS algorithm and the decoding complexity grows with the error weight. With higher multiplicity m, the GS algorithm has better error correction capability, but at the expense of much higher computation. Comparing the computational complexity between the original and modified GS algorithm, it shows that the lower the error weight, the more computation can be reduced. For RS(63, 15) the modification can reduce the computational complexity by 40% in low error weight situations, but in high error weight situations the complexity is only reduced by 3%. For RS(63, 31), the complexity reduction varies from 30 to 2% with increasing error weight. In low error weight situations the conventional algebraic decoding algorithm would be used since it is more efficient than the GS algorithm. Therefore even with the complexity-reduced modification the GS algorithm is not desirable for low error weight situations. However, in high error weight situations, where conventional algebraic decoder could not be used, this modification is useful for reducing the GS algorithm's complexity.

The modification's error dependent property is analysed as follows. During the iterative interpolation process, we define the maximal leading order of the polynomial group G_{i_k} as

$$\max \text{lod}(G_{i_k}) = \max\{\text{lod}(g_{i_k,j}) | g_{i_k,j} \in G_{i_k}\} \tag{53}$$

The modification (44) will start to act when $\text{maxlod}(G_{i_k}) > C$. We use i_a to denote the iterative index

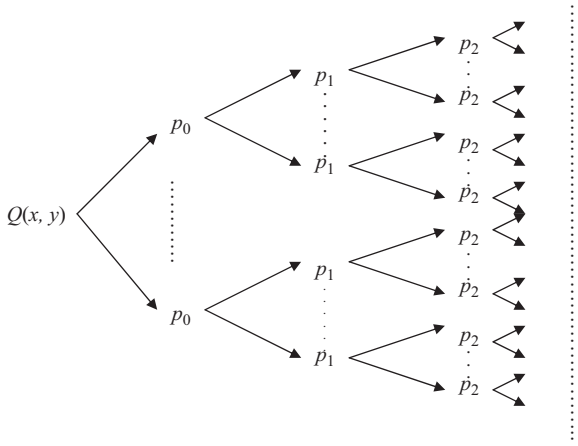

Fig. 1 *Coefficient deduction in Roth–Ruckenstein algorithm*

Table 4: Computational complexity for RS(63, 15)

m	$C + 1$	Finite-field arithmetic operations
1	64	1.75×10^5
2	190	4.57×10^6
4	631	1.67×10^8
6	1324	1.55×10^9

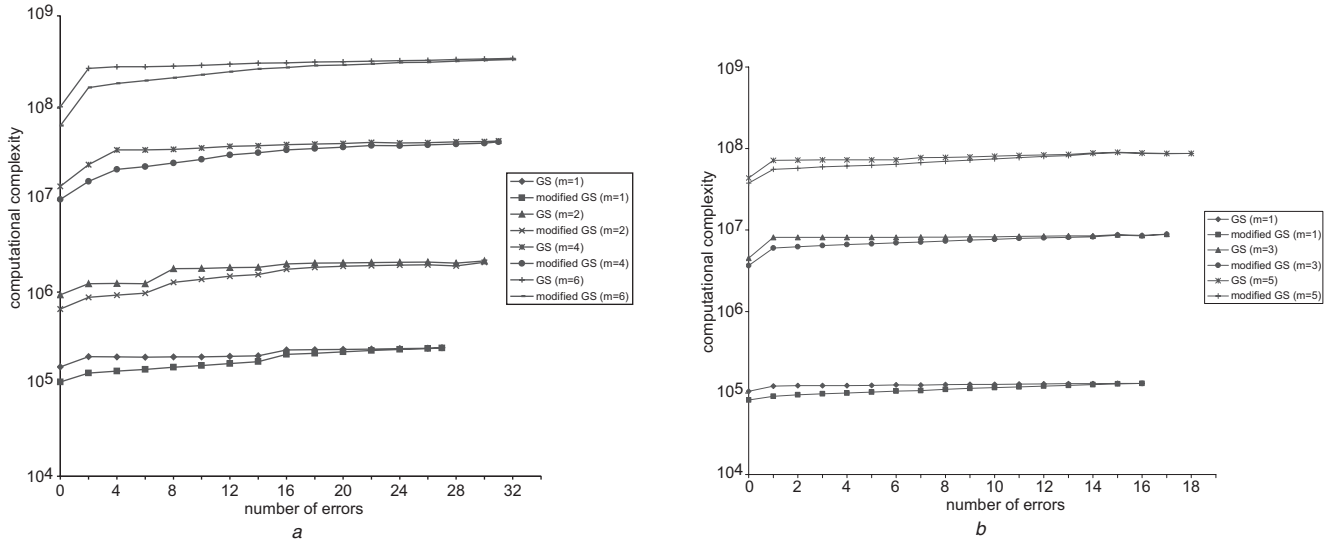

Fig. 2 *Computation complexity of GS decoding*
a RS(63, 15)
b RS(63, 31)

Table 5: Computational complexity for RS(63, 31)

m	$C+1$	Finite-field arithmetic operations
1	64	1.75×10^5
3	379	3.63×10^7
5	946	5.64×10^8

when the modification starts to act, which can be explained as

$$i_a = \{i_k | \max \mathrm{lod}(G_{i_k}) > C \text{ and } \max \mathrm{lod}(G_{i_k-1}) \leq C\} \quad (54)$$

i_a is error dependent. Under two different situations with error weight e_1 and e_2 (e_1, $e_2 \leq t_m$), decoding the same code with multiplicity m, we have

$$i_a(e_1) \leq i_a(e_2) \quad \text{if } e_1 \leq e_2 \quad (55)$$

which means the lower the error weight, the earlier the modification starts to act.

It has been observed that $g_{i_k,0}$ is always the first polynomial in the polynomial group to have leading order over C. Therefore analysing the leading order increase pattern of polynomial $g_{i_k,0}$ is useful to explain the modified algorithm's error dependent property (55). According to the polynomial property (33) and the leading order increase relationship (40), one can see that during the iterative process, $g_{i_k,0}$'s leading monomial M_L always satisfies $\deg_y(M_L) = 0$ and (40) can be simplified for $g_{i_k,0}$ as

$$\mathrm{lod}(g_{i_k+1,0}) = \mathrm{lod}(g_{i_k,0}) + \left\lfloor \frac{\deg_x g_{i_k,0}}{k-1} \right\rfloor + 1 \quad (56)$$

At the beginning of the iterative process, $\mathrm{lod}(g_{0,0}) = 0$. From (56), $g_{i_k,0}$ will be modified by (37) with $\mathrm{lod}(g_{i_k+1,0}) = \mathrm{lod}(g_{i_k,0}) + 1$ for $k-1$ times until $\deg_x(g_{i_k,0}) = k-1$. Following that, $g_{i_k,0}$ will again be modified by (37) with $\mathrm{lod}(g_{i_k+1,0}) = \mathrm{lod}(g_{i_k,0}) + 2$ for $k-1$ times until $\deg_x(g_{i_k,0}) = 2(k-1)$. This periodic process continues and the leading order of $g_{i_k,0}$ is accumulated as $1(k-1) + 2(k-1) + \cdots$. Term $g_{i_k,0}$ will be eliminated once its leading order is over C, therefore the periodic process will stop when $\mathrm{lod}(g_{i_k+1,0}) = \mathrm{lod}(g_{i_k,0}) + \lambda$, where λ is defined as

$$\lambda = \min\left\{ x | (k-1) \sum_{i=1}^{x} i > C \right\} \quad (57)$$

As there are $(k-1)(m+1)/2$ iterative steps for each of the periodic processes, under the zero error situation the upper bound for $i_a(0)$ can be defined as

$$i_a(0) \leq \frac{(k-1)(m+1)}{2} \lambda \quad (58)$$

Once $i_a(0)$ has been determined, $i_a(e)$ would always satisfy

$$i_a(e) \leq i_a(0) + e\binom{m+1}{2} \quad (59)$$

which means the lower the error weight, the earlier the modification starts to act and more computation can be reduced as a consequence. Tables 6 and 7 show some experimental data of $i_a(e)$ from the authors' implementation of RS(63, 15) and RS(63, 31), both of which reveal that (59) is being observed.

Table 6: $i_a(e)$ for RS(63, 15)

m	C	λ	$i_a(0)$ upper bound	$i_a(0)$	$i_a(1)$	$i_a(2)$	$i_a(3)$	$i_a(4)$	$i_a(5)$
1	63	3	42	36	37	38	39	40	41
2	189	5	105	99	102	105	108	111	114
4	630	10	350	350	318	328	338	348	358
6	1323	14	686	651	672	693	714	735	756

Table 7: $i_a(e)$ **for RS(63, 31)**

m	C	λ	$i_a(0)$ upper bound	$i_a(0)$	$i_a(1)$	$i_a(2)$	$i_a(3)$	$i_a(4)$	$i_a(5)$
1	63	2	60	47	48	49	50	51	52
3	378	5	300	271	277	283	289	295	301
5	945	8	720	673	688	703	718	733	748

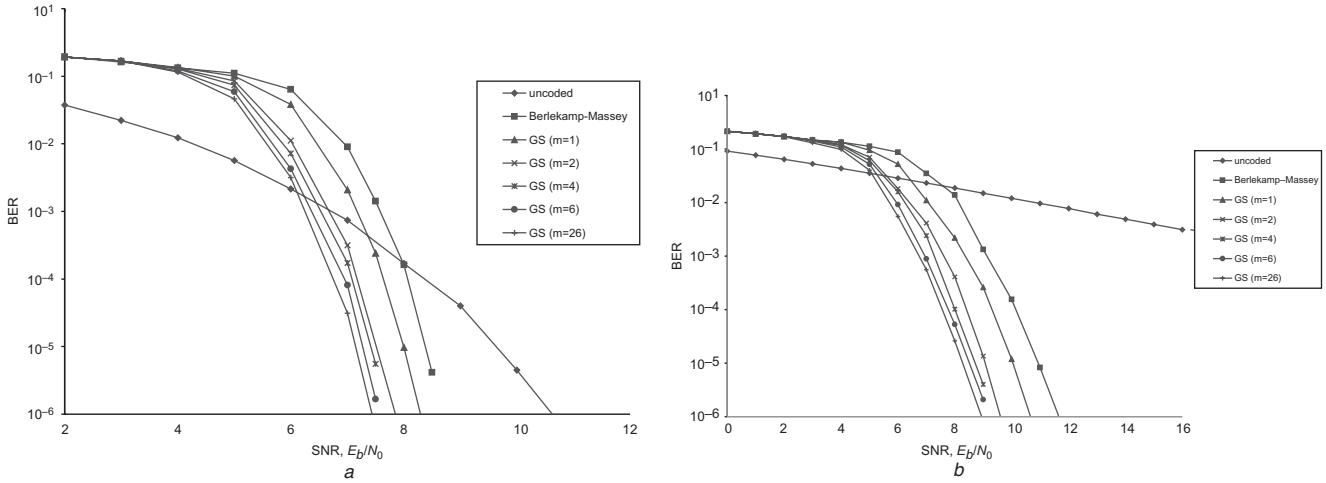

Fig. 3 *RS(63, 15) with QPSK performance*
a AWGN channels
b Rayleigh fading channels

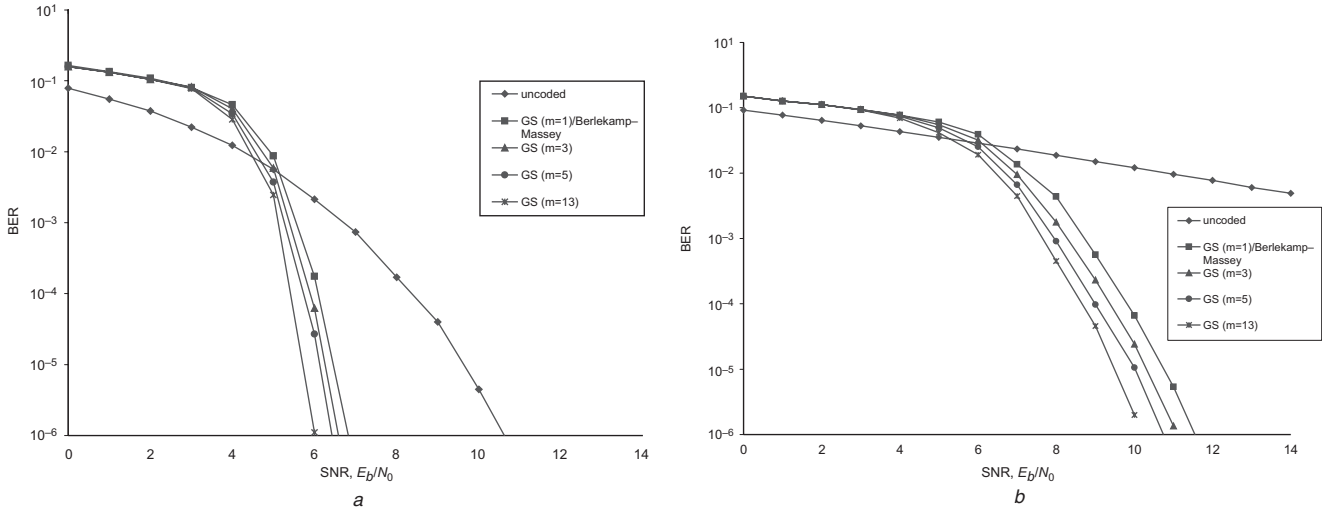

Fig. 4 *RS(63, 31) with QPSK performance*
a AWGN channels
b Rayleigh fading channels

6 Simulation results

The authors have developed a software platform using the C programming language for the GS algorithm with the complexity reduced modification and a few simulation results have been achieved. The performances of the two RS codes which are defined by Examples 1 and 2 are shown in Figs. 3 and 4. In the simulations the performance of a conventional unique RS decoding algorithm (Berlekamp–Massey algorithm) is used to compare with the GS algorithm, and the Rayleigh fading channels have variance $\sigma^2 = 0.5$ per dimension. Figs. 3*a* and *b* show the performance of RS(63, 15) over AWGN and Rayleigh fading channels. Over AWGN channels about 0.4–1.3 dB coding gain can be achieved at BER $= 10^{-5}$ with different multiplicity m, while over the Rayleigh fading channels the coding gain is about 1–2.8 dB. Figs. 4*a* and *b* show the performance of RS(63, 31). For this code the GS algorithm has no performance advantage with multiplicity $m = 1$. However, with multiplicity $m > 1$, at BER $= 10^{-5}$ it can achieve 0.2–0.8 dB coding gain over AWGN channels and 0.5–1.4 dB coding gain over Rayleigh fading channels.

7 Conclusions

This paper has explained in detail the GS algorithm for list decoding RS codes from an algebraic–geometric point of view. To improve the algorithm's decoding efficiency, a

novel modification to the interpolation part has been presented. This modification is based on eliminating unnecessary polynomials during the iterative interpolation process. According to the complexity analysis, the decoding complexity is error dependent and the modification can reduce the decoding complexity, especially for low error weight situations. Based on this modified GS algorithm, simulation results are presented showing the coding gains over a unique decoding algorithm with more significant gains for low rate codes and in a fading environment. It is very important that this performance advantage is still at the cost of high decoding complexity. Therefore further work such as the complexity reducing transformation for interpolation [22] need to be carried out to make the GS algorithm more efficient.

8 References

1 Elias, P.: 'List decoding for noisy channels'. Research Laboratory of Electronics, MIT, Technical Report 335, 1957

2 Wozencraft, J.M.: 'List decoding'. Quarterly Progress Report, 1958, Research Laboratory of Electronics, MIT 48, pp. 90–95

3 Sudan, M.: 'Decoding of Reed–Solomon codes beyond the error-correction bound', *J. Complexity*, 1997, **13**, (1), pp. 180–193

4 Reed, I.S., and Solomon, G.: 'Polynomial codes over certain finite fields', *J. Soc. Ind. Appl. Math.*, 1960, **8**, pp. 300–304

5 Guruswami, V., and Sudan, M.: 'Improved decoding of Reed–Solomon and algebraic–geometric codes', *IEEE Trans. Inf. Theory*, 1999, **45**, (6), pp. 1757–1767

6 Guruswami, V.: 'List decoding of error-correcting codes' (Springer-Verlag, Berlin, Heidelberg, 2004)

7 Kotter, R.: 'Fast generalized minimum-distance decoding of algebraic–geometric and Reed–Solomon codes', *IEEE Trans. Inf. Theory*, 1996, **42**, (3), pp. 721–736

8 Kotter, R., and Vardy, A.: 'Algebraic soft-decision decoding of Reed–Solomon codes', *IEEE Trans. Inf. Theory*, 2003, **49**, (11), pp. 2809–2825

9 Kotter, R.: 'On algebraic decoding of algebraic–geometric and cyclic codes'. Linkoping Studies in Science and Technology, No. 419, PhD dissertation, Department of Electrical Engineering, Linkoping University, 1996

10 Roth, R., and Ruckenstein, G.: 'Efficient decoding of Reed–Solomon codes beyond half the minimum distance', *IEEE Trans. Inf. Theory*, 2000, **46**, (1), pp. 246–257

11 McEliece, R.J.: 'The Guruswami–Sudan decoding algorithm for Reed–Solomon codes'. IPN Progress Report, 15 May 2003, pp. 42–153

12 Massey, J.L.: 'Shift register synthesis and BCH decoding', *IEEE Trans. Inf. Theory*, 1969, **IT-15**, (1), pp. 122–127

13 Sugiyama, Y., Kasahara, M., Hirasawa, S., and Namekawa, T.: 'A method for solving key equations for decoding Goppa codes', *Inf. Control*, 1975, **27**, pp. 87–99

14 Mandelbaum, D.M.: 'Decoding beyond the designed distance for certain algebraic codes', *Inf. Control*, 1977, **29**, pp. 207–228

15 Cassuto, Y., and Bruck, J.: 'On the average complexity of Reed–Solomon algebraic list decoder'. Proc. 8th IEE/IEEE Int. Symp. on Communication Theory and Applications, July 2005, Ambleside, UK, pp. 30–35

16 Cox, D., Little, J., and O'Shea, D.: 'Ideals, varieties, and algorithms' (Springer-Verlag, New York, 1992)

17 Moon, T.K.: 'Error correction coding – mathematical methods and algorithms' (Wiley-Interscience, 2005)

18 Forney, G.D.: 'Generalized minimum distance decoding', *IEEE Trans. Inf. Theory*, 1966, **IT-12**, (2), pp. 125–131

19 Pretzel, O.: 'Codes and algebraic curves' (Oxford, Clarendon Press, 1998)

20 Hasse, H.: 'Theorie der hoheren Differentiale in einem algebraishen Funcktionenkorper mit vollkommenem Konstantenkorper nei beliebeger Charakteristic', *J. Reine. Aug. Math.*, 1936, **175**, pp. 50–54

21 Huffman, W.C., and Pless, V.: 'Fundamentals of error-correcting codes' (Cambridge University Press, 2003)

22 Kotter, R., and Vardy, A.: 'A complexity reducing transformation in algebraic list decoding of Reed–Solomon codes'. Proc. IT Workshop ITW-2003, Paris, France, 31 March–4 April 2003