5

List Decoding

5.1 Introduction

In this chapter, we discuss an alternative decoding algorithm which produces a list of candidate messages instead of just one. This class of algorithm is known as a *List Decoding* algorithm and was developed by Elias [1, 2] and Wozencraft [3]. A list decoder has the advantage of being able to correct more errors than a conventional decoding algorithm that only returns a single decoded message, from now on known as a unique decoding algorithm. This is unfortunately at the expense of increased decoding complexity.

The chapter begins by introducing the Guruswami–Sudan algorithm [4] for the list decoding of Reed–Solomon codes. This algorithm has two processes: interpolation and factorization. It is shown that most of the decoding complexity of the list decoding algorithm is due to the interpolation process, but we present a method to reduce this complexity. This is followed by an explanation of the soft-decision list decoding of Reed–Solomon codes using the Kotter–Vardy algorithm [5].

We also describe how to modify the Guruswami–Sudan algorithm to list decode AG codes and again show we can still reduce the complexity of the interpolation process. The Kotter–Vardy algorithm is then extended for the soft-decision list decoding of AG codes. The performance of hard- and soft-decision list decoding of Reed–Solomon codes and AG codes is evaluated, and simulation result are presented showing the performance of both coding schemes on the AWGN and fading channels. The idea of list decoding is as follows: given a received word R, reconstruct a list of all code words with a distance τ to the received word R, in which τ can be greater than $\tau_{\text{unique}} = \lfloor \frac{d-1}{2} \rfloor$.

This idea is illustrated in Figure 5.1, where c_1, c_2 and c_3 are three independent code words at a distance d from each other. A received word r_1, which has distance less than τ_{unique} to code word c_1, can be decoded by the unique decoding algorithm, which results in c_1. However, for received word r_2, which has distance greater than τ_{unique} to any of the code words, the unique decoding algorithm will fail to decode it. But

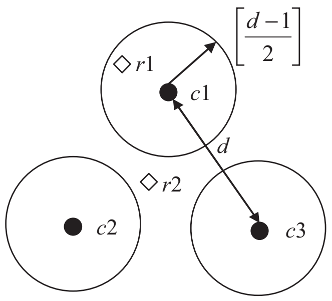


Figure 5.1 The idea of list decoding.

using the list decoding algorithm, a list of possible transmitted code words will be produced. For example, a decoded output list $\{c_1, c_2, c_3\}$ is produced by the decoder, then the code word that has the minimal distance to r_2 is chosen from the list and decoding is completed.

5.2 List Decoding of Reed–Solomon Codes Using the Guruswami–Sudan Algorithm

5.2.1 Weighted Degrees and Lexicographic Orders

To apply the list decoding algorithm we must first impose an ordering on the powers of a set of monomials. If we define the weighted (u, v)-degree of a monomial $x^a y^b$ by:

$$\deg_{u,v}(x^a y^b) = ua + vb \tag{5.1}$$

then we can order a sequence of monomials by their weighted degrees. The *lexicographic order* of the monomials is then defined as:

$$x^{a_1} y^{b_1} < x^{a_2} y^{b_2} \tag{5.2}$$

if $\deg_{u,v}(x^{a_1} y^{b_1}) < \deg_{u,v}(x^{a_2} y^{b_2})$ or $\deg_{u,v}(x^{a_1} y^{b_1}) = \deg_{u,v}(x^{a_2} y^{b_2})$ and $a_1 > a_2$. This has already been used in Chapter 4 to order the monomial basis used to construct algebraic–geometric codes. The lexicographic order of the weighted (1, 1)-degree of the sequence of monomials $x^a y^b$ is actually the Total Graduated Degree Order $<_\mathrm{T}$ defined in (4.9).

To list decode a (n, k) Reed–Solomon code we use the $(1, k-1)$-weighted degree. For example, in order to decode a (7, 5) RS code defined in GF(8), (1, 4)-lexicographic order is used. The generation of this order is shown in Table 5.1. The entries in Table 5.1a and 5.1b represent the (1, 4)-weighted degree and (1, 4)-lexicographic order of monomials M with x degree a and y degree b, respectively. Applying (5.1) with $u = 1$ and $v = 4$, we can generate the (1, 4)-weighted degree of monomials M shown by Table 5.1a. For example, for monomial x^2y, its (1, 4)-weighted degree is

Table 5.1 (a) (1, 4)-weighted degree of monomial $x^a y^b$; (b) (1, 4)-lexicographic order of monomial $x^a y^b$.

a/b	0	1	2	3	4	5	6	7	8	9	10	11	12	...
							(a)							
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							...						
⋮	⋮													

a/b	0	1	2	3	4	5	6	7	8	9	10	11	12	...
							(b)							
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							...						
⋮	⋮													

$\deg_{1,4}(x^2y) = 2 \cdot 1 + 4 = 6$, which is shown in Table 5.1a. Based on Table 5.1a, and applying the above lexicographic order rule, we can generate the (1, 4)-lexicographic order of monomials M shown in Table 5.1b and denoted as ord(M). From Table 5.1b, it can be observed that $x^4 < x^2y < y^2$, since ord(x^4) = 4, ord(x^2y) = 9 and ord(y^2) = 14.

With the definition of the weighted degree and order of a monomial, we can define the weighted degree of a nonzero bivariate polynomial in $F_q[x, y]$ as the weighted degree of its leading monomial, M_L. Any nonzero bivariate polynomial $Q(x, y)$ can be written as:

$$Q(x, y) = Q_0M_0 + Q_1M_1 + \cdots + Q_LM_L, \tag{5.3}$$

with bivariate monomial $M = x^ay^b$ ordered as $M_0 < M_1 < \cdots < M_L$, Q_0, $Q_1, \ldots, Q_L \in \mathrm{GF}(q)$ and $Q_L \neq 0$. The $(1, k-1)$-weighted degree of $Q(x, y)$ can be defined as:

$$\deg_{1,k-1}(Q(x, y)) = \deg_{1,k-1}(M_L) \tag{5.4}$$

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

$$\mathrm{lod}(Q(x, y)) = \mathrm{ord}(M_L) = L. \tag{5.5}$$

For example, given a polynomial $Q(x, y) = 1 + x^2 + x^2y + y^2$, applying the above (1, 4)-lexicographic order, it has leading monomial $M_L = y^2$. Therefore, $\deg_{1,4}(Q(x, y)) = \deg_{1,4}(y^2) = 8$ and $\mathrm{lod}(Q(x, y)) = \mathrm{ord}(y^2) = 14$. Consequently, any two nonzero polynomials Q and H ($Q, H \in F_q[x, y]$) can be compared with respect to their leading order:

$$Q \leq H, \text{ if } \mathrm{lod}(Q) \leq \mathrm{lod}(H) \tag{5.6}$$

$S_x(T)$ and $S_y(T)$ are denoted as the highest degree of x and y respectively under the $(1, k-1)$-lexicographic order such that:

$$S_x(T) = \max\{a : \mathrm{ord}(x^a y^0) \leq T\} \tag{5.7}$$

$$S_y(T) = \max\{b : \mathrm{ord}(x^0 y^b) \leq T\}, \tag{5.8}$$

where T is any nonnegative integer. The error-correction capability, τ_m, and the maximum number of candidate messages, l_m, in the output list with respect to a certain multiplicity, m, of the GS algorithm can be stated as [4]:

$$\tau_m = n - 1 - \left\lfloor \frac{S_x(C)}{m} \right\rfloor \tag{5.9}$$

$$l_m = S_y(C), \tag{5.10}$$

in which T in (5.7) and (5.8) is replaced by the nonnegative integer C, defined as:

$$C = n\binom{m+1}{2} = n\frac{(m+1)!}{(m+1-2)!2!} = \frac{n(m+1)m}{2} \tag{5.11}$$

C represents the number of iterations in the interpolation process, or alternatively the maximum number of Hasse derivative evaluations of an individual interpolated polynomial. These parameters will be proven in Section 5.2.5, when the factorization theorem is presented. τ_m and l_m grow monotonically with multiplicity m [4]:

$$\tau_{m_1} \leq \tau_{m_2} \tag{5.12}$$

$$l_{m_1} < l_{m_2}, \tag{5.13}$$

if $m_1 < m_2$.

5.2.2 Interpolation Theorem

From an algebraic geometric point of view, the interpolation process in the GS algorithm requires a hard decision to be made on the received word $(r_0, r_1, \ldots, r_{n-1})$ and the set of affine points corresponding to the location of each symbol in the received word. For Reed–Solomon codes, the affine points are simply the elements of the finite field over which it is defined. Each affine point and received symbol is paired up to form a set of interpolation points (x_i, r_i), where $i = 0, 1, \ldots, n-1$. The idea of interpolation is to generate a polynomial $Q(x, y)$ that intersects all these points a given number of times. The number of times $Q(x, y)$ intersects each point is called its *multiplicity*, denoted as m, and this parameter determines the error-correcting capability of the GS algorithm. The interpolation polynomial $Q(x, y)$ can be expressed as [4, 6]:

$$Q(x,y) = \sum_{a,b} Q_{ab} x^a y^b, \tag{5.14}$$

where $Q_{ab} \in \mathrm{GF}(q)$.

For $Q(x, y)$ to be satisfied at the point (x_i, r_i), $Q(x_i, r_i) = 0$. We can write x^a and y^b as:

$$x^a = (x - x_i + x_i)^a = \sum_{u \le a} \binom{a}{u} x_i^{a-u} (x - x_i)^u \quad \text{and} \quad y^b = (y - r_i + r_i)^b$$
$$= \sum_{v \le b} \binom{b}{v} r_i^{b-v} (y - r_i)^v$$

and therefore:

$$\begin{aligned} Q(x,y) &= \sum_{a,b} Q_{ab} \sum_{u \le a, v \le b} \binom{a}{u} x_i^{a-u} (x - x_i)^u \binom{b}{v} r_i^v (y - r_i)^v \\ &= \sum_{u,v} Q_{uv} (x - x_i)^u (y - r_i)^v \end{aligned} , \tag{5.15}$$

where:

$$Q_{uv}^{(x_i, r_i)} = \sum_{a \ge u, b \ge v} \binom{a}{u} \binom{b}{v} x_i^{a-u} r_i^{b-v}. \tag{5.16}$$

(5.16) is the (u, v)-*Hasse Derivative* evaluation at the point (x_i, r_i) of $Q(x, y)$ and defines the constraints on the coefficients of $Q(x, y)$ in order to have a zero of multiplicity m at that point.

Example 5.1: Given the polynomial $Q(x, y) = \alpha^5 + \alpha^5 x + y + xy$ defined over GF(8), we can use (5.16) to show that $Q(x, y)$ has a zero of multiplicity of at least $m = 2$ at the point $(1, \alpha^5)$. In order to have a zero of at least multiplicity $m = 2$, the (u, v)-Hasse Derivative at this point must be zero for all $u + v < m$, that is (u, v) can be (0, 0), (1, 0) and (0, 1).

$$\begin{aligned} Q_{00}^{(1,\alpha^5)} &= Q_{00} \binom{0}{0} \binom{0}{0} 1^{0-0} (\alpha^5)^{0-0} + Q_{10} \binom{1}{0} \binom{0}{0} 1^{1-0} (\alpha^5)^{0-0} \\ &\quad + Q_{01} \binom{0}{0} \binom{1}{0} 1^{0-0} (\alpha^5)^{1-0} + Q_{11} \binom{1}{0} \binom{1}{0} 1^{1-0} (\alpha^5)^{1-0} \\ &= 1 + 1 + \alpha^5 + \alpha^5 = 0 \\ Q_{10}^{(1,\alpha^5)} &= Q_{10} \binom{1}{1} \binom{0}{0} 1^{1-1} (\alpha^5)^{0-0} + Q_{11} \binom{1}{1} \binom{1}{0} 1^{1-1} (\alpha^5)^{1-0} = \alpha^5 + \alpha^5 = 0 \\ Q_{01}^{(1,\alpha^5)} &= Q_{01} \binom{0}{0} \binom{1}{1} 1^{0-0} (\alpha^5)^{1-1} + Q_{11} \binom{1}{0} \binom{1}{1} 1^{1-0} (\alpha^5)^{1-1} = 1 + 1 = 0. \end{aligned}$$

Therefore, since $Q_{00}^{(1,\alpha^5)} = Q_{10}^{(1,\alpha^5)} = Q_{01}^{(1,\alpha^5)} = 0$, $Q(x,y)$ must have a multiplicity of at least $m = 2$.

We denote the Hasse derivative operator by D_{uv} where [7]:

$$D_{uv}Q(x_i, r_i) = \sum_{a \geq u, b \geq v} Q_{ab} \binom{a}{u} \binom{b}{v} x_i^{a-u} r_i^{b-v}. \tag{5.17}$$

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

$$Q(x, y) = \min_{\text{lod}(Q)} \{Q(x, y) | D_{uv}Q(x_i, r_i) = 0 \quad \text{for} \quad i = 0, 1, \ldots, n \quad \text{and} \quad u + v < m\}. \tag{5.18}$$

5.2.3 *Iterative Polynomial Construction*

To find the interpolated polynomial of (5.18), an iterative polynomial construction algorithm [4, 8–12] is employed. In this algorithm a group of polynomials is initialized, tested by applying the Hasse derivative (5.16) and modified interactively. The interactive modification between two polynomials is based on the following two properties of the Hasse derivative [7, 10]:

Property 1: Linear functional of Hasse derivative If $H, Q \in F_q[x, y]$, δ_1 and $\delta_2 \in$ GF(q), then:

$$D(\delta_1 H + \delta_2 Q) = \delta_1 D(H) + \delta_2 D(Q). \tag{5.19}$$

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

$$[H, Q]_D = HD(Q) - QD(H). \tag{5.20}$$

If the Hasse derivative evaluation of $D(Q) = \delta_1$ and of $D(H) = \delta_2$ ($d_1, d_2 \neq 0$), based on Property 1, it is straightforward to prove that the Hasse derivative evaluation of (5.20) is zero, as follows:

$$D([H, Q]_D) = D(HD(Q) - QD(H)) = D(\delta_1 H - \delta_2 Q).$$

Using Property 1:

$$D(\delta_1 H - \delta_2 Q) = \delta_1 D(H) - \delta_2 D(Q) = \delta_1\delta_2 - \delta_2\delta_1 = 0.$$

Therefore:

$$D([H, Q]_D) = 0. \tag{5.21}$$

If lod(H) > lod(Q), the new constructed polynomial from (5.20) has leading order lod(H). Therefore, by performing the bilinear Hasse derivative over two polynomials with nonzero Hasse derivatives, we can reconstruct a polynomial which has a Hasse derivative of zero. Based on this principle, the implementation of an algorithm for interpolation will 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) Reed–Solomon code with multiplicity m, there are $C = n\binom{m+1}{2}$ iterations required to construct a polynomial defined by (5.18).

At the start of the algorithm, a group of polynomials is initialized as:

$$G_0 = \{Q_{0,j} = y^j_, j = 0, 1, \ldots, l_m\}, \tag{5.22}$$

where l_m is the maximum number of messages in the output list defined by (5.10). If M_L denotes the leading monomial of polynomial Q, it is important to point out that:

$$Q_{0,j} = \min\{Q(x, y) \in F_q[x, y] | \deg y(M_L) = j\}. \tag{5.23}$$

Let i_k denotes the iteration index of the algorithm, where $i_k = i\binom{m+1}{2} + r$, $i = 0, 1, \ldots, n-1$ and $r = 0, 1, \ldots, \binom{m+1}{2} - 1$. For iteration i_k of the algorithm, each polynomial $Q_{i_k,j}$ in group G_{i_k} is tested by (5.17), and the value of each after Hasse derivative evaluation is denoted by Δ_j as:

$$\Delta_j = D_{i_k}(Q_{i_k,j}). \tag{5.24}$$

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 (5.20). In order to construct a group of polynomials which satisfy:

$$\begin{aligned} Q_{i_k+1,j} = \min\big\{&Q \in F_q[x, y] | D_{i_k}(Q_{i_k+1,j}) = 0, D_{i_k-1}(Q_{i_k+1,j}) = 0, \ldots, \\ &D_0(Q_{i_k+1,j}) = 0, \text{ and } \deg y(M_L) = j\big\}, \end{aligned} \tag{5.25}$$

the minimal polynomial among those polynomials with $\Delta_j \neq 0$ is chosen. Denote the index of the minimal polynomial as j' and record it as Q':

$$j' = \text{index}(\min\{Q_{i_k,j} | \Delta_j = 0\}) \tag{5.26}$$

$$Q' = Q_{i_k,j'}. \tag{5.27}$$

For the remaining polynomials with $\Delta_j \neq 0$ but $j \neq j'$, (5.20) is used to modify them without the leading order being increased:

$$Q_{i_k+1,j'} = [Q_{i_k}, Q']_{D_{i_k}} = \Delta_j Q_{i_k,j} - Q'. \tag{5.28}$$

Based on (5.21), we know that $D_{i_k}(Q_{i_k+1,j}) = 0$. As $\text{lod}(Q_{i_k,j}) > \text{lod}(Q')$, it follows that $\text{lod}(Q_{i_k+1,j}) = \text{lod}(Q_{i_k,j})$.

Q' is modified by (5.20) with the leading order increasing:

$$Q_{i_k+1,j'} = [xQ', Q']_{D_{i_k}} = (x - x_i)Q', \tag{5.29}$$

where x_i is the x-coordinate of current interpolating point (x_i, r_i). $\Delta_{j^*} = D_{i_k}(Q\prime) \neq 0$ and so, as $D_{i_k}(xQ\prime) \neq 0$, $D_{i_k}(Q_{i_k+1,j\prime}) = 0$. As $\text{lod}(xQ') > \text{lod}(Q')$, $\text{lod}(Q_{i_k+1,j\prime}) = \text{lod}(xQ') > \text{lod}(Q_{i_k,j'})$. Therefore whenever (5.29) is performed, we have: $\text{lod}(Q_{i_k+1,j}) > \text{lod}(Q_{i_k,j})$.

After C iterative modifications, the minimal polynomial in group G_c is the interpolated polynomial that satisfies (5.18), and it is chosen to be factorized in the next step:

$$Q(x, y) = \min\{Q_{C,j} | Q_{C,j} \in G_C\}. \tag{5.30}$$

5.2.4 Complexity Reduced Modification

Based on the above analysis, it can be observed that when decoding a (n, k) Reed–Solomon code with multiplicity m, $l_m + 1$ bivariate polynomials are being interactively modified over C iterative steps in which Hasse derivative evaluation (5.17) and bilinear Hasse derivative modification (5.20) are being performed. This process has a complexity of approximately $O(n^2m^4)$ [4] and is responsible for the GS algorithm's high decoding complexity. Therefore, reducing the complexity of interpolation is essential to improving the algorithm's efficiency.

The leading order of the polynomial group G_{i_k} is defined as the minimal leading order (lod) among the group's polynomials [13]:

$$\text{lod}(G_{i_k}) = \min\{\text{lod}(Q_{i_k,j}) | Q_{i_k,j} \in G_{i_k}\}. \tag{5.31}$$

Based on the initialization defined in (5.22), the leading order of polynomial group G_0 is $\text{lod}(G_0) = \text{lod}(Q_{0,0}) = 0$. In the i_k modification, if no polynomial needs to be modified then the polynomial group is unchanged; $\text{lod}(G_{i_k+1}) = \text{lod}(G_{i_k})$. When a polynomial needs to be modified, (5.29) must be used. If M_L is the leading monomial of Q^*, we have [13]:

$$\text{lod}(xQ^*) = \text{lod}(Q^*) + \left\lfloor \frac{\deg_x Q^*}{k-1} \right\rfloor + \deg_y(M_L) + 1 \tag{5.32}$$

when k is the dimension of the code. Based on (5.32), it can be seen that $\text{lod}(G_{i_k})$ will be increased if Q^* 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:

$$\text{lod}(G_{i_k}) \leq i_k. \tag{5.33}$$

Based on (5.33), after C iterative steps we have:

$$\text{lod}(G_C) \leq C. \tag{5.34}$$

From (5.30) we know that only the minimal polynomial is chosen from the polynomial group G_C as $Q(x, y) = \{Q_{c,j} | Q_{c,j} \in G_c \text{ and } \text{lod}(Q_{c,j}) = \text{lod}(G_c)\}$, therefore:

$$\text{lod}(Q(x, y)) \leq C, \tag{5.35}$$

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 for $Q(x, y)$. Therefore, during the iterative process, we can modify the group of polynomials by eliminating those with leading order greater than C, as [13]:

$$G_{i_k} = \{Q_{i_k,j} | \text{lod}(Q_{i_k,j}) = C\}. \tag{5.36}$$

We now prove this modification will not affect the final result. In iteration i_k, if there is a polynomial $Q_{i_k,j}$ with $\text{lod}(Q_{i_k,j}) > C$, it may be modified by either (5.28) or (5.29), which will result in its leading order being unchanged or increased. Therefore, at the end $\text{lod}(Q_{c,j}) > C$, and based on (5.35) it cannot be $Q(x, y)$. However, if $Q_{i_k,j}$ is the minimal polynomial defined by (5.27), this implies that those polynomials with leading order less than C do not need to be modified. If $Q_{i_k,j}$ is not the minimal polynomial defined by (5.27), $Q_{i_k,j}$ will not be chosen to perform bilinear Hasse derivative (5.28) with other polynomials. Therefore, $Q(x, y)$ has no information introduced from $Q_{i_k,j}$, since $\text{lod}(Q_{i_k,j}) > C$. As a result, eliminating the polynomials with leading order greater than C will not affect the final outcome.

This complexity modification scheme can be generally applied to the iterative interpolation process, for example to soft-decision list decoding of Reed–Solomon codes and hard/soft-decision list decoding of Hermitian codes. Based on the total number of iterations C for interpolation, the interpolated polynomial's leading order always satisfies $\text{lod}(Q(x, y)) \leq C$. It implies that those polynomials in the group G can be eliminated once their leading order is greater than C.

This modification can reduce some unnecessary computation in terms of avoiding Hasse derivative evaluation (5.24) and bilinear Hasse derivative modification (5.28) and (5.29) of polynomials with leading order over C. Based on the above analysis, the modified interpolation process can be summarized as:

Algorithm 5.1: Interpolation for list decoding a (*n*, *k*) Reed–Solomon code [13, 14]

```
1. Initialize a group of polynomials by (5.22), and set the index of the interpolated
   point $i = 0$.
2. Set interpolation point to $(x_i, r_i)$.
3. For each $(u, v)$ where $u + v < m$.
   {
4. Modify the polynomial group by (5.36).
5. Perform Hasse derivative evaluation (5.24) for each polynomial in the group.
6. If all the polynomials' Hasse derivative evaluations are zero, choose another pair
   $(u, v)$ and go to step 3.
7. Find the minimal polynomial defined by (5.26) and (5.27).
8. For the minimal polynomial, modify it by (5.29). For the other polynomials with
   nonzero Hasse derivative evaluation, modify them by (5.28).
   }
9. $i = i + 1$.
10. If $i = n$, stop the process and choose $Q(x, y)$ defined by (5.30). Else go to step 2.
```

Example 5.2 shows the modified interpolation process.

Example 5.2: Decoding the (7, 2) Reed–Solomon code defined over GF(8) with multiplicity m = 2 As $C = 7\binom{3}{1} = 21$, based on (5.9) and (5.10) we have $\tau_2 = 3$ and $l_2 = 5$. The transmitted code word is generated by evaluating the message polynomial $f(x) = \alpha + \alpha^6 x$ over the set of points $x = (1, \alpha, \alpha^3, \alpha^2, \alpha^6, \alpha^4, \alpha^5)$, and the corresponding received word is $R = (\alpha^5, \alpha^3, \alpha^4, 0, \alpha^6, \alpha^2, \alpha^2)$, where α is a primitive element in GF(8) satisfying $\alpha^3 + \alpha + 1 = 0$. Construct a bivariate polynomial that has a zero of multiplicity $m = 2$ over the n points $(x_i, r_i)|_{i=0}^{n-1}$.

At the beginning, six polynomials are initialized as:

$Q_{0,0} = 1$, $Q_{0,1} = y$, $Q_{0,2} = y^2$, $Q_{0,3} = y^3$, $Q_{0,4} = y^4$ and $Q_{0,5} = y^5$. Their leading orders are $\text{lod}(Q_{0,0}) = 0$, $\text{lod}(Q_{0,1}) = 2$, $\text{lod}(Q_{0,2}) = 5$, $\text{lod}(Q_{0,3}) = 9$, $\text{lod}(Q_{0,4}) = 14$ and $\text{lod}(Q_{0,5}) = 20$ respectively. $\text{lod}(G_0) = \text{lod}(Q_{0,0}) = 0$.

When $i = 0$ and $(u, v) = (0, 0)$, $i_k = 0$. No polynomial is eliminated from the group G_0.

Perform Hasse derivative evaluation for each of the polynomials in G_0 as:

$$\Delta_0 = D_{(0,0)}^{(1,\alpha^5)} Q_{0,0} = 1,\ \Delta_1 = D_{(0,0)}^{(1,\alpha^5)} Q_{0,1} = \alpha^5$$
$$\Delta_2 = D_{(0,0)}^{(1,\alpha^5)} Q_{0,2} = \alpha^3,\ \Delta_3 = D_{(0,0)}^{(1,\alpha^5)} Q_{0,3} = \alpha$$
$$\Delta_4 = D_{(0,0)}^{(1,\alpha^5)} Q_{0,4} = \alpha^7,\ \Delta_5 = D_{(0,0)}^{(1,\alpha^5)} Q_{0,5} = \alpha.$$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$$j' = 0 \quad \text{and} \quad Q' = Q_{0,0}.$$

Modify polynomials in G_0 with $\Delta_j \neq 0$ as:

$$Q_{1,0} = \Delta_0(x - x_0)Q' = 1 + x, \text{ and } \mathrm{lod}(Q_{1,0}) = 1$$
$$Q_{1,1} = \Delta_0 Q_{0,1} - \Delta_1 Q' = \alpha^5 + y \text{ and } \mathrm{lod}(Q_{1,1}) = 2$$
$$Q_{1,2} = \Delta_0 Q_{0,2} - \Delta_2 Q' = \alpha^3 + y^2 \text{ and } \mathrm{lod}(Q_{1,2}) = 5$$
$$Q_{1,3} = \Delta_0 Q_{0,3} - \Delta_3 Q' = \alpha + y^3 \text{ and } \mathrm{lod}(Q_{1,3}) = 9$$
$$Q_{1,4} = \Delta_0 Q_{0,4} - \Delta_4 Q' = \alpha^6 + y^4 \text{ and } \mathrm{lod}(Q_{1,4}) = 14$$
$$Q_{1,5} = \Delta_0 Q_{0,5} - \Delta_5 Q' = \alpha^4 + y^5 \text{ and } \mathrm{lod}(Q_{1,5}) = 20$$
$$\mathrm{lod}(G_1) = \mathrm{lod}(Q_{1,0}) = 1.$$

When $i = 0$ and $(u, v) = (0, 1)$, $i_k = 1$. No polynomial is eliminated from the group G_1.

Perform Hasse derivative evaluation for each of the polynomial in G_1 as:

$$\Delta_0 = D_{(0,1)}^{(1,\alpha^5)}(Q_{1,0}) = 0.\ \Delta_1 = D_{(0,1)}^{(1,\alpha^5)}(Q_{1,1}) = 1$$
$$\Delta_2 = D_{(0,1)}^{(1,\alpha^5)}(Q_{1,2}) = 0.\ \Delta_3 = D_{(0,1)}^{(1,\alpha^5)}(Q_{1,3}) = \alpha^3$$
$$\Delta_4 = D_{(0,1)}^{(1,\alpha^5)}(Q_{1,4}) = 0.\ \Delta_5 = D_{(0,1)}^{(1,\alpha^5)}(Q_{1,5}) = \alpha^6.$$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$$j' = 1 \quad \text{and} \quad Q' = Q_{1,1}.$$

As $\Delta_0 = \Delta_2 = \Delta_4 = 0$:

$$Q_{2,0} = Q_{1,0} = 1 + x, \text{ and } \mathrm{lod}(Q_{2,0}) = 1$$
$$Q_{2,2} = Q_{1,2} = \sigma^3 + y^2, \text{ and } \mathrm{lod}(Q_{2,2}) = 5$$
$$Q_{2,4} = Q_{1,4} = \sigma^6 + y^4, \text{ and } \mathrm{lod}(Q_{2,4}) = 14.$$

Modify polynomials in G_1 with $\Delta_j \neq 0$ as:

$$\begin{aligned} Q_{2,1} &= \Delta_1(x - x_0)Q' = \alpha^5 + \alpha^5 x + y(1 + x),\ \mathrm{lod}(Q_{2,1}) = 4 \\ Q_{2,3} &= \Delta_1 Q_{1,3} - \Delta_3 Q' = \alpha^3 y + y^3,\ \mathrm{lod}(Q_{2,3}) = 9 \\ Q_{2,5} &= \Delta_1 Q_{2,5} - \Delta_5 Q' = \alpha^6 y + y^5,\ \mathrm{lod}(Q_{2,5}) = 20 \\ \mathrm{lod}(G_2) &= \mathrm{lod}(Q_{2,0}) = 1. \end{aligned}$$

When $i = 0$ and $(u, v) = (1, 0)$, $i_k = 2$. No polynomial is eliminated from the group G_2.

Perform Hasse derivative evaluation for each of the polynomial in G_2 as:

$$\begin{aligned} \Delta_0 &= D^{(1,\alpha^5)}_{(1,0)}(Q_{2,0}) = 1.\ \Delta_1 = D^{(1,\alpha^5)}_{(1,0)}(Q_{2,1}) = 0 \\ \Delta_2 &= D^{(1,\alpha^5)}_{(1,0)}(Q_{2,2}) = 0.\ \Delta_3 = D^{(1,\alpha^5)}_{(1,0)}(Q_{2,3}) = 0 \\ \Delta_4 &= D^{(1,\alpha^5)}_{(1,0)}(Q_{2,4}) = 0.\ \Delta_5 = D^{(1,\alpha^5)}_{(1,0)}(Q_{2,5}) = 0. \end{aligned}$$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$$j' = 0 \text{ and } Q' = Q_{2,0}.$$

As $\Delta_1 = \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = 0$:

$$\begin{aligned} Q_{3,1} &= Q_{2,1} = \alpha^5 + \alpha^5 x + y(1 + x),\ \mathrm{lod}(Q_{3,1}) = 4 \\ Q_{3,2} &= Q_{2,2} = \alpha^3 + y^2,\ \text{and } \mathrm{lod}(Q_{3,2}) = 5 \\ Q_{3,3} &= Q_{2,3} = \alpha^3 y + y^3,\ \mathrm{lod}(Q_{3,3}) = 9 \\ Q_{3,4} &= Q_{2,4} = \alpha^6 + y^4,\ \text{and } \mathrm{lod}(Q_{3,4}) = 14 \\ Q_{3,5} &= Q_{2,5} = \alpha^6 y + y^5,\ \mathrm{lod}(Q_{3,5}) = 20. \end{aligned}$$

Modify polynomials in G_2 with $\Delta_j \neq 0$ as:

$$\begin{aligned} Q_{3,0} &= \Delta_0(x - x_0)Q' = 1 + x^2,\ \mathrm{lod}(Q_{3,0}) = 3 \\ \mathrm{lod}(G_3) &= \mathrm{lod}(Q_{3,0}) = 3. \end{aligned}$$

Based on the same process, interpolation is run through all the rest of the points (x_i, r_i) ($i = 1$ to 6). In order to illustrate the complexity reducing modification scheme, Table 5.2 shows the whole iterative process with respect to the polynomials' leading order.

From Table 5.2 we can see that the modified algorithm starts to take action at $i_k = 10$ when there is at least one polynomial with leading order over 21 and eliminating such polynomials will not affect the final outcome. At the end, both the original and the modified GS algorithm produce the same result: $Q(x, y) = \min\{G_{21}\} = Q_{21,2} = 1 + \alpha^4 x^2 + \alpha^2 x^4 + y^2(\alpha^5 + \alpha^4 x^2)$. From this example we can see that more computation can be reduced if the modified algorithm starts to take action at earlier steps.

Table 5.2 Iterative process of Example 5.2.

i (i_k)	0 (0)	0 (1)	0 (2)	1 (3)	1 (4)	1 (5)	2 (6)	2 (7)	2 (8)	3 (9)
lod($Q_{i_k,0}$)	0	1	1	3	6	6	10	15	15	21
lod($Q_{i_k,1}$)	2	2	4	4	4	7	7	7	11	11
lod($Q_{i_k,2}$)	5	5	5	5	5	5	5	5	5	5
lod($Q_{i_k,3}$)	9	9	9	9	9	9	9	9	9	9
lod($Q_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14
lod($Q_{i_k,5}$)	20	20	20	20	20	20	20	20	20	20
lod(G_{i_k})	0	1	1	3	4	5	5	5	5	5

Original GS

i (i_k)	3 (10)	3 (11)	4 (12)	4 (13)	4 (14)	5 (15)	5 (16)	5 (17)	6 (18)	6 (19)	6 (20)	7 (21)
lod($Q_{i_k,0}$)	28	28	36	45	45	55	55	55	55	66	66	78
lod($Q_{i_k,1}$)	11	16	16	16	22	22	22	22	22	22	29	29
lod($Q_{i_k,2}$)	5	5	5	5	5	5	8	8	12	12	12	$\boxed{12}$
lod($Q_{i_k,3}$)	9	9	9	9	9	9	9	13	13	13	13	13
lod($Q_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14	14	14
lod($Q_{i_k,5}$)	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

Modified GS

i (i_k)	3 (10)	3 (11)	4 (12)	4 (13)	4 (14)	5 (15)	5 (16)	5 (17)	6 (18)	6 (19)	6 (20)	7 (21)
lod($Q_{i_k,0}$)	—	—	—	—	—	—	—	—	—	—	—	—
lod($Q_{i_k,1}$)	11	16	16	16	—	—	—	—	—	—	—	—
lod($Q_{i_k,2}$)	5	5	5	5	5	5	8	8	12	12	12	$\boxed{12}$
lod($Q_{i_k,3}$)	9	9	9	9	9	9	9	13	13	13	13	13
lod($Q_{i_k,4}$)	14	14	14	14	14	14	14	14	14	14	14	14
lod($Q_{i_k,5}$)	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 the corresponding polynomial is eliminated; □ means the corresponding polynomial is chosen as $Q(x, y)$.

5.2.5 Factorization

In this section, the factorization theorem is explained, followed by a detailed description of an efficient algorithm known as Roth–Ruckenstein's algorithm [15].

As mentioned in Section 5.2.2, given the interpolated polynomial $Q(x, y)$, the transmitted message polynomial $f(x)$ can be found by determining $Q(x, y)$'s y roots.

Lemma 5.1 *If $Q(x, y)$ has a zero of multiplicity at least m over (x_i, r_i) and $p(x)$ is a polynomial in $F_q[x^{k-1}]$ that $p(x_i) = r_i$, then $(x - x_i)^m | Q(x, p(x))$ [6].*

Define $\Lambda(p, R)$ as the number of symbols in received word R that satisfy $p(x_i) = r_i$ as:

$$\Lambda(p, R) = |\{i : p(x_i) = r_i, i = 0, 1, \ldots, n-1\}|. \tag{5.37}$$

Lemma 5.2 *$p(x)$ is a polynomial in $F_q[x^{k-1}]$ and $p(x_i) = r_i$ for at least $\Lambda(p, R)$ values. If $m\ \Lambda(p, R) > \deg_{1,k-1}(Q(x, y))$ then $y - p(x)|Q(x, y)$, or $Q(x, p(x)) = 0$ [6].*

Based on Lemma 5.1, if $p(x_i) = r_i$ then $(x - x_i)^m|Q(x, y)$. If S is the set of i that satisfies $p(x_i) = r_i$. as $|S| = \Lambda(p, R)$ then $\prod_{i\in S}(x - x_i)^m|Q(x, p(x))$. Assume $g_1(x) = \prod_{i\in S}(x - x_i)^m$ and $g_2(x) = Q(x, p(x))$; therefore $g_1(x)|g_2(x)$. It is obvious that $g_1(x)$ has x-degree $m\ \Lambda(p, R)$ and $g_2(x)$ has x-degree equal to $\deg_{1,k-1}Q(x, y)$. If $m\ \Lambda(p, R) > \deg_{1,k-1} Q(x, y)$ and $g_1(x)|g_2(x)$, the only solution for these two preconditions is $g_2(x) = 0$. Therefore, if $m\ (p, R) > \deg_{1,k-1}(Q(x, y))$, $Q(x, p(x)) = 0$, or equivalently, $y - p(x)|Q(x, y)$.

As $Q(x, y)$ is the interpolated polynomial from the last step, according to (5.35), $\mathrm{lod}(Q) \leq C$. Based on (5.9), $\deg_{1,k-1}(Q(x, y)) \leq S_x(C)$. If $m\ \Lambda(f, R) \geq S_x(C)$ then $m\ \Lambda(f, R) \geq \deg_{1,k-1}(Q(x, y))$. Based on Lemma 5.2, if $\Lambda(f, R) \geq 1 + \left\lfloor \frac{S_x(C)}{m} \right\rfloor$ then the transmitted message polynomial $f(x)$ can be found by factorizing $Q(x, y)$. As $\Lambda(f, R)$ represents the number of points that satisfy $r_i = f(x_i) = c_i$, those points that do not satisfy this equation are where the errors are located.

Therefore, the error-correction capability of the GS algorithm is $\tau_m = n - \left\lfloor \frac{S_x(C)}{m} \right\rfloor - 1$, which is defined by (5.10). Under $(1, k - 1)$-lexicographic order, x^0y^j is the maximal monomial with weighted 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, $\max\{\deg_y Q(x, y)\} \leq S_y(C)$. As the factorization output list contains the y-roots of $Q(x, y)$, and the number of y-roots of $Q(x, y)$ should not exceed its y-degree, the maximal number of candidate messages in the output list is $l_m = S_y(C)$, which is defined by (5.11).

5.2.6 Recursive Coefficient Search

To find the y-roots of the interpolated polynomial $Q(x, y)$, Roth and Ruckenstein [15] introduced an efficient algorithm for factorizing these bivariate polynomials.

In general, the factorization output $p(x) \in F_q[x^{k-1}]$ can be expressed in the form of:

$$p(x) = p_0 + p_1x + \cdots + p_{k-1}x^{k-1}, \tag{5.38}$$

where $p_0, p_1, \ldots, p_{k-1} \in \mathrm{GF}(q)$. In order to find the polynomials $p(x)$, we must determine their coefficients $p_0, p_1, \ldots, p_{k-1}$, respectively. The idea of Roth–Ruckenstein's algorithm is to recursively 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 [15]:

$$Q^*(x, y) = \frac{Q(x, y)}{x^h}. \tag{5.39}$$

If we denote $p_0 = p(x)$ and $Q_0(x, y) = Q^*(x, y)$, where $Q(x, y)$ is the new interpolated polynomial (5.30), we can define the recursive updated polynomials $p_s(x)$ and $Q_s(x, y)$, where $s \geq 1$, as [15]:

$$p_s(x) = \frac{p_{s-1}(x) - p_{s-1}(0)}{x} = p_s + \cdots + p_{k-1}x^{k-1-s}, \quad (s = k - 1). \quad (5.40)$$

$$Q_s(x, y) = Q^*_{s-1}(x, xy + p_{s-1}). \quad (5.41)$$

Lemma 5.3 *With* $p_s(x)$ *and* $Q_s(x, y)$ *defined by (5.40) and (5.41), when* $s \geq 1$, $(y - p(x))|Q(x, y)$ *if and only if* $(y - p_s(x))|Q_s(x, y)$ *[4].*

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 $Q(x, p(x)) = 0$, $p_0(x)$ should satisfy $Q_0(x, p_0(x)) = 0$. When $x = 0$, $Q_0(0, p_0(0)) = 0$. According to (5.38), $p_0(0) = p_0$, 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 can deduce further to find the rest of p_s $(s = 1, \ldots, k - 1)$ based on the recursive transformation (5.40) and (5.41).

Assume that after $s - 1$ deductions, polynomial $p_{s-1}(x)$ is the y-root of $Q_{s-1}(x, y)$. Based on (5.40), $p_{s-1}(0) = p_{s-1}$ 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 can find p_s. As $Q_{s-1}(x, p_{s-1}(x)) = 0$, $(y - p_{s-1}(x))|Q_{s-1}(x, y)$. 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 (5.40), $xy + p_{s-1} - p_{s-1}(x) = xy - xp_s(x)$. 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, as shown in Figure 5.2. 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 (5.40), 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 5.3, $(y - p(x))|Q(x, y)$ and $p(x)$ is found.

Based on the above analysis, the factorization process can be summarized as [4, 15]:

Algorithm 5.2: Factorization of list decoding a (*n, k*) Reed–Solomon code [15]

1. Initialize $Q_0(x, y) = Q^*(x, y)$, $s = 0$.
2. Find roots p_s of $Q_s(0, y) = 0$.
3. For each p_s, perform Q transformation (5.41) to calculate $Q_{s+1}(x, y)$.
4. $s = s + 1$.
5. 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 $p_{s-1}, \ldots, p_1, p_0$.

Example 5.3 demonstrates Roth–Ruckenstein's algorithm.

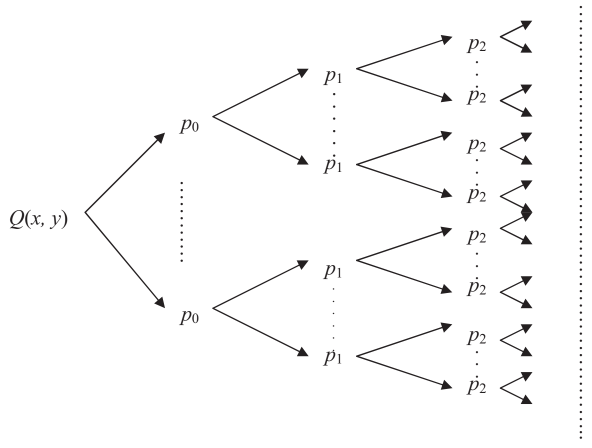


Figure 5.2 Coefficients deduction in Roth–Ruckenstein's algorithm.

Example 5.3: Based on polynomial $Q(x, y) = 1 + \alpha^4x^2 + \alpha^2x^4 + y^2(\alpha^5 + \alpha^4x^2)$, which is the interpolation result of Example 5.2, determine the factorization output list L using Roth–Ruckenstein's algorithm.

Initialize $Q_0(x, y) = Q^*(x, y) = 1 + \alpha^4x^2 + \alpha^2x^4 + y^2(\alpha^5 + \alpha^4x^2)$ and $s = 0$.

$Q_0(0, y) = 1 + \alpha^5y^2$ and $p_0 = \alpha$ is the root of $Q_0(0, y) = 0$.

For $p_0 = \alpha$, generate $Q_1(x, y) = Q_0(x, xy + p_0)$:

$$\begin{aligned} Q_0(x, xy + \alpha) &= 1 + \alpha^4x^2 + \alpha^2x^4 + (xy + \alpha)^2(\alpha^5 + \alpha^4x^2) \\ &= 1 + \alpha^4x^2 + \alpha^2x^4 + (x^2y^2 + \alpha^2)(\alpha^5 + \alpha^4x^2) \\ &= 1 + \alpha^4x^2 + \alpha^2x^4 + \alpha^5x^2y^2 + \alpha^4x^4y^2 + 1 + \alpha^6x^2 \\ &= \alpha^3x^2 + \alpha^2x^4 + y^2(\alpha^5x^2 + \alpha^4x^4). \end{aligned}$$

Now, from (5.39), we can see that x^2 is the highest power of x that divides $Q_0(x, xy + \alpha)$, so:

$$Q_0^*(x, xy + \alpha) = \frac{Q_0(x, xy + \alpha)}{x^h} = \frac{Q_0(x, xy + \alpha)}{x^2} = \alpha^3 + \alpha^2x^2 + y^2(\alpha^5 + \alpha^4x^2).$$

$s = s + 1 = 1$. As $s < k$, go to step 2 of Algorithm 5.2.

$Q_1(0, y) = \alpha^3 + \alpha^5y^2$ and $p_1 = \alpha^6$ is a root of $Q_1(0, y) = 0$.

For $p_1 = \alpha^6$, generate $Q_2(x, y) = Q_1^*(x, xy + p_1) = y^2(\alpha^5 + \alpha^4x^2)$. $s = s + 1 = 2$. As $s = k$ and $Q_2(x, 0) = 0$, trace this route to find its output $p_0 = \alpha$ and $p_1 = \alpha^6$.

As a result, factorization output list $L = \{p(x) = \alpha + \alpha^6x\}$. From Example 5.2, $p(x)$ matches the transmitted message polynomial $f(x)$.

Example 5.4: Factorizing an interpolation polynomial containing two messages Using the same (7, 2) Reed–Solomon code as in Example 5.3, assume that the interpolated polynomial is $Q(x, y) = \alpha x + \alpha^6 x^2 + y(\alpha^3 + \alpha^3 x) + \alpha^2 y^2$.

For $s = 0$, $Q_0(x, y) = Q^*(x, y) = \alpha x + \alpha^6 x^2 + y(\alpha^3 + \alpha^3 x) + \alpha^2 y^2$.

Setting $x = 0$, $Q_0(0, y) = \alpha^3 y + \alpha^2 y^2$ and has two roots, $p_0 = 0$ and α.

Taking $p_0 = 0$:

$$Q_1(x, y) = Q_0^*(x, xy + 0) = \alpha + \alpha^6 x + y(\alpha^3 + \alpha^3 x) + \alpha^2 x y^2.$$

$s = s + 1 = 1$. As $s < k = 2$ we go back to step 2 of Algorithm 5.2.

For $s = 1$, $Q_1(0, y) = \alpha + \alpha^3 y$, which has one root, $p_1 = \alpha^5$.

Taking $p_1 = \alpha^5$:

$$Q_2(x, y) = Q_1^*(x, xy + \alpha^5) = y(\alpha^3 + \alpha^3 x) + \alpha^2 x^2 y^2$$

$s = s + 1 = 2$. Now $s = k$ and $Q_2(x, 0)$, so this route is terminated and the message is $p(x) = 0 + \alpha^5 x = \alpha^5 x$.

However, we must now determine whether there is another message contained within the interpolation polynomial and so we now take the second root of $Q_0(0, y)$.

Taking $p_0 = \alpha$:

$$Q_1(x, y) = Q_0^*(x, xy + \alpha) = \alpha^2 + \alpha^6 x + y(\alpha^3 + \alpha^3 x) + \alpha^2 x y^2$$

$s = s + 1 = 1$. As $s < k = 2$ we go back to step 2 of Algorithm 5.2.

For $s = 1$, $Q_1(0, y) = \alpha^2 + \alpha^3 y$, which has one root, $p_1 = \alpha^6$.

Taking $p_1 = \alpha^6$:

$$Q_2(x, y) = Q_1^*(x, xy + \alpha^6) = y(\alpha^3 + \alpha^3 x) + \alpha^2 x^2 y^2$$

$s = s + 1 = 2$. Now $s = k$ and $Q_2(x, 0)$, so this route is terminated and the message is $p(x) = \alpha + \alpha^6 x$.

The roots of $Q(x, y)$ are shown graphically in Figure 5.3. To complete the decoding procedure, each message would be re-encoded and the code word with the minimum Hamming distance from the received word would be chosen, along with its corresponding message.

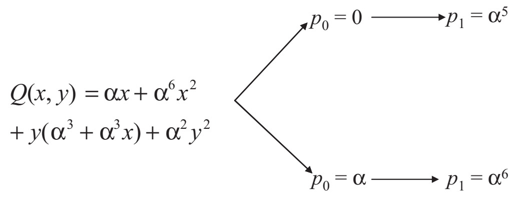

$Q(x, y) = \alpha x + \alpha^6 x^2 + y(\alpha^3 + \alpha^3 x) + \alpha^2 y^2$
$p_0 = 0$
$p_1 = \alpha^5$
$p_0 = \alpha$
$p_1 = \alpha^6$

Figure 5.3 Two messages from Roth–Ruckenstein's algorithm for Example 5.3.

5.3 Soft-Decision List Decoding of Reed–Solomon Codes Using the Kötter–Vardy Algorithm

Increases in the performance of a Reed–Solomon code can be achieved by taking into consideration the soft values from the output of the demodulator. These can be used to give a measure of the reliability of each symbol in the received word. To modify the Guruswami–Sudan algorithm to use reliability values instead of hard values, there needs to be a method of mapping reliability values to multiplicity values. Kotter and Vardy presented a seminal paper in 2003 [5] which allowed the algebraic soft-decision decoding of Reed–Solomon codes. In this paper, they also gave an algorithm to convert the reliability of each symbol in the received word to a multiplicity value of each point (x_j, ρ_i) for the interpolation process, where ρ_i can be one of q finite field elements in $\mathrm{GF}(q) = \{\rho_0, \rho_1, \rho_2, \ldots, \rho_{q-1}\}$. The reliability values of each received symbol are arranged in a reliability matrix $\mathbf{\Pi}$ and converted into a multiplicity matrix $\mathbf{M}$. The interpolation and factorization processes then follow in the same way as described previously. The whole system model for soft-decision list decoding is illustrated in Figure 5.4.

5.3.1 Mapping Reliability Values into Multiplicity Values

Instead of making a hard decision on the received symbols, the soft values of each symbol are used to give a measure of reliability. Therefore, the received vector $\mathbf{R} = (r_0, r_1, \ldots, r_{n-1})$ now contains soft values and not finite field elements. The reliability of each received symbol is denoted as $\pi_{i,j}$, which gives the probability of the jth transmitted coded symbol c_j being the $j = i$th element ρ_i in $\mathrm{GF}(q)$, given the soft value of the jth received symbol r_j.

$$p_{i,j} = \mathrm{P}(c_j = \rho_i | r_j)(i = 0, 1, \ldots, q-1 \text{ and } j = 0, 1, \ldots, n-1). \tag{5.42}$$

These reliabilities are entered into a $q \times n$ reliability matrix $\mathbf{\Pi}$.

$$\mathbf{\Pi} = \begin{bmatrix} \pi_{0,0} & \pi_{0,1} & \cdots & \cdots & \cdots & \pi_{0,n-1} \\ \pi_{1,0} & \pi_{1,1} & & & & \pi_{1,n-1} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & \pi_{i,j} & & \vdots \\ \vdots & & & & \ddots & \vdots \\ \pi_{q-1,0} & \pi_{q-1,1} & \cdots & \cdots & \cdots & \pi_{q-1,n-1} \end{bmatrix}. \tag{5.43}$$

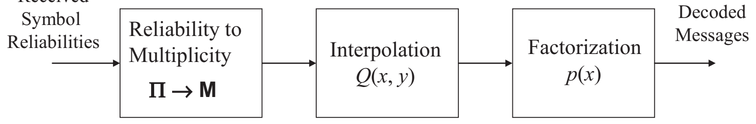


Figure 5.4 System model for soft-decision list decoding.

Referring to Figure 5.1, the matrix $\mathbf{\Pi}$ is taken as an input to the soft-decision decoder and converted to a multiplicity matrix $\mathbf{M}$, followed by the interpolation and factorization processes. An algorithm presented in [5] to convert the reliability matrix $\mathbf{\Pi}$ to a multiplicity matrix $\mathbf{M}$ is now given.

Algorithm 5.3: Convert reliability matrix $\mathbf{\Pi}$ to multiplicity matrix M [5] Input: Reliability matrix $\mathbf{\Pi}$ and a desired value of the sum of multiplicities in matrix $\boldsymbol{M}$ as: $s = \sum_{i=0}^{q-1} \sum_{j=0}^{n-1} m_{i,j}$.

Initialization: Set $\mathbf{\Pi}^* = \mathbf{\Pi}$ and $q \times n$ all-zero multiplicity matrix $\mathbf{M}$:

```
1. While (s > 0)
   {
2. Find the maximal entry $\pi^*_{i,j}$ in $\mathbf{\Pi}^*$ with position $(i, j)$.
3. Update $\pi^*_{i,j}$ in $\mathbf{\Pi}^*$ as $\pi^*_{i,j} = \frac{\pi_{i,j}}{m_{i,j}+2}$.
4. Update $m_{i,j}$ in $\mathbf{M}$ as $m_{i,j} = m_{i,j} + 1$.
5. $s = s - 1$.
   }
```

Algorithm 5.3 results in a $q \times n$ multiplicity matrix $\mathbf{M}$, which can be written as:

$$\mathbf{M} = \begin{bmatrix} m_{0,0} & m_{0,1} & \cdots & \cdots & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & & & & m_{1,n-1} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & m_{i,j} & & \vdots \\ \vdots & & & & \ddots & \vdots \\ m_{q-1,0} & m_{q-1,1} & \cdots & \cdots & \cdots & m_{q-1,n-1} \end{bmatrix}, \tag{5.44}$$

The entry $m_{i,j}$ represents the multiplicity value of interpolated point (x_j, ρ_i) ($j = 0, 1, \ldots, n-1$ and $i = 0, 1, \ldots, q-1$). x_j are the finite field elements used in the encoding process described in Chapter 3. In Algorithm 5.3, the desired value s indicates the total value of multiplicity of all interpolated points. This algorithm gives priority to those interpolated points which correspond to a higher reliability value $\pi_{i,j}$, to be assigned with a higher multiplicity value $m_{i,j}$. For an illustration of the algorithm, see Example 5.5.

Example 5.5: For soft-decision list decoding of the (7, 2) Reed–Solomon code defined in GF(8), the following 8×7 reliability matrix $\mathbf{\Pi}$ is obtained by the receiver:

$$\mathbf{\Pi} = \begin{bmatrix} 0.959796 & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.954880 & 0.000006 \\ 0.009543 & 0.736533 & \underline{0.968097} & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}.$$

(Note: in the matrix $\mathbf{\Pi}$ ($\mathbf{\Pi}^*$), the maximal entry is underlined).

Apply Algorithm 5.3 with a desired value $s = 20$.

Initialization: Set $\mathbf{\Pi}^* = \mathbf{\Pi}$ and $\mathbf{M} = 0$.

As $s = 20 > 0$, find the maximal entry $\pi^*_{i,j} = 0.968097$ in $\mathbf{\Pi}^*$ with position $(i, j) = (4, 2)$.

Update $\pi^*_{4,2}$ as $\pi^*_{4,2} = \dfrac{\pi_{4,2}}{m_{4,2} + 2} = \dfrac{0.968097}{0 + 2} = 0.484048.$

Update $m_{4,2}$ in $\mathbf{M}$ as $m_{4,2} = 0 + 1 = 1$

$$s = s - 1 = 19.$$

Now the updated $\mathbf{\Pi}^*$ is:

$$\mathbf{\Pi}^* = \begin{bmatrix} \underline{0.959796} & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.954880 & 0.000006 \\ 0.009543 & 0.736533 & 0.484048 & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}$$

and the updated $\mathbf{M}$ is:

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

In the next iteration, as $s = 19 > 0$, find the maximal entry $\pi^*_{i,j} = 0.959696$ in $\mathbf{\Pi}^*$ with position $(i, j) = (0, 0)$.

Update $\pi^*_{0,0}$ as $\pi^*_{0,0} = \dfrac{\pi_{0,0}}{m_{0,0}+2} = \dfrac{0.959796}{0+2} = 0.479898$.
Update $m_{0,0}$ in $\mathbf{M}$ as $m_{0,0} = 0 + 1 = 1$.

$$s = s - 1 = 18.$$

Now the updated $\mathbf{\Pi}^*$ is:

$$\mathbf{\Pi}^* = \begin{bmatrix} 0.479898 & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & \underline{0.954880} & 0.000006 \\ 0.009543 & 0.736533 & 0.484048 & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}$$

and the updated $\mathbf{M}$ is:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Following the same process until $s = 0$, the updated $\mathbf{\Pi}^*$ is:

$$\mathbf{\Pi}^* = \begin{bmatrix} 0.239949 & 0.214170 & 0.005453 & 0.230535 & 0.001125 & 0.000505 & 0.230576 \\ 0.001749 & 0.005760 & 0.000000 & 0.175013 & 0.224388 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.238720 & 0.000006 \\ 0.009543 & \underline{0.245511} & 0.242024 & 0.003180 & 0.000000 & 0.000000 & 0.139395 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix}$$

and the updated $\mathbf{M}$ is:

$$\mathbf{M} = \begin{bmatrix} 3 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 2 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 2 & 3 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

In the resulting multiplicity matrix **M**, it can be seen that the sum of its entries $\sum_{i=0}^{7}\sum_{j=0}^{6} m_{i,j} = 20$, which is the desired value s set at the beginning. From **M** we see that there are nine nonzero entries, implying that there are now nine points used to generate the interpolation polynomial $Q(x, y)$. These points are:

$$\begin{gathered}(x_0, r_0) \text{ with } m_{0,0} = 3,\ (x_1, \rho_4) \text{ with } m_{4,1} = 2,\ (x_2, r_4) \text{ with } m_{4,2} = 3,\\ (x_3, r_0) \text{ with } m_{0,3} = 1,\ (x_3, r_1) \text{ with } m_{1,3} = 2,\ (x_4, r_1) \text{ with } m_{1,4} = 3,\\ (x_5, r_3) \text{ with } m_{3,5} = 3,\ (x_6, r_0) \text{ with } m_{0,6} = 2,\ (x_6, r_4) \text{ with } m_{4,6} = 1.\end{gathered}$$

5.3.2 *Solution Analysis for Soft-Decision List Decoding*

Based on Section 5.2.2, to have a multiplicity of m_{ij} over interpolated point (x_j, ρ_i), the Hasse derivative evaluation of an interpolation polynomial is now defined as:

$$D_{uv}Q(x_j, \rho_i) = \sum_{a\geq u, b\geq v} \binom{a}{u}\binom{b}{v} Q_{ab}x_j^{a-u}\rho_i^{b-v},\ u + v < m_{i,j} \tag{5.45}$$

where r_i in (5.17) is replaced with ρ_i. The total number of iterations for all points is:

$$C_M = \frac{1}{2}\sum_{i=0}^{q-1}\sum_{j=0}^{n-1} m_{i,j}(m_{i,j} + 1). \tag{5.46}$$

C_M is called the 'cost' of multiplicity matrix **M**, which also denotes the number of iterations in the interpolation process.

Based on Lemma 5.1, if $f(x)$ is the message polynomial that satisfies $f(x_j) = c_j$ $(j = 0, 1, \ldots, n-1)$, polynomial $Q(x, f(x))$ should satisfy:

$$(x - x_0)^{m_0}(x - x_1)^{m_1} \cdots (x - x_{n-1})^{m_{n-1}} | Q(x, f(x)). \tag{5.47}$$

Again, if we let $g_1(x) = (x - x_0)^{m_0}(x - x_1)^{m_1} \cdots (x - x_{n-1})^{m_{n-1}}$ and $g_2(x) = Q(x, f(x))$, based on (5.12), $g_1(x)|g_2(x)$. $g_1(x)$ has x-degree $\deg_x(g_1(x)) = m_0 + m_1 + \cdots + m_{n-1}$. The x-degree of $g_1(x)$ is defined as the code word score S_M with respect to multiplicity matrix **M**:

$$\begin{aligned} S_M(\overline{c}) &= \deg_x(g_1(x)) = m_0 + m_1 + \cdots + m_{n-1}\\ &= \sum_{j=0}^{n-1}\{m_{i,j}|\rho_i = c_j, i = 0, 1, \ldots, q-1\}. \end{aligned} \tag{5.48}$$

The x-degree of $g_2(x)$ is bounded by $\deg_x(g_2(x)) \leq \deg_{1,k-1} Q(x, y)$. Therefore, if $S_M(\overline{c}) > \deg_{1,k-1} Q(x, y)$ then $\deg_x(g_1(x)) > \deg_x(g_2(x))$. To satisfy both $\deg_x(g_1(x)) > \deg_x(g_2(x))$ and $g_1(x)|g_2(x)$, the only solution is $g_2(x) = 0$, which indicates

$Q(x, f(x)) = 0$, or equivalently $y - f(x)|Q(x, y)$, and the message polynomial $f(x)$ can be found by determining $Q(x, y)$'s y-roots. As a result, if the code word score with respect to multiplicity matrix **M** is greater than the interpolated polynomial $Q(x, y)$'s $(1, k - 1)$-weighted degree

$$S_M(\overline{c}) > \deg_{1,k-1} Q(x, y) \tag{5.49}$$

then $Q(x, f(x)) = 0$, or equivalently $y - f(x)|Q(x, y)$. Message polynomial $f(x)$ can be found by determining the y roots of $Q(x, y)$.

Based on the $(1, k - 1)$-weighted degree definition of monomial $x^a y^b$ given in Section 5.2.1, let us define the following two parameters:

$$N_{1,k-1}(\delta) = \left|\{x^a y^b : a, b \geq 0 \text{ and } \deg_{1,k-1}(x^a y^b) \leq \delta, \delta \in \mathbb{N}\}\right|, \tag{5.50}$$

which represents the number of bivariate monomial $x^a y^b$ with $(1, k - 1)$-weighted degree not greater than a nonnegative integer δ [5]; and:

$$\Delta_{1,k-1}(v) = \min\{\delta : N_{1,k-1}(\delta) > v, v \in \mathbb{N}\}, \tag{5.51}$$

which denotes the minimal value of δ that guarantees $N_{1,k-1}(\delta)$ is greater than a nonnegative integer v [5].

If the $(1, k - 1)$-weighted degree of interpolated polynomial Q is δ^*, based on (5.50), Q has at most $N_{1,k-1}(\delta^*)$ nonzero coefficients. The interpolation procedure generates a system of C_M linear equations of type (5.45). The system will be solvable if [5]:

$$N_{1,k-1}(\delta^*) > C_M. \tag{5.52}$$

Based on (5.51), in order to guarantee the solution, the $(1, k - 1)$-weighted degree δ^* of the interpolated polynomial Q should be large enough that:

$$\deg_{1,k-1}(Q(x, y)) = \delta^* = \Delta_{1,k-1}(C_M). \tag{5.53}$$

Therefore, according to (5.49), given the soft-decision code word score (5.48) and the $(1, k - 1)$-weighted degree of the interpolated polynomial Q (5.53), the message polynomial f can be found if:

$$S_M(\overline{c}) > \Delta_{1,k-1}(C_M). \tag{5.54}$$

As the $(1, k - 1)$-weighted degree of the interpolated polynomial $Q(x, y)$ can be determined by (5.53), while $\Delta_{1,k-1}(C_M)$ can be realized by $\Delta_{1,k-1}(C_M) = \deg_{1,k-1}(x^a y^b|\text{ord}(x^a y^b) = C_M)$, a stopping rule for Algorithm 5.3 based on the designed length of output list l can be imposed. This is more realistic for assessing the performance soft-decision list decoding. As the factorization outputs are the y-roots of the interpolated polynomial Q, the maximal number of outputs l_M based on the interpolated polynomial Q is:

$$l_M = \deg_{0,1} Q(x, y) = \left\lfloor \frac{\deg_{1,k-1} Q(x, y)}{k-1} \right\rfloor = \left\lfloor \frac{\Delta_{1,k-1}(C_M)}{k-1} \right\rfloor. \tag{5.55}$$

Therefore, after step 5 of Algorithm 5.3, the updated cost C_M of the multiplicity matrix **M** can be determined using (5.46). As C_M has been determined, the interpolated polynomial $Q(x, y)$'s $(1, k-1)$-weighted degree can be determined by (5.53). (5.55) can then be applied to calculate the maximal number of factorization outputs, l_M. Based on a designed length of output list l, Algorithm 5.3 is stopped once l_M is greater than l.

In practice, due to the decoding complexity restriction, soft-decision list decoding can only be performed based on a designed length of output list l. This output length restriction in fact leads to practical decoding performance degradation. This phenomenon will be seen later when the simulation results are discussed.

As mentioned in Section 5.2.5, to build interpolated polynomial $Q(x, y)$ there are in total C_M (5.46) iterations. Therefore, the iteration index i_k used in Algorithm 5.1 is: $i_k = 0, 1, \ldots, C_M$. Based on a designed length of output list l, the initialization at step 1 of Algorithm 5.1 can be modified as:

$$G_0 = \{Q_{0,j} = y^j, j = 0, 1, \ldots, l\}, \tag{5.56}$$

where l is the designed length of the output list. As there are in total C_M iterations, based on the complexity reducing scheme's description given in Section 5.2.4, the interpolated polynomial Q's leading order is less than or equal to the total number of iterations C_M:

$$\text{lod}(Q(x, y)) \leq C_M. \tag{5.57}$$

This indicates the fact that (5.53) is an upper bound for the interpolated polynomial's $(1, k-1)$-weighted degree:

$$\deg_{1,k-1} Q(x, y) \leq (C_M). \tag{5.58}$$

Based on (5.56), those polynomials with leading order greater than C_M will neither be chosen as the interpolated polynomial nor be modified with the interpolated polynomial. Therefore they can be eliminated from the polynomial group and the modification at step 2 can be rewritten as:

$$G_{i_k} = \{Q_{i_k,j} | \text{lod}(Q_{i_k,j}) = C_M\}. \tag{5.59}$$

With respect to interpolated point (x_j, ρ_i) and Hasse derivative parameter (u, v), where $u + v < m_{i,j}$, the Hasse derivative evaluation performed at step 3 of Algorithm 5.1 can be modified and determined by (5.45). The resulting process is the same as Algorithm 5.1, with the exception that for polynomial modification in (5.59) the interpolated point's x-coordinate x_i should be replaced by the x_j which is the current interpolated point's x-coordinate. Also, the index j should be used for the interpolated point's x-coordinate x_j and polynomials in the group $Q_{i_k,j}$.

5.3.3 *Simulation Results*

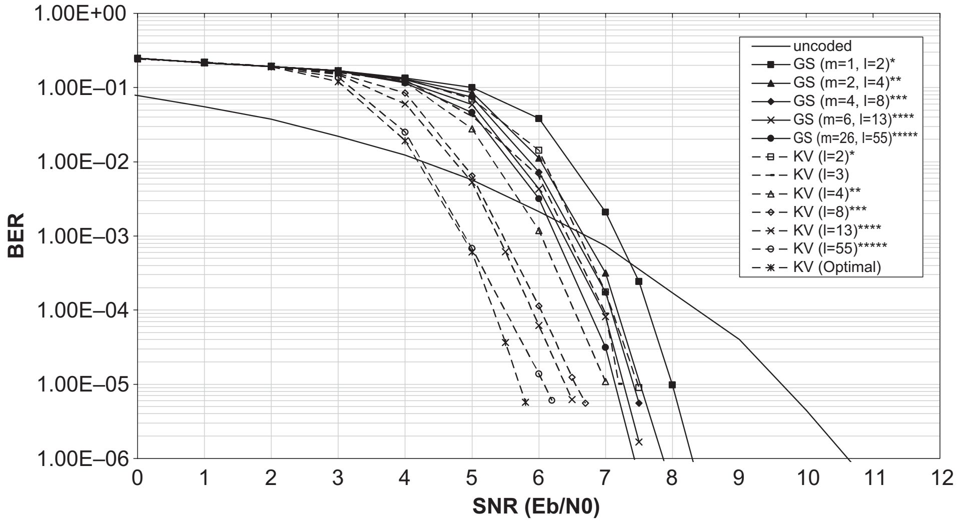
This section presents both hard-decision and soft-decision list decoding results for two Reed–Solomon codes: (63, 15) and (63, 31). They are simulated on both the AWGN and Rayleigh fading channels. The Rayleigh fading channel is frequency nonselective with Doppler frequency [16] 126.67 Hz and data rate 30 kb/s. The fading profile is generated using Jakes' method [16]. The fading coefficients have mean value 1.55 and variance 0.60. On the Rayleigh fading channel, a block interleaver of size 63×63 is used to combat the fading effect. QPSK modulation is used and simulations are run using the C programming language.

Comparisons between hard-decision and soft-decision are made based on output length l. For an output length l, there are $l + 1$ polynomials taking part in the iterative interpolation process. The total number of iterations (C_m in (5.11) for hard-decision and C_M in (5.46) for soft-decision) also grow with length l. The number of polynomials $l + 1$ and the number of iterations (C_m, C_M) are the important parameters that determine the decoding complexity. Based on the same designed length, from Figures 5.5 and 5.6 it can be seen that soft-decision can achieve significant coding gains over hard-decision list decoding, especially on the Rayleigh fading channel. For example, with a designed length $l = 2$, soft-decision list decoding of the (63, 15) Reed–Solomon code can achieve about a 5.8 dB coding gain at BER $= 10^{-5}$ over hard-decision decoding.

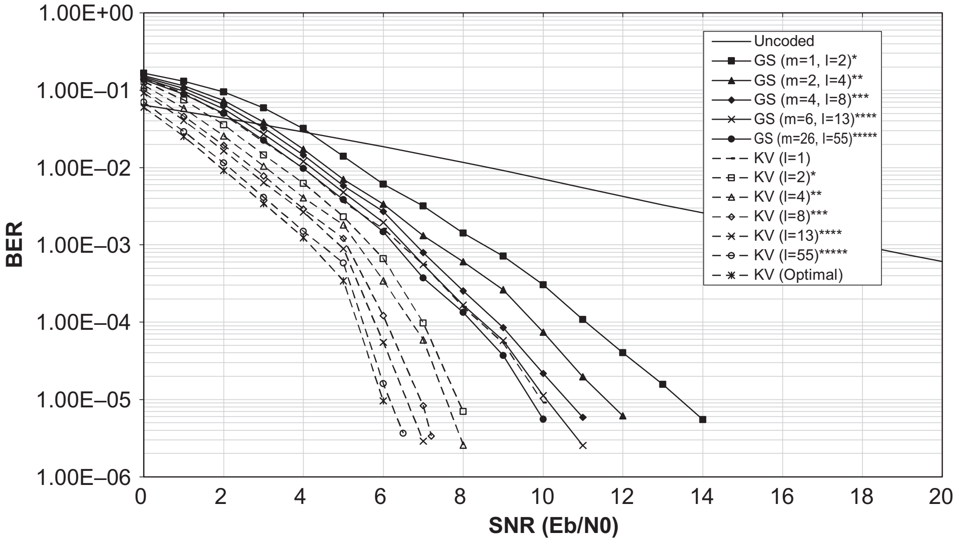
According to the analysis in [14], the performance improvement of soft-decision list decoding over hard-decision list decoding is achieved with an insignificant complexity penalty. This is because for the list decoding algorithm, the complexity is mainly dominated by the interpolation process, and the complexity introduced by the a priori process (Algorithm 5.3) is marginal. For the interpolation process, the important parameter that determines its complexity is the iteration number. As the iteration number of soft-decision does not vary much from hard-decision based on the same designed output length, the complexity of soft-decision list decoding is not much higher than that of hard-decision list decoding.

5.4 List Decoding of Algebraic–Geometric Codes

The GS algorithm consists of two processes: interpolation and factorization. Given a received word $R = (r_0, r_1, \ldots, r_{n-1})$ ($r_i \in \mathrm{GF}(q)$, $i = 0, 1, \ldots, n - 1$), n interpolated



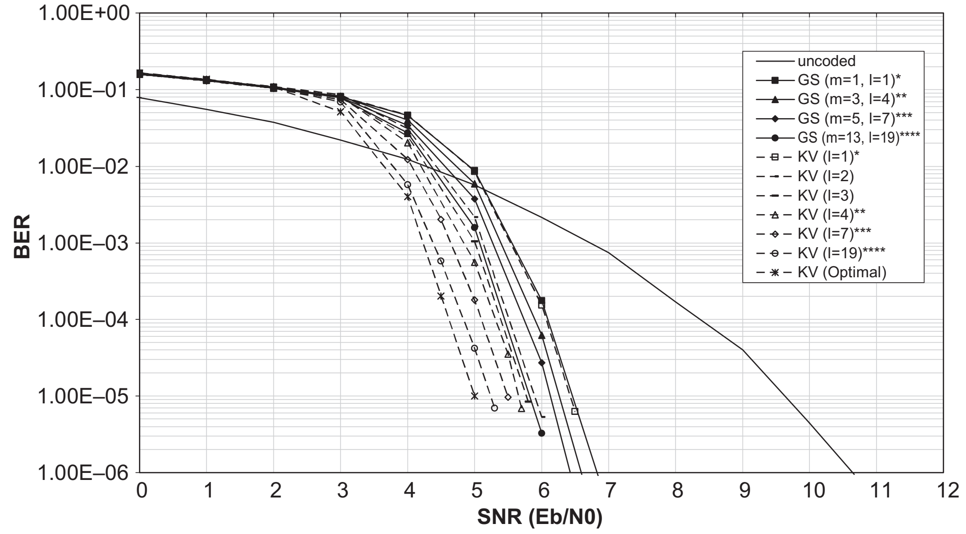
(a) over AWGN channel



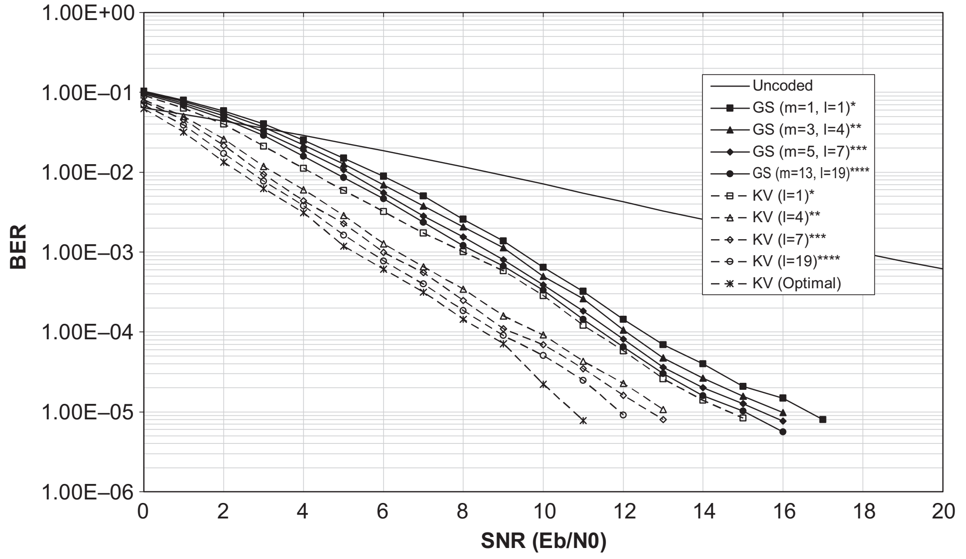
(b) over Rayleigh fading channel

Figure 5.5 Hard-decision and soft-decision list decoding Reed–Solomon code (63, 15).

units can be formed by combining each received symbol with its respective affine point used in encoding, as: $(p_0, r_0), (p_1, r_1), \ldots, (p_{n-1}, r_{n-1})$. Interpolation builds the minimal polynomial $Q \in F_q[x, y, z]$, which has a zero of multiplicity of at least m over the n interpolated units. Q can be written as: $Q = \sum_{a,b} Q_{ab}\phi_a z^b$, where $Q_{ab} \in \mathrm{GF}(q)$ and ϕ_a is a set of rational functions with pole orders up to a [17–19]. If (p_i, r_i)



(a) over AWGN channel



(b) over Rayleigh fading channel

Figure 5.6 Hard-decision and soft-decision list decoding Reed–Solomon code (63, 31).

is the intended interpolated unit, it can also be written with respect to the zero basis functions in Z_{w,p_i} defined in Chapter 2 as [9]:

$$Q = \sum_{u,v} Q_{uv}^{(p_i,r_i)} \psi_{p_i,u}(z - r_i)^v, \tag{5.60}$$

where $Q_{uv}^{(p_i,r_i)} \in \mathrm{GF}(q)$. If $Q_{uv}^{(p_i,r_i)} = 0$ for $u + v < m$, polynomial Q has a zero of multiplicity at least m at unit (p_i, r_i) [9, 11]. As $z^b = (z - r_i + r_i)^b = \sum_{v \le b} \binom{b}{v} r_i^{b-v}(z - r_i)^v$ and $\phi_a = \sum_u \gamma_{a,p_i,u}\psi_{p_i,u}$, substitute them into (5.15):

$$\begin{aligned} Q &= \sum_{a,b} Q_{ab} \left(\sum_u \gamma_{a,p_i,u}\psi_{p_i,u} \right) \left(\sum_{v \le b} \binom{b}{v} r_i^{b-v}(z - r_i)^v \right) \\ &= \sum_{u,v} \left(\sum_{a,b \ge v} Q_{ab} \binom{b}{v} \gamma_{a,p_i,u} r_i^{b-v} \right) \psi_{p_i,u}(z - r_i)^v \end{aligned} . \tag{5.61}$$

Therefore, the coefficients $Q_{uv}^{(p_i,r_i)}$ of (5.61) can be written as:

$$Q_{uv}^{(p_i,r_i)} = \sum_{a,b \ge v} Q_{ab} \binom{b}{v} \gamma_{a,p_i,u} r_i^{b-v}. \tag{5.62}$$

(5.62) defines the zero condition constraints on the coefficients Q_{ab} of polynomial Q analogous to the Hasse derivative, so that Q has a zero of multiplicity at least m over unit (p_i, r_i). Example 5.6 shows how to define the zero condition of a polynomial in $F_q[x, y, z]$ using (5.62).

Example 5.6: Given the polynomial $Q(x, y, z) = 1 + \alpha y + \alpha x^2 + z^2(1 + \alpha^2 y)$ defined in $\mathrm{GF}(4)[x, y, z]$ justify the fact that it has a zero of multiplicity at least 2 over the unit $(p, r) = ((1, \alpha), \alpha)$.

Polynomial $Q(x, y, z) = 1 + \alpha y + \alpha x^2 + z^2(1 + \alpha^2 y) = Q_{00}\phi_0 z^0 + Q_{20}\phi_2 z^0 + Q_{30}\phi_3 z^0 + Q_{02}\phi_0 z^2 + Q_{22}\phi_2 z^2$. Supporting the zero condition calculations, the corresponding coefficients $\gamma_{a,p,u}$ are shown in Table 5.3.

Table 5.3 Corresponding coefficients $\gamma_{a,p,u}$ given $p = (1, \alpha)$.

a \ u	0	1	2	3	...
0	1	1	α	1	...
1	0	1	1	0	...
⋮	⋮	⋮	⋮	⋮	

Based on the above description, to justify that Q has a zero of multiplicity m over unit (p, r), its coefficients Q_{ab} should satisfy $Q_{uv}^{(p,r)} = 0$ for $u + v < 2$ as: $Q_{00}^{(p,r)} = 0$, $Q_{01}^{(p,r)} = 0$ and $Q_{10}^{(p,r)} = 0$.

Based on definition (5.62):

$$\begin{aligned}
Q_{00}^{(p,r)} &= Q_{00}\binom{0}{0}\gamma_{0,p,0}\alpha^{0-0} + Q_{20}\binom{0}{0}\gamma_{2,p,0}\alpha^{0-0} + Q_{30}\binom{0}{0}\gamma_{3,p,0}\alpha^{0-0} \\
&\quad + Q_{02}\binom{2}{0}\gamma_{0,p,0}\alpha^{2-0} + Q_{22}\binom{2}{0}\gamma_{2,p,0}\alpha^{2-0} \\
&= 1 + \alpha^2 + \alpha + \alpha^2 + \alpha^2 = 0 \\
Q_{01}^{(p,r)} &= Q_{02}\binom{2}{1}\gamma_{0,p,0}\alpha^{2-1} + Q_{22}\binom{2}{1}\gamma_{2,p,0}\alpha^{2-1} = 0 + 0 = 0 \\
Q_{10}^{(p,r)} &= Q_{00}\binom{0}{0}\gamma_{0,p,1}\alpha^{0-0} + Q_{20}\binom{0}{0}\gamma_{2,p,1}\alpha^{0-0} \\
&\quad + Q_{30}\binom{0}{0}\gamma_{3,p,1}\alpha^{0-0} + Q_{02}\binom{2}{0}\gamma_{0,p,1}\alpha^{2-0} + Q_{22}\binom{2}{0}\gamma_{2,p,1}\alpha^{2-0} \\
&= \alpha + \alpha = 0.
\end{aligned}$$

Therefore, polynomial Q has a zero of multiplicity at least 2 over unit $(p, r) = ((1, \alpha), \alpha)$.

If constraint (5.62) for the coefficients of polynomial Q is denoted as $D_{uv}^{(p_i,r_i)}(Q)$, such that:

$$D_{uv}^{(p_i,r_i)}(Q) = Q_{uv}^{(p_i,r_i)} = \sum_{a,b\geq v} Q_{ab}\binom{b}{v}\gamma_{a,p_i,u}r_i^{b-v} \tag{5.63}$$

then interpolation builds a polynomial Q defined as:

$$\begin{aligned}
Q = \min_{\mathrm{lod}(Q)} \big\{ Q \in F_q[x, y, z] | D_{uv}^{(p_i,r_i)}(Q) = 0 \quad \text{for} \quad i = 0, 1, \ldots, n-1 \wedge u \\
+ v < m \ (u, v \in \mathbb{N})\} \big\}.
\end{aligned} \tag{5.64}$$

As there are $\binom{m+1}{2}$ permutations of (u, v) for $u + v < m$, there are in total:

$$C = n\binom{m+1}{2} \tag{5.65}$$

zero condition constraints that the coefficients Q_{ab} of polynomial Q need to satisfy. C also represents the number of iterations in the interpolation algorithm [9, 11], in which each iteration imposes a zero condition constraint to Q_{ab}.

Definition: For monomial $\phi_a z^b$, where $\phi_a \in L_w$ and L_w is the Hermitian curve's pole basis defined in GF(w^2), its (1, w_z)-weighted degree is defined as:

$$\deg_{1,w_z}(\phi_a z^b) = v_{p_\infty}(\phi_a^{-1}) + b \cdot w_z,$$

where w_z is the weighted degree for variable z, and defined as: $w_z = v_{p_\infty}(z^{-1}) = v_{p_\infty}(\phi_{k-1}^{-1})$. ϕ_{k-1} is the maximal term in the message polynomial. A (1, w_z)-lexicographic order (ord) can be defined to arrange monomials $\phi_a z^b$:

$$\phi_{a_1} z^{b_1} < \phi_{a_2} z^{b_2},$$

if $\deg_{1,w_z}(\phi_{a_1} z^{b_1}) < \deg_{1,w_z}(\phi_{a_2} z^{b_2})$, or $\deg_{1,w_z}(\phi_{a_1} z^{b_1}) = \deg_{1,w_z}(\phi_{a_2} z^{b_2})$ and $b_1 < b_2$ [9]. If $\phi_{a'} z^{b'}$ is the maximal monomial in polynomial $Q = \sum_{a,b} Q_{ab} \phi_a z^b$ as:

$$\phi_{a'} z^{b'} = \max\{\phi_a z^b | Q_{ab} \neq 0\}.$$

$\phi_{a'} z^{b'}$ is called Q's leading monomial (LM) and its coefficient $Q_{a'b'}$ is called f's leading coefficient (LC), denoted as: $\text{LM}(f) = \phi_{a'} z^{b'}$ and $\text{LC}(f) = f_{a'b'}$. Polynomial Q's (1, w_z)-weighted degree ($\deg_{1,w_z}(Q)$) and leading order ($\text{lod}(Q)$) are defined as:

$$\deg_{1,w_z}(Q) = \deg_{1,w_z}(\phi_{a'} z^{b'}), \quad \text{and} \quad \text{lod}(Q) = \text{ord}(\phi_{a'} z^{b'}).$$

The (1, w_z)-weighted degree upper bound of polynomial Q is defined as [9, 11]:

$$\max\{\deg_{1,w_z} Q\} = l_m v_{p_\infty}(z-1) + t_m, \tag{5.66}$$

where l_m is the maximal number of output candidates from factorization, defined as:

$$l_m = \max\left\{u | \binom{u}{2} v_{p_\infty}(z-1) - (u-1)g \leq C\right\} - 1, \tag{5.67}$$

and parameter t_m is defined as:

$$t_m = \max\left\{u | (l_m+1)u - \Gamma(u) + \binom{l_m+1}{2} v_{p_\infty}(z-1) - l_m g \leq C\right\}, \tag{5.68}$$

where g is the genus of the Hermitian curve, $u \in \mathbb{N}$ and $\Gamma(u)$ denotes the number of gaps that are less than or equal to the nonnegative integer u [11].

If there exists a polynomial $h \in F_q^{u_z}[x, y]$ such that:

$$\Lambda(h, R) = |\{i | h(p_i) = r_i, i = 0, 1, \ldots, n-1\}| \tag{5.69}$$

then the total zero orders of polynomial $Q(x, y, h)$ over all the interpolated units is:

$$\sum_{i=0}^{n-1} v_{p_i}(Q(x, y, h)) = m\Lambda(h, R). \tag{5.70}$$

To define the total zero order of polynomial $Q(x, y, h)$, the following lemma is applied:

Lemma 5.4 *If $Q(x, y, z)$ has a zero of multiplicity m over unit (p_i, r_i) and h is a polynomial in $F_q^{u_z}[x, y]$ that satisfies $h(p_i) = r_i$ then $Q(x, y, h)$ has a zero order of at least m at p_i, as $v_{p_i}(Q(x, y, h)) \geq m$ [9, 11].*

(5.69) defines the total number of affine points that satisfy $h(p_i) = r_i$, and therefore the total zero order of polynomial $Q(x, y, h)$ over all the affine points is defined by (5.70).

Theorem 5.1 *If polynomial $Q(x, y, h)$'s total zero order is greater than its pole order, as:*

$$\sum_{i=0}^{n-1} v_{p_i}(Q(x, y, h)) > v_{p_\infty}(Q(x, y, h)^{-1}), \tag{5.71}$$

then h is the z root of Q: $Q(x, y, h) = 0$, or equivalently $z - h|Q(x, y, z)$ [6, 9, 11].

As $h \in F_q^{u_z}[x, y]$, $v_{p_\infty}(Q(x, y, h)^{-1}) = v_{p_\infty}(Q(x, y, z)^{-1}) = \deg_{1,w_z}(Q(x, y, z))$. Therefore, based on (5.69) and (5.70), Theorem 5.1 results in the following corollary:

Corollary 5.1 *If there exists a polynomial $h \in F_q^{u_z}[x, y]$ such that:*

$$m\,\Lambda(h, R) > \deg_{1,w_z}(Q(x, y, z)) \tag{5.72}$$

then the list decoding outputs h can be found by factorizing the interpolated polynomial $Q(x, y, z)$ as: $z - h|Q(x, y, z)$ [18].

If $h = f$, (5.69) defines the number of uncorrupted received symbols. Therefore, the GS algorithm's error-correction capability τ_m is:

$$t_m = n - \Lambda(h, R) = n - \left\lfloor \frac{\deg_{1,w_z} Q}{m} \right\rfloor - 1. \tag{5.73}$$

Since the upper bound of $\deg_{1,w_z} Q$ is defined by (5.70):

$$t_m \geq n - \left\lfloor \frac{l_m v_{p_\infty}(z^{-1}) + t_m}{m} \right\rfloor - 1. \tag{5.74}$$

The GS algorithm's error-correction capability upper bound for a (n, k) Hermitian code is defined by:

$$t_{\mathrm{GS}} = n - \left\lfloor \sqrt{n(n - d^*)} \right\rfloor - 1.$$

5.5 Determining the Corresponding Coefficients

Based on (5.62), the corresponding coefficients $\gamma_{a,p_i,u}$ are critical for defining the zero condition of a polynomial in $F_q[x, y, z]$. Without knowing them, we have to transfer a general polynomial written with respect to the zero basis functions and find the coefficients $Q_{uv}^{(p_i,r_i)}$, which is not efficient during the iterative interpolation. In fact, the corresponding coefficients $\gamma_{a,p_i,u}$ can be determined independently of the received word. Therefore, if they can be determined beforehand and applied during the iterations, the interpolation efficiency can be greatly improved. This section proposes an algorithm to determine them.

The problem we intend to solve can be simply stated as: given an affine point $p_i = (x_i, y_i)$ of curve H_w and a pole basis monomial ϕ_a, determine the corresponding coefficients $\gamma_{a,p_i,u}$ so that ϕ_a can be written as a sum of the zero basis functions $\psi_{p_i,u} : \phi_a = \sum_u \gamma_{a,p_i,u}\psi_{p_i,u}$. For any two pole basis monomials ϕ_{a_1} and ϕ_{a_2} in L_w, $\phi_{a_1}\phi_{a_2} = \sum_{a\in\mathrm{N}} \phi_a$ and the zero basis function $\psi_{p_i,u}$ (2.13) can be written as a sum of pole basis monomials ϕ_a [9]:

$$\psi_{p_i,u} = \sum_a \zeta_a\phi_a, \tag{5.75}$$

where coefficients $\zeta_a \in \mathrm{GF}(q)$. Partition $\psi_{p_i,u}(x, y)$ as:

$$\psi_{p_i,u} = \psi^A_{p_i,u} \cdot \psi^B_{p_i,u}, \tag{5.76}$$

where $\psi^A_{p_i,u} = (x - x_i)^\lambda$ and $\psi^B_{p_i,u} = [(y - y_i) - x_i^w(x - x_i)]^\delta = [y - x_i^w x - (y_i - x_i^{w+1})]^\delta$. It is easy to recognize that $\psi^A_{p_i,u}$ has leading monomial $\mathrm{LM}(\psi^A_{p_i,u}) = x^\delta$ and leading coefficient $\mathrm{LC}(\psi^A_{p_i,u}) = 1$. As $v_{p_\infty}(y^{-1}) > v_{p_\infty}(x^{-1})$, $\psi^B_{p_i,u}$ has leading monomial $\mathrm{LM}(\psi^B_{p_i,u}) = y^\delta$ and leading coefficient $\mathrm{LC}(\psi^B_{p_i,u}) = 1$. Based on (5.76), $\psi_{p_i,u}$ has leading monomial $\mathrm{LM}(\psi^A_{p_i,u}) \cdot \mathrm{LM}(\psi^B_{p_i,u}) = x^\lambda y^\delta$ and leading coefficient $\mathrm{LC}(\psi^A_{p_i,u}) \cdot \mathrm{LC}(\psi^B_{p_i,u}) = 1$. As $0 \le \lambda \le w$ and $\delta \ge 0$, the set of leading monomials of zero basis functions in Z_{w,p_i} contains all the monomials defined in pole basis L_w. Summarizing the above analysis, Corollary 5.2 is proposed as follows:

Corollary 5.2 *If ϕ_L is the leading monomial of zero basis function $\psi_{p_i,u}$ as* $\mathrm{LM}(\psi_{p_i,u}) = \phi_L$, *the leading coefficient of $\psi_{p_i,u}$ equals 1 and (5.75) can be written as [18]:*

$$\psi_{p_i,u} = \sum_{a<L} \zeta_a\phi_a + \phi_L. \tag{5.77}$$

The set of leading monomials of zero basis functions in Z_{w,p_i} contains all the monomials in L_w:

$$\{\mathrm{LM}(\psi_{p_i,u}) = \phi_L, \psi_{p_i,u} \in Z_{w,p_i}\} \subseteq L_w. \tag{5.78}$$

Following on, by identifying the second-largest pole basis monomial ϕ_{L-1} with coefficient $\zeta_{L-1} \in \mathrm{GF}(q)$ in $\psi_{p_i,u}$, (5.77) can also be written as [18]:

$$\psi_{p_i,u} = \sum_{a<L-1} \zeta_a \phi_a + \zeta_{L-1}\phi_{L-1} + \phi_L. \tag{5.79}$$

Now it is sufficient to propose the new efficient algorithm in order to determine the corresponding coefficients $\gamma_{a,p_i,u}$.

Algorithm 5.4: Determine the corresponding coefficients $\gamma_{a,p_i,u}$ between a pole basis monomial and zero basis functions [14, 18]

```
1. Initialize all corresponding coefficients $\gamma_{a,p_i,u} = 0$.
2. Find the zero basis function $\psi_{p_i,u}$ with $\mathrm{LM}(\psi_{p_i,u}) = \phi_a$, and let $\gamma_{a,p_i,u} = 1$.
3. Initialize function $\hat{\psi} = \psi_{p_i,u}$.
4. While ($\hat{\psi} \neq \phi_a$)
   {
5. Find the second-largest pole basis monomial $\phi_{L-1}$ with coefficient $\zeta_{L-1}$ in $\hat{\psi}$.
6. In $Z_{w,p_i}$, find a zero basis function $\psi_{p_i,\alpha}$ whose leading monomial $\mathrm{LM}(\psi_{p_i,u}) = \phi_{L-1}$, and let the corresponding coefficient $\gamma_{a,p_i,u} = \zeta_{L-1}$.
7. Update $\hat{\psi} = \hat{\psi} + \gamma_{a,p_i,u}\psi_{p_i,u}$.
   }
```

Proof: Notice that functions $\psi_{p_i,\alpha}$ with $\mathrm{LM}(\psi_{p_i,u}) > \phi_a$ will not contribute to the sum calculation of (5.62) and their corresponding coefficients $\gamma_{a,p_i,u} = 0$. The zero basis function $\psi_{p_i,u}$ found at step 2 has leading monomial $\phi_L = \phi_a$. Based on (5.79), it can be written as [18]:

$$\psi_{p_i,u} = \sum_{a'<L-1} \zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1} + \phi_a \tag{5.80}$$

(5.80) indicates that the corresponding coefficient between ϕ_a and $\psi_{p_i,u}$ is 1; $\gamma_{a,p_i,u} = 1$. Polynomial $\hat{\psi}$, initialized by step 3, is an accumulated polynomial resulting in ϕ_a. While $\hat{\psi} \neq \phi_a$, in (5.80), the second-largest monomial ϕ_{L-1} with coefficient ζ_{L-1} is identified by step 5. Next find another zero basis function $\psi_{p_i,u}$ in Z_{w,p_i} such that $\mathrm{LM}(\psi_{p_i,u}) = \phi_{L-1}$. According to Corollary 5.2, this zero basis function always exists and it can be written as: $\psi_{p_i,u} = \sum_{a'<L-1} \zeta_{a'}\phi_{a'} + \phi_{L-1}$. At step 6, the corresponding coefficient between monomial ϕ_a and the found zero basis function $\psi_{p_i,u}$ can be determined as: $\gamma_{a,p_i,u} = \zeta_{L-1}$. As a result, the accumulated calculation of step 7 can be written as:

$$\begin{aligned}\hat{\psi} &= \sum_{a'<L-1} \zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1} + \phi_a + \gamma_{a,p_i,u}\psi_{p_i,u} \\ &= \sum_{a'<L-1} \zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1} + \phi_a + \sum_{a'<L-1} \zeta_{L-1}\zeta_{a'}\phi_{a'} + \zeta_{L-1}\phi_{L-1}.\end{aligned} \tag{5.81}$$

Therefore, in the new accumulated $\hat{\psi}$, $\zeta_{L-1}\phi_{L-1}$ is eliminated, while the leading monomial ϕ_a is preserved. If the updated $\hat{\psi} \neq \phi_a$, its second-largest monomial ϕ_{L-1} is again eliminated, while ϕ_a is always preserved as a leading monomial by the same process. The algorithm terminates after all monomials that are smaller than ϕ_a have been eliminated and results in $\hat{\psi} = \phi_a$. This process is equivalent to the sum calculation of (5.62). Example 5.7 illustrates Algorithm 5.4.

Example 5.7: Given $p_i = (\alpha^2, \alpha^2)$ is an affine point on curve H_2 and a pole basis (L_2) monomial $\phi_5 = y^2$, determine the corresponding coefficients $\gamma_{5,p_i,u}$ so that ϕ_5 can be written as $\phi_5 = \sum\limits_u \gamma_{5,p_i,u}\psi_{p_i,u}$.

Based on (2.13), the first eight zero basis functions in Z_{2,p_i} can be listed as:

$$\begin{aligned}
&\psi_{p_i,0} = (x-\alpha^2)^0 = 1 \quad \psi_{p_i,1} = (x-\alpha^2)^1 = \alpha^2 + x\\
&\psi_{p_i,2} = (x-\alpha^2)^2 = \alpha + x^2 \quad \psi_{p_i,3} = (y-\alpha^2) - \alpha(x-\alpha^2) = \alpha + \alpha x + y\\
&\psi_{p_i,4} = (x-\alpha^2)[(y-\alpha^2) - \alpha(x-\alpha^2)] = 1 + \alpha^2 x + \alpha^2 y + \alpha x^2 + xy\\
&\psi_{p_i,5} = (x-\alpha^2)^2[(y-\alpha^2) - \alpha(x-\alpha^2)] = \alpha^2 + \alpha^2 x + \alpha x^2 + \alpha y^2 + x^2 y\\
&\psi_{p_i,6} = [(y-\alpha^2) - \alpha(x-\alpha^2)]^2 = \alpha^2 + \alpha^2 x^2 + y^2\\
&\psi_{p_i,7} = (x-\alpha^2)[(y-\alpha^2) - \alpha(x-\alpha^2)]^2 = \alpha + \alpha^2 x + \alpha^2 y + \alpha x^2 + xy^2.
\end{aligned}$$

Initialize all $\gamma_{5,p_i,u} = 0$. In Z_{2,p_i}, as $\mathrm{LM}(\psi_{p_i,6}) = \phi_5$, we let $\gamma_{5,p_i,6} = 1$ and initialize the accumulated polynomial $\hat{\psi} = \psi_{p_i,6} = \alpha^2 + \alpha^2 x^2 + y^2$.

As $\hat{\psi} \neq \phi_5$, its second-largest monomial $\phi_{L-1} = x^2$ with coefficient $\zeta_{L-1} = \alpha^2$ is identified. Among the zero basis functions in Z_{2,p_i}, we find $\psi_{p_i,2}$ with $\mathrm{LM}(\psi_{p_i,2}) = \phi_{L-1} = x^2$, and let $\gamma_{5,p_i,2} = \zeta_{L-1} = \alpha^2$. Update $\hat{\psi} = \hat{\psi} + \gamma_{5,p_i,2}\psi_{p_i,2} = \alpha + y^2$.

As $\hat{\psi} \neq \phi_5$, again its second-largest monomial $\phi_{L-1} = 1$ with coefficient $\zeta_{L-1} = \alpha$ is identified. Among the zero basis functions in Z_{2,p_i}, we find $\psi_{p_i,0}$ with $\mathrm{LM}(\psi_{p_i,0}) = \phi_{L-1} = 1$, and let $\gamma_{5,p_i,0} = \zeta_{L-1} = \alpha$. Update $\hat{\psi} = \hat{\psi} + \gamma_{5,p_i,0}\psi_{p_i,0} = y^2$.

Now, $\hat{\psi} = \phi_5$, we can stop the algorithm and output $\gamma_{5,p_i,0} = \alpha$, $\gamma_{5,p_i,2} = \alpha^2$ and $\gamma_{5,p_i,6} = 1$. The rest of the corresponding coefficients $\gamma_{5,p_i,u} = 0$ $(u \neq 0, 2, 6)$.

Before interpolation, monomials ϕ_a that exist in the interpolated polynomial Q are unknown. However, the $(1, w_z)$-weighted degree upper bound of polynomial Q is defined by (5.70), from which the largest pole basis monomial $\phi_{\max}$ that might exist in Q can be predicted by $v_{p_\infty}(\phi_{\max}^{-1}) = \max\{\deg_{1,w_z} Q\}$. Based on interpolation multiplicity m, with parameter $u < m$, the corresponding coefficients that might be used in interpolation are $\gamma_{0,p_i,u} \sim \gamma_{\max,p_i,u}(u < m)$. Therefore Algorithm 5.4 can be used to determine all the corresponding coefficients $\gamma_{0,p_i,u} \sim \gamma_{\max,p_i,u}$ and only $\gamma_{0,p_i,u} \sim \gamma_{\max,p_i,u}$ $(u < m)$ are stored for interpolation in order to minimize the memory requirement. For example, to list decode the (8, 4, 4) Hermitian code with multiplicity $m = 2$, $\max\{\deg_{1,w_z} Q\} = 13$. Therefore, the largest pole basis monomial that might exist in Q is $\phi_{\max} = \phi_{12} = x^2y^3$ and Algorithm 5.4 can be applied to calculate all the corresponding coefficients $\gamma_{0,p_i,u} \sim \gamma_{12,p_i,u}$ ($u < 2$) that are stored.

5.6 Complexity Reduction Interpolation

Interpolation determines a polynomial Q that intersects the points (p_i, r_i) ($i = 0, 1, \ldots, n-1$). This can be implemented by an iterative polynomial construction algorithm [4, 9, 11, 13]. At the beginning, a group of polynomials are initialized. During the iterations, they are tested by different zero condition constraints and modified interactively. As with the hard-decision list decoding of Reed–Solomon codes, there are from (5.11) a total of C iterations, after which the minimal polynomial in the group is chosen as the interpolated polynomial Q. According to the iterative process analysis given in Section 5.2.4 and also [13], the interpolated polynomial Q has leading order $\text{lod}(Q) \leq C$. This indicates that those polynomials with leading order greater than C will not be the chosen candidates. Also, if there is a polynomial in the group with leading order greater than C during the iterations, the chosen polynomial Q will not be modified with this polynomial, otherwise $\text{lod}(Q) > C$. Therefore, those polynomials with leading order greater than C can be eliminated from the group during iterations in order to save unnecessary computations.

If $f \in F_q[x, y, z]$ has leading monomial $\text{LM}(f) = \phi_{a'}z^{b'}$, polynomials in $F_q[x, y, z]$ can be partitioned into the following classes according to their leading monomial's z-degree b' and $\phi_{a'}$'s pole order $v_{p_\infty}(\phi_{a'}^{-1})$ [18]:

$$\begin{aligned} V_{\lambda+w\delta} = \{ & f \in F_q[x, y, z] | b' = \delta \wedge v_{p_\infty}(\phi_{a'}^{-1}) = uw + \lambda, \\ & \text{LM}(f) = \phi_{a'}z^{b'}, (\delta, u, \lambda) \in \mathbb{N},\ \lambda < w\}, \end{aligned} \tag{5.82}$$

such that $F_q[x, y, z] = \bigcup\limits_{\lambda<w} V_{\lambda+w\delta}$. The factorization outputs are the z-roots of Q. Therefore, the z-degree of Q is less than or equal to the maximal number of the output list l_m (5.67) and Q is a polynomial chosen from the following classes:

$$V_j = V_{\lambda+w\delta} \quad (0 \leq \lambda < w, 0 \leq \delta \leq l_m). \tag{5.83}$$

At the beginning of the iterative process, a group of polynomials are initialized to represent each of the polynomial classes defined by (5.83) as:

$$G = \{Q_j = Q_{\lambda+w\delta} = y^\lambda z^\delta,\ Q_j \in V_j\}. \tag{5.84}$$

During the iterations, each polynomial Q_j in the group G is the minimal polynomial within its class V_j that satisfies all the tested zero conditions. At the beginning of each iteration, the polynomial group G is modified by [18]:

$$G = \{Q_j | \text{lod}(Q_j) \leq C\} \tag{5.85}$$

in order to eliminate those polynomials with leading order greater than C. Then the remaining polynomials in G are tested by the zero condition constraint defined by (5.62) as:

$$\Delta_j = D_{uv}^{(p_i, r_i)}(Q_j). \tag{5.86}$$

The determined corresponding coefficients $\gamma_{a,p_i,u}$ are applied for this calculation. Those polynomials with $\Delta_j = 0$ satisfy the zero condition and do not need to be modified. However, those polynomials with $\Delta_j \neq 0$ need to be modified. Among them, the index of the minimal polynomial is found as j' and the minimal polynomial is recorded as Q':

$$j' = \text{index}\,(\min_{\text{lod}(Q_j)}\{Q_j | \Delta_j \neq 0\}) \tag{5.87}$$

$$Q' = Q_{j'}. \tag{5.88}$$

$Q_{j'}$ is modified as:

$$Q_{j'} = (x - x_i)Q', \tag{5.89}$$

where x_i is the x-coordinate of affine point p_i which is included in the current interpolated unit (p_i, r_i). The modified $Q_{j'}$ satisfies $D_{uv}^{(p_i,r_i)}(Q_{j'}) = 0$. Based on Properties 1 and 2 mentioned in Section 5.2.3, $D_{uv}^{(p_i,r_i)}[(x - x_i)Q'] = D_{uv}^{(p_i,r_i)}(xQ') - x_i D_{uv}^{(p_i,r_i)}(Q') = x_i D_{j'} - x_i \Delta_{j'} = 0$. The rest of the polynomials with $\Delta_j \neq 0$ are modified as:

$$Q_j = \Delta_{j'} Q_j - \Delta_j Q'. \tag{5.90}$$

The modified Q_j satisfies $D_{uv}^{(p_i,r_i)}(Q_j) = 0$ because $D_{uv}^{(p_i,r_i)}[\Delta_{j'} Q_j - \Delta_j Q'] = \Delta_{j'} D_{uv}^{(p_i,r_i)}(Q_j) - \Delta_j D_{uv}^{(p_i,r_i)}(Q') = \Delta_{j'} - \Delta_{j'} = 0$. After C iterations, the minimal polynomial in the group G is chosen as the interpolated polynomial Q:

$$Q = \min_{\text{lod}(Q_j)}\{Q_j | Q_j \in G\}. \tag{5.91}$$

From the above description, it can be seen that by applying the complexity reduction scheme in (5.85) the zero condition calculation from (5.86) and modifications from (5.89) and (5.90), for those polynomials Q_j with lod(Q_j) $> C$, can be avoided and therefore the interpolation efficiency can be improved. According to [13], this complexity reduction scheme is error-dependent, so that it reduces complexity more significantly in low-error-weight situations. This is because the modification scheme of (5.89) takes action in earlier iteration steps for low-error-weight situations, and therefore computation can be reduced. Figure 5.7 shows interpolation complexity reduction (with different multiplicity m) when applying the scheme in (5.85) to decode the (64, 19, 40) Hermitian code. It is shown that complexity can be reduced

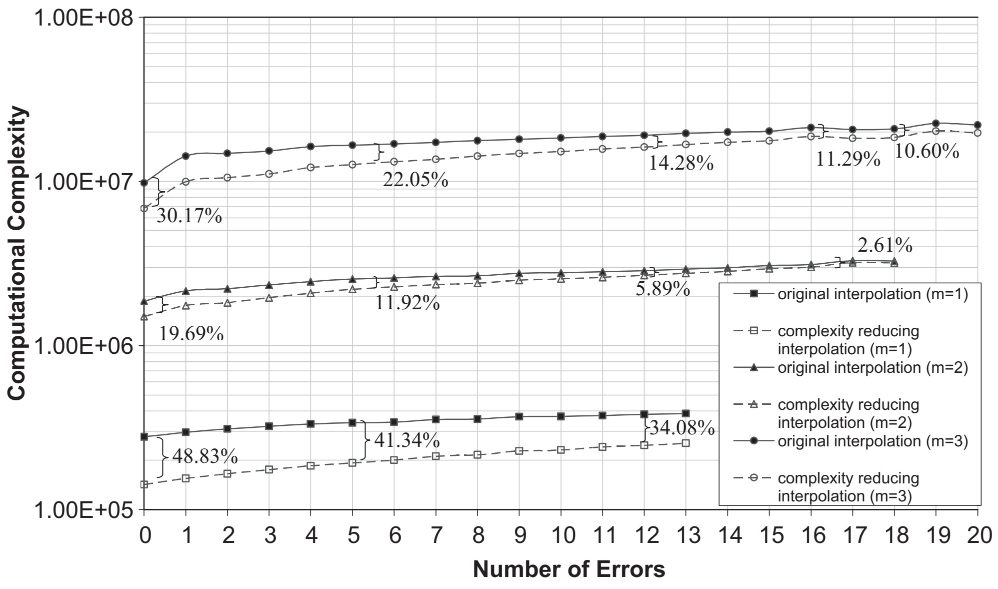


Figure 5.7 Complexity analysis for the interpolation of GS decoding Hermitian code (64, 19, 40).

significantly in low-error-weight situations, especially when $m = 1$ where complexity can be reduced up to 48.83%. However, in high-error-weight situations, complexity reduction is not as significant. Based on Figure 5.7, it can also be observed that the complexity reduction also depends on the interpolation multiplicity m. When $m = 1$, complexity reduction is most significant; when $m = 2$, complexity reduction is the most marginal.

Summarizing Sections 5.5 and 5.6, the modified complexity reduction interpolation process for GS decoding Hermitian codes can be stated as follows:

Initial computation: Apply Algorithm 5.4 to determine all the necessary corresponding coefficients $\gamma_{a,p_i,u}$ and store them for use by the iterative polynomial construction algorithm (Algorithm 5.5).

Algorithm 5.5: Iterative polynomial construction [14, 18] *Initialization*:: Initialize the group of polynomials G with (5.84).

```
1. For each interpolated unit $(p_i, r_i)$ $(i = 0, 1, \dots, n-1)$
   {
2. For each pair of the zero condition parameters $(u, v)$ $(u + v < m)$
   {
3. Modify polynomial group $G$ by (5.85).
4. Test the zero condition $\Delta_j$ of each polynomial in $G$ with (5.86).
5. For polynomials $Q_j$ with $\Delta_j \neq 0$
   {
```

6. Denote the minimal polynomial's index as j' by (5.87) and record it as Q' by (5.88).
7. If $j = j'$, Q_j is modified by (5.89).
8. If $j \neq j'$, Q_j is modified by (5.90).
   }}}

At the end of the iterations, the minimal polynomial Q is chosen from the group G, as in (5.91).

Example 5.8 illustrates this complexity reduction interpolation process.

Example 5.8: Decode the (8, 4, 4) Hermitian code defined in GF(4) using the GS algorithm with interpolation multiplicity $m = 2$.

The Hermitian code word is generated by evaluating the message polynomial over the following affine points: $p_0 = (0, 0), p_1 = (0, 1), p_2 = (1, \alpha), p_3 = (1, \alpha^2)$, $p_4 = (\alpha, \alpha), p_5 = (\alpha, \alpha^2), p_6 = (\alpha^2, \alpha), p_7 = (\alpha^2, \alpha^2)$. The received word is given as $R = (1, \alpha^2, \alpha, \alpha^2, \alpha, \alpha^2, \alpha^2, \alpha)$.

Applying (5.11), the iteration number $C = 8\begin{pmatrix}3\\2\end{pmatrix} = 24$. Based on C, the length of the output list can be determined as $l_2 = 3$, and of parameter t_2 as $t_2 = 1$, by using (5.67) and (5.68) respectively. As a result, the (1, 4)-weighted degree upper bound for the interpolated polynomial can be determined by (5.66) as $\max\{\deg_{1,4} Q\} = 13$. Therefore, the maximal pole basis (L_2) monomial that might exist in the interpolated polynomial is $\phi_{\max} = \phi_{12} = x^2y^3$. As the interpolation multiplicity $m = 2$, Algorithm 5.4 is applied to determine the corresponding coefficients $\gamma_{0,p_i,u} - \gamma_{12,p_i,u}$ and $\gamma_{0,p_i,u} - \gamma_{12,p_i,u}$ $(u < 2)$ are stored for the following interpolation process. Table 5.4 lists all the resulted corresponding coefficients $\gamma_{0,p_i,u} \sim \gamma_{12,p_i,u}$ $(u < 2)$.

Following on, Algorithm 5.5 is performed to find the interpolated polynomial $Q(x, y, z)$. At the beginning, a group of polynomials is initialized as: $Q_0 = 1$, $Q_1 = y$, $Q_2 = z$, $Q_3 = yz$, $Q_4 = z^2$, $Q_5 = yz^2$, $Q_6 = z^3$, $Q_7 = yz^3$. Their leading orders are: $\text{lod}(Q_0) = 0$, $\text{lod}(Q_1) = 2$, $\text{lod}(Q_2) = 4$, $\text{lod}(Q_3) = 9$, $\text{lod}(Q_4) = 12$, $\text{lod}(Q_5) = 20$, $\text{lod}(Q_6) = 24$, $\text{lod}(Q_7) = 35$.

For interpolated unit, $(p_0, r_0) = ((0, 0), 1)$.

For the zero parameter, $u = 0$ and $v = 0$.

As $\text{lod}(Q_7) > C$, polynomial Q_7 is eliminated from the group.

Test the zero condition of the remaining polynomials in the group as:

$$\Delta_0 = D_{00}^{(p_0,r_0)}(Q_0) = 1,\ \Delta_1 = D_{00}^{(p_0,r_0)}(Q_1) = 0,\ \Delta_2 = D_{00}^{(p_0,r_0)}(Q_2) = 1,$$
$$\Delta_3 = D_{00}^{(p_0,r_0)}(Q_3) = 0,$$
$$\Delta_4 = D_{00}^{(p_0,r_0)}(Q_4) = 1,\ \Delta_5 = D_{00}^{(p_0,r_0)}(Q_5) = 0,\ \Delta_6 = D_{00}^{(p_0,r_0)}(Q_6) = 1.$$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$$j' = 0 \quad \text{and} \quad Q' = Q_0.$$

Table 5.4 Predetermined corresponding coefficients for Example 5.8.

$p_0, \gamma_{a,p_0,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	1	0	0	0	0	0	0	0	0	0	0	0
$p_1, \gamma_{a,p_1,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	0	1	0	0	1	0	0	1	0	0	1	0
	1	0	1	0	0	1	0	0	1	0	0	1	0	0
$p_2, \gamma_{a,p_2,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	α	1	α	α^2	α	α^2	1	α^2	1	α	1
	1	0	1	1	0	α^2	0	1	α^2	α^2	0	α	0	α^2
$p_3, \gamma_{a,p_3,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	α^2	1	α^2	α	α^2	α	1	α	1	α^2	1
	1	0	1	1	0	α	0	1	α	α	0	α^2	0	α
$p_4, \gamma_{a,p_4,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	α	α	α^2	α^2	α^2	1	1	1	α	α	α	α^2
	1	0	1	α^2	0	α^2	0	α	α^2	α	0	α	0	1
$p_5, \gamma_{a,p_5,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	α	α^2	α^2	1	α	α	α^2	1	1	α	α^2	α^2
	1	0	1	α^2	0	α	0	α	α	1	0	α^2	0	α^2
$p_6, \gamma_{a,p_6,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	α^2	α	α	1	α^2	α^2	α	1	1	α^2	α	α
	1	0	1	α	0	α^2	0	α^2	α^2	1	0	α	0	α
$p_7, \gamma_{a,p_7,u}$	a/u	0	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	α^2	α^2	α	α	α	1	1	1	α^2	α^2	α^2	α
	1	0	1	α	0	α	0	α^2	α	α^2	0	α^2	0	1

As $\Delta_1 = \Delta_3 = \Delta_5 = 0$:

$$Q_1 = Q_1 = y, \text{ and } \operatorname{lod}(Q_1) = 2$$
$$Q_3 = Q_3 = yz, \text{ and } \operatorname{lod}(Q_3) = 9$$
$$Q_5 = Q_5 = yz^2, \text{ and } \operatorname{lod}(Q_5) = 20.$$

Modify polynomials in the group with $\Delta_j \neq 0$ as:

$$Q_0 = (x - 0)Q' = x, \text{ and } \operatorname{lod}(Q_0) = 1$$
$$Q_2 = \Delta_0 Q_2 - \Delta_2 Q' = 1 + z, \text{ and } \operatorname{lod}(Q_2) = 4$$
$$Q_4 = \Delta_0 Q_4 - \Delta_4 Q' = 1 + z_2, \text{ and } \operatorname{lod}(Q_4) = 12$$
$$Q_6 = \Delta_0 Q_6 - \Delta_6 Q' = 1 + z_3 \text{ and } \operatorname{lod}(Q_6) = 24.$$

For the zero parameter, $u = 0$ and $v = 1$.

As there is no polynomial in the group with leading order over C, no polynomial is eliminated in this iteration.

Test the zero condition of the remaining polynomials in the group as:

$$\Delta_0 = D_{01}^{(p_0,r_0)}(Q_0) = 0,\ \Delta_1 = D_{01}^{(p_0,r_0)}(Q_1) = 0,\ \Delta_2 = D_{01}^{(p_0,r_0)}(Q_2) = 1,$$
$$\Delta_3 = D_{01}^{(p_0,r_0)}(Q_3) = 0,$$
$$\Delta_4 = D_{01}^{(p_0,r_0)}(Q_4) = 0,\ \Delta_5 = D_{01}^{(p_0,r_0)}(Q_5) = 0,\ \Delta_6 = D_{01}^{(p_0,r_0)}(Q_6) = 1.$$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$$j' = 2 \quad \text{and} \quad Q' = Q_2.$$

As $\Delta_0 = \Delta_1 = \Delta_3 = \Delta_4 = \Delta_5 = 0$:

$$Q_0 = Q_0 = x,\ \text{and lod}(Q_0) = 1$$
$$Q_1 = Q_1 = y,\ \text{and lod}(Q_1) = 2$$
$$Q_3 = Q_3 = y_z,\ \text{and lod}(Q_3) = 9$$
$$Q_4 = Q_4 = 1 + z_2,\ \text{and lod}(Q_4) = 12$$
$$Q_5 = Q_5 = yz^2 \text{ and lod}(Q_5) = 20.$$

Modify polynomials in the group with $\Delta_j \neq 0$ as:

$$Q_2 = (x - 0)Q' = x + xz \text{ and lod}(Q_2) = 7$$
$$Q_6 = \Delta_2 Q_6 - \Delta_6 Q' = z + z^3 \text{ and lod}(Q_6) = 24.$$

For zero parameter, $u = 1$ and $v = 0$.

As there is no polynomial in the group with leading order over C, no polynomial is eliminated in this iteration.

Test the zero condition of the remaining polynomials in the group as:

$$\Delta_0 = D_{10}^{(p_0,r_0)}(Q_0) = 1,\ \Delta_1 = D_{10}^{(p_0,r_0)}(Q_1) = 0, \Delta_2 = D_{10}^{(p_0,r_0)}(Q_2) = 0,$$
$$\Delta_3 = D_{10}^{(p_0,r_0)}(Q_3) = 0,$$
$$\Delta_4 = D_{10}^{(p_0,r_0)}(Q_4) = 0,\ \Delta_5 = D_{10}^{(p_0,r_0)}(Q_5) = 0,\ \Delta_6 = D_{10}^{(p_0,r_0)}(Q_6) = 0.$$

Find the minimal polynomial with $\Delta_j \neq 0$ as:

$$j' = 0 \quad \text{and} \quad Q' = Q_0.$$

As $\Delta_1 = \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = \Delta_6 = 0$:

$$Q_1 = Q_1 = y,\ \text{and lod}(Q_1) = 2$$
$$Q_2 = Q_2 = x + xz,\ \text{and lod}(Q_2) = 7$$
$$Q_3 = Q_3 = yz,\ \text{and lod}(Q_3) = 9$$
$$Q_4 = Q_4 = 1 + z^2,\ \text{and lod}(Q_4) = 12$$
$$Q_5 = Q_5 = yz^2,\ \text{and lod}(Q_5) = 20$$
$$Q_6 = Q_6 = z + z^3,\ \text{and lod}(Q_6) = 24.$$

Modify polynomials in the group with $\Delta_j \neq 0$ as:

$$Q_0 = (x - 0)Q' = x^2, \text{ and } \mathrm{lod}(Q_0) = 3.$$

Following the same process, interpolation runs through the rest of the interpolated units $(p_1, r_1) \sim (p_7, r_7)$ and with respect to all zero parameters $(u, v) = (0, 0), (0, 1)$ and $(1, 0)$. After C iterations, the chosen interpolated polynomial is: $Q(x, y, z) = \alpha^2 + \alpha x + y + \alpha x^2 + y^2 + \alpha^2 x^2 y + \alpha^2 x y^2 + \alpha^2 y^3 + \alpha^2 x^2 y^2 + z(x + xy + xy^2) + z^2(\alpha^2 + \alpha x + \alpha^2 y + \alpha x^2)$, and $\mathrm{lod}(Q(x, y, z)) = 23$. Polynomial $Q(x, y, z)$ has a zero of multiplicity at least 2 over the eight interpolated units.

5.7 General Factorization

Based on the interpolated polynomial, factorization finds the polynomial's z-roots in order to determine the output list. Building upon the work of [15, 20], this section presents a generalized factorization algorithm, or the so-called recursive coefficient search algorithm, which can be applied to both Reed–Solomon codes and algebraic–geometric codes. This section's work is based on [21]. In general, the algorithm is described with application to algebraic–geometric codes. Therefore, it has to be stated that when applying this algorithm to Reed–Solomon codes, the rational functions in an algebraic–geometric code's pole basis are simplified to univariate monomials in the Reed–Solomon code's pole basis. As a consequence, polynomials in $F_q[x, y]$ are simplified to univariate polynomials with variable x.

Those polynomials $h \in F_q^{w_z}[x, y]$ will be in the output list if $Q(x, y, h) = 0$. The outcome of the factorization can be written as:

$$\begin{cases} h_1 = h_{1,0}\phi_0 + \cdots + h_{1,k-1}\phi_{k-1} \\ \qquad\vdots \\ h_l = h_{l,0}\phi_0 + \cdots + h_{l,k-1}\phi_{k-1} \end{cases}, \tag{5.92}$$

with $l \leq l_m$. Rational functions $\phi_0, \ldots, \phi_{k-1}$ are predetermined by the decoder; therefore, finding the list of polynomials is equivalent to finding their coefficients, $h_{1,0}, \ldots, h_{1,k-1}, \ldots, h_{l,0}, \ldots, h_{l,k-1}$, respectively. Substituting h into the interpolated polynomial $Q = \sum_{a,b} Q_{ab}\phi_a z^b$, we have:

$$Q(x, y, h) = \sum_{a,b} Q_{ab}\phi_a h^b = \sum_{a,b} Q_{ab}\phi_a (h_0\phi_0 + \cdots + h_{k-1}\phi_{k-1})^b. \tag{5.93}$$

It is important to notice that:

$$(\phi_i\phi_j) \mod \zeta = \sum_v \phi_v, \tag{5.94}$$

where χ is the algebraic curve (e.g. Hermitian curve H_w) and ϕ_i, ϕ_j and ϕ_v are rational functions in the pole basis associated with the curve χ (for example the pole basis L_w associated with curve H_w). Therefore (5.93) can be rewritten as a polynomial in $F_q[x, y]$:

$$Q(x, y, h) = \sum_a Q_a \phi_a, \tag{5.95}$$

where coefficients Q_a are equations with unknowns $h_0, \ldots, h_{k-1}$. If $T = |\{Q_a | Q_a \neq 0\}|$, the rational functions ϕ_a with $Q_a \neq 0$ can be arranged as $\phi_{a_1} < \phi_{a_2} < \cdots < \phi_{a_T}$ and (5.95) can again be rewritten as:

$$Q(x, y, h) = Q_{a_1}\phi_{a_1} + Q_{a_2}\phi_{a_2} + \cdots + Q_{a_T}\phi_{a_T}. \tag{5.96}$$

Again, coefficients $Q_{a_1}, Q_{a_2}, \ldots,$ and Q_{a_T} are equations of unknowns $h_0, \ldots, h_{k-1}$. To have $Q(x, y, h) = 0$, we need $Q_{a_1} = Q_{a_2} = \cdots = Q_{a_T} = 0$. Therefore, $h_0, \ldots, h_{k-1}$ can be determined by solving the following simultaneous set of equations [21]:

$$\begin{cases} Q_{a_1}(h_0, \ldots, h_{k-1}) = 0 \\ Q_{a_2}(h_0, \ldots, h_{k-1}) = 0 \\ \qquad \vdots \\ Q_{a_T}(h_0, \ldots, h_{k-1}) = 0 \end{cases}. \tag{5.97}$$

In order to solve (5.97), a recursive coefficient search algorithm is applied to determine $h_0, \ldots, h_{k-1}$ [20, 22]. Here a more general and efficient factorization algorithm is presented. Let us denote the following polynomials with respect to a recursive index s $(0 \leq s \leq k-1)$:

$$h^{(s)}(x, y) = h_0\phi_0 + \cdots + h_{k-1-s}k_{-1-s}, \tag{5.98}$$

which is a candidate polynomial with coefficients $h_0, \ldots, h_{k-1-s}$ undetermined. Update $Q(x, y, z)$ recursively as:

$$Q^{(s+1)}(x, y, z) = Q^{(s)}(x, y, z + h_{k-1-s}\phi_{k-1-s}), \tag{5.99}$$

with $Q^{(0)}(x, y, z) = Q(x, y, z)$, which is the interpolated polynomial (5.68). Substituting $h_{k-1-s}\phi_{k-1-s}$ into $Q^{(s)}(x, y, z)$, we have:

$$\tilde{Q}^{(s)}(x, y) = Q^{(s)}(x, y, h_{k-1-s}\phi_{k-1-s}). \tag{5.100}$$

$\tilde{Q}^{(s)}$ mod χ can be transferred into a polynomial in $F_q[x,\ y]$ with coefficients expressed as $\sum_i \varpi_i h^i_{k-1-s}$ where $\varpi_i \in \mathrm{GF}(q)$. Denote $\tilde{Q}^{(s)}$'s leading monomial with its leading coefficient as [21]:

$$\phi_L^{(s)} = \mathrm{LM}(\tilde{Q}^{(s)}) \tag{5.101}$$

$$C_L^{(s)}(h_{k-1-s}) = LC(\tilde{Q}^{(s)}). \tag{5.102}$$

Based on (5.102), it can be seen that $\mathrm{LM}(h^{(s)}) = \phi_{k-1-s}$ and $\mathrm{LC}(h^{(s)}) = h_{k-1-s}$. Therefore, for any recursive polynomial $Q^{(s)}(x, y, z)$, we have:

$$\mathrm{LM}(Q^{(s)}(x, y, h^{(s)})) = \mathrm{LM}(Q^{(s)}(x, y, h_{k-1-s}\phi_{k-1-s})) = \mathrm{LM}(\tilde{Q}^{(s)}) = \phi_L^{(s)}) \tag{5.103}$$

$$\mathrm{LC}(Q^{(s)}(x, y, h^{(s)})) = \mathrm{LC}(Q^{(s)}(x, y, h_{k-1-s}\,\phi_{k-1-s})) = \mathrm{LC}(\tilde{Q}^{(s)}) = C_L^{(s)}(h_{k-1-s}). \tag{5.104}$$

As all the candidate outputs should satisfy $Q(x, y, h) = 0$, and from the above definitions it can be seen that $h = h^{(0)}$ and $Q^{(0)}(x, y, z) = Q(x, y, z)$, $Q(x, y, h) = 0$ is equivalent to $Q^{(0)}(x, y, h^{(0)}) = 0$. Based on (5.107) and (5.108), in order to have $Q^{(0)}(x, y, h^{(0)}) = 0$ we need to find its leading monomial $\phi_L^{(0)}$ with leading coefficient $C_L^{(0)}(h_{k-1})$ and determine the values of h_{k-1} that satisfy $C_L^{(0)}(h_{k-1}) = 0$. As a result, the leading monomial of $Q^{(0)}(x, y, h^{(0)})$ is eliminated. Based on each value of h_{k-1} and performing the polynomial update (5.99), $Q^{(1)}(x, y, z)$ is generated, in which $\phi_L^{(0)}$ has been eliminated. Now, $Q(x, y, h) = 0$ is equivalent to $Q^{(1)}(x, y, h^{(1)}) = 0$. Again, to have $Q^{(1)}(x, y, h^{(1)}) = 0$, we need $C_L^{(1)}(h_{k-2}) = 0$. Therefore, h_{k-2} can be determined by solving $C_L^{(1)}(h_{k-2}) = 0$. Based on each value of h_{k-2}, we can trace further to find the rest of the coefficients. In general, after the coefficients h_{k-1-s} $(0 \leq s < k - 1)$ have been determined by solving $C_L^{(s)}(h_{k-1-s}) = 0$, based on each value of them, perform the polynomial update (5.99) to generate $Q^{(s+1)}(x, y, z)$. From $Q^{(s+1)}(x, y, z)$, $\tilde{Q}^{(s+1)}$ can be calculated and $h_{k-1-(s+1)}$ can be determined by solving $C_L^{(s+1)}(h_{k-1-(s+1)}) = 0$. This process is illustrated in Figure 5.8.

From Figure 5.8 it can be seen that there might be an exponential number of routes to find coefficients $h_{k-1}, \ldots, h_0$. However, not every route will be able to reach h_0, as during the recursive process there may be no solution for $C_L^{(s)}(h_{k-1-s}) = 0 = 0$. If h_0 is produced and $Q^{(k-1)}(x, y, h_0\phi_0) = 0$, this route can be traced to find the rest of the

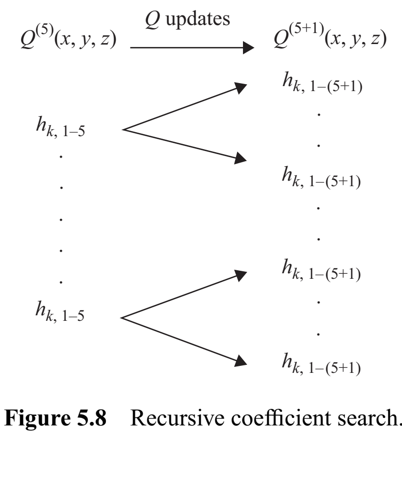


Figure 5.8 Recursive coefficient search.

coefficients $h_1, \ldots, h_{k-1}$ to construct polynomial h which will satisfy $Q(x, y, h) = 0$. The correctness of this judgement will be proven later.

Based on the above description, the generalized factorization algorithm is summarized in Algorithm 5.6.

Algorithm 5.6: Recursive coefficient search [14, 21, 22] *Initialization*: $Q^{(0)}(x, y, z) = Q(x, y, z)$. The recursive index $s = 0$ and output candidate index $l = 1$.

Perform: Recursive coefficient search (s) ($RCS(s)$).

Recursive coefficient search (RCS):

Input parameter: s ($0 \le s \le k - 1$).

```
1. Perform (5.100) to calculate $\tilde{Q}^{(s)}(x, y)$.
2. Find out $\phi_L^{(s)}$ with its coefficient $C_L^{(s)}(h_{k-1-s})$.
3. Determine $h_{k-1-s}$ by solving $C_L^{(s)}(h_{k-1-s}) = 0$.
4. For each value of $h_{k-1-s}$, do
   {
5. $h_{l,k-1-s} = h_{k-1-s}$.
6. If $s = k - 1$, calculate $Q^{(k-1)}(x, y, h_0\phi_0)$ and go to step 7. Else go to step 8.
7. If $Q^{(k-1)}(x, y, h_0\phi_0) = 0$, trace this route to find coefficients $h_{l,k-1}, h_{l,k-2}, \ldots,$ and
   $h_{l,0}$ to construct the candidate polynomial $h_l$ and $l = l + 1$. Else stop this route.
8. Perform polynomial update (5.99) to generate $Q^{(s+1)}(x, y, z)$.
9. Perform $RCS(s + 1)$.
   }
```

If a number of h_{k-1-s} have been determined, the algorithm will choose one of them to determine the rest of the coefficients until all the possible routes starting from this h_{k-1-s} have been traced. After this, it will choose another value of h_{k-1-s} and repeat the process. This algorithm will terminate after all the possible routes started from h_{k-1} have been traced. To prove the correctness of this algorithm, we need to justify the fact that the polynomial h_l produced in step 7 satisfies $Q(x, y, h_l) = 0$.

Proof: As $Q^{(k-1)}(x, y, h_0\phi_0) = 0$ and $h^{(k-1)}(x, y) = h_0\phi_0$, we have $Q^{(k-1)}(x, y, h^{(k-1)}) = 0$. Assuming h_1 is the previously-found coefficient, $Q^{(k-1)}(x, y, z)$ is generated by (5.99): $Q^{(k-1)}(x, y, z) = Q^{(k-2)}(x, y, z + h_1\phi_1)$. From $Q^{(k-1)}(x, y, h^{(k-1)}) = 0$ we have $Q^{(k-2)}(x, y, h^{(k-1)} + h_1\phi_1) = 0$. Based on (5.98), it can be seen that $h^{(k-2)} = h_0\phi_0 + h_1\phi_1 = h^{(k-1)} + h_1\phi_1$. Therefore, $Q^{(k-2)}(x, y, h^{(k-2)}) = 0$. It can be deduced further to get $Q^{(k-3)}(x, y, h^{(k-3)}) = 0, \ldots,$ and $Q^{(0)}(x, y, h^{(0)}) = 0$. As $Q^{(0)}(x, y, z) = Q(z)$ and $h(0) = h_0\phi_0 + h_1\phi_1 + \cdots + h_{k-1}\phi_{k-1}$, whose coefficients have been traced as the coefficients of the output candidate h_l, it can be concluded that $Q(x, y, h_l) = 0$.

Here two worked examples are given to illustrate the generalized factorization algorithm applied to both a Hermitian code and a Reed–Solomon code.

Example 5.9: List decoding of a (8, 4, 4) Hermitian code defined in GF(4)
Given the interpolated polynomial is $Q(x, y, z) = \alpha^2y + \alpha^2x^2 + \alpha xy + \alpha^2y^2 + \alpha^2x^2y + x^2y^2 + xy^3 + (\alpha x + \alpha xy + \alpha xy^2)z + (x + x^2)z^2$, apply Algorithm 5.6 to find out its z-roots.

Initialization: $Q^{(0)}(x, y, z) = Q(x, y, z)$, $s = 0$ and $l = 1$.

$RCS(0)$:

$\tilde{Q}^{(0)}(x, y) = Q^{(0)}(x, y, h_3x^2) = (\alpha^2 + \alpha h_3)y + \alpha^2x^2 + \alpha xy + (\alpha^2 + h_3^2)y^2 + (\alpha^2 + h_3^2)x^2y + (1 + h_3^2)x^2y^2 + xy^3 + (\alpha h_3 + h_3^2)y^4$, with $\phi_L^{(0)} = y^4$ and $C_L^{(0)}(h_3) = \alpha h_3 + h_3^2$. Solving $C_L^{(0)}(h_3) = 0$, we have $h_3 = 0$ or $h_3 = \alpha$.

For $h_3 = 0$, $h_{1,3} = h_3 = 0$. As $s = 0 < 3$, update $Q^{(1)}(x, y, z) = Q^{(0)}(x, y, z + 0x^2) = Q(x, y, z)$, and perform $RCS(1)$...

Based on the same progress, the outcomes from $RCS(1)$, $RCS(2)$ and $RCS(3)$ are summarized in Table 5.5.

Table 5.5 Recursive coefficient search from $h_3 = 0$.

RCS(s)	$\phi_L^{(s)}$	$C_L^{(s)}(h_{k-1-s})$	$h_{2,k-1-s} = h_{k-1-s}$
$RCS(1)$	xy^3	$1 + \alpha h_2$	α^2
$RCS(2)$	x^2y^2	$\alpha^2 + \alpha h_1$	α
$RCS(3)$	xy^2	αh_0	0

After $RCS(2)$ we have $Q^{(3)}(x, y, z) = (\alpha x + \alpha xy + \alpha xy^2)z + (x + x^2)z^2$. In $RCS(3)$, by solving $C_L^{(3)}(h_0) = 0$, we have $h_0 = 0$. Therefore, $h_{1,0} = h_0 = 0$. As $s = 3$ and $Q^{(3)}(x, y, h_0\phi_0) = Q^{(3)}(x, y, 0\cdot 1) = 0$, this route can be traced to construct candidate polynomial $h_1 = \alpha x + \alpha^2y$ and update the candidate index $l = l + 1 = 2$.

Going back to the closest division point (when $s = 0$), we have:

For $h_3 = \alpha$, $h_{2,3} = h_3 = \alpha$. As $s = 0 < 3$, update $Q^{(1)}(x, y, z) = Q^{(0)}(x, y, z + \alpha x^2) = \alpha^2x^2 + \alpha xy + \alpha x^2y^2 + xy^3 + (\alpha x + \alpha xy + \alpha xy^2)z + (x + x^2)z^2$ and perform $RCS(1)$...

Again, the outcomes of $RCS(1)$, $RCS(2)$ and $RCS(3)$ are summarized in Table 5.6.

Table 5.6 Recursive coefficient search from $h_3 = \alpha$.

RCS(s)	$\phi_L^{(s)}$	$C_L^{(s)}(h_{k-1-s})$	$h_{2,k-1-s} = h_{k-1-s}$
$RCS(1)$	xy^3	$1 + \alpha h_2$	α^2
$RCS(2)$	x^2y^2	αh_1	0
$RCS(3)$	xy^2	$\alpha^2 + \alpha h_0$	α

After $RCS(2)$ we have $Q^{(3)}(x, y, z) = \alpha^2x^2 + \alpha^2xy + \alpha^2xy^2 + (\alpha x + \alpha xy + \alpha xy^2)z + (x + x^2)z^2$. In $RCS(3)$, by solving $C_L^{(3)}(h_0) = 0$, we have $h_{2,0} = h_0 = \alpha$.

As $s = 3$ and $Q^{(3)}(x, y, h_0\phi_0) = Q^{(3)}(x, y, \alpha\cdot 1) = 0$, this route can be traced to construct the candidate polynomial $h_2 = \alpha + \alpha^2 y + \alpha x^2$. As all the possible routes from h_0 have been traced, the factorization process terminates and outputs: $h_1 = \alpha x + \alpha^2 y$, $h_2 = \alpha + \alpha^2 y + \alpha x^2$.

Example 5.10: List decoding of a (7, 2, 6) Reed–Solomon code defined in GF(8)
α is a primitive element in GF(8) satisfying $\alpha^3 + \alpha + 1 = 0$.

Given the interpolated polynomial $Q(x, z) = \alpha x + \alpha^6 x^2 + (\alpha^3 + \alpha^3 x)z + \alpha^2 z^2$, apply Algorithm 5.6 to determine its z-roots.

Initialization: $Q^{(0)}(x, z) = Q(x, z)$, $s = 0$ and $l = 1$.

RCS(0):

$\tilde{Q}^{(0)}(x, z) = Q^{(0)}(x, h_1 x) = (\alpha + \alpha^3 h_1)x + (\alpha^6 + \alpha^3 h_1 + \alpha^2 h_1^2)x^2$, with $\phi_L^{(0)} = x^2$ and $C_L^{(0)}(h_1) = \alpha^6 + \alpha^3 h_1 + \alpha^2 h_1^2$. Solving $C_L^{(0)}(h_1) = 0$, we have $h_1 = \alpha^5$ or $h_1 = \alpha^6$.

For $h_1 = \alpha^5$, $h_{1,1} = h_1 = \alpha^5$. As $s = 0 < 1$, update $Q^{(1)}(x, z) = Q^{(0)}(x, z + \alpha^5 x)$ $= (\alpha^3 + \alpha^3 x)z + \alpha^2 z^2$ and perform *RCS*(1).

In *RCS*(1), following the same progress, we have $\phi_L^{(1)} = x$ and $C_L^{(1)}(h_0) = \alpha^3 h_0$. Solving $C_L^{(1)}(h_0) = 0$, we have $h_0 = 0$.

For $h_0 = 0$, $h_{1,0} = h_0 = 0$. As $s = 1$ and $Q^{(1)}(x, h_0\phi_0) = Q^{(1)}(x, 0\cdot 1) = 0$, this route can be traced to construct candidate polynomial $h_1 = \alpha^5 x$. Update the output candidate index as $l = l + 1 = 2$.

Going back to the closest division point (when $s = 0$), we have:

For $h_1 = \alpha^6$, $h_{2,1} = h_1 = \alpha^6$. As $s = 0 < 1$, update $Q^{(1)}(x, z) = Q^{(0)}(x, z + \alpha^6 x)$ $= \alpha^4 x + (\alpha^3 + \alpha^3 x)z + \alpha^2 z^2$, and perform *RCS*(1).

In *RCS*(1) we have $\phi_L^{(1)} = x$ and $C_L^{(1)}(h_0) = \alpha^4 + \alpha^3 h_0$. Solving $C_L^{(1)}(h_0) = 0$, we have $h_0 = \alpha$.

For $h_0 = \alpha$, $h_{2,0} = h_0 = \alpha$. As $s = 1$ and $Q^{(1)}(x, h_0\phi_0) = Q^{(1)}(x, \alpha\cdot 1) = 0$, this route can be traced to construct candidate polynomial $h_2 = \alpha + \alpha^6 x$. As all the possible routes from h_0 have been traced, the factorization process terminates and outputs: $h_1 = \alpha^5 x$, $h_2 = \alpha + \alpha^6 x$.

5.8 Soft-Decision List Decoding of Hermitian Codes

We can extend the hard-decision GS decoding algorithm for AG codes to soft-decision by extending the Kotter–Vardy algorithm. As with Reed–Solomon codes, soft-decision list decoding is almost the same as hard-decision list decoding, with the addition of a process to convert the reliability values of the received symbols into a multiplicity matrix. In this section, a comparison of the code word scores is made between soft- and hard-decision list decoding and conditions are derived to ensure that the equations from the interpolation process are solvable.

Based on w_z, the two parameters first defined by (5.50) and (5.51) for analyzing bivariate monomial $x^a y^b$ ($a, b \in \mathbb{N}$) can be extended to trivariate monomials $\phi_a z^b$ ($a, b \in \mathbb{N}$):

$$N_{1,w_z}(d) = |\{\phi_a z^b : a, b \geq 0 \text{ and } \deg_{1,w_z}(\phi_a z^b) \leq \delta, \delta \in \mathbb{N}\}|, \tag{5.105}$$

which represents the number of monomials with $(1, w_z)$-weighted degree not greater than δ, and:

$$\Delta_{1,w_z}(v) = \min\{\delta : N_{1,w_z}(\delta) > v, v \in \mathbb{N}\}, \tag{5.106}$$

which represents the minimal value of δ that guarantees $N_{1,w_z}(\delta)$ is greater than v.

The difference between soft-decision list decoding of Reed–Solomon codes and of Hermitian codes is in the size of the reliability matrix. For soft-decision list decoding of Reed–Solomon codes, the reliability matrix $\mathbf{\Pi}$ has size $q \times n$, where $n = q - 1$. For soft-decision list decoding of Hermitian codes, the reliability matrix $\mathbf{\Pi}$ has size $q \times n$, where $n = q^{3/2}$.

5.8.1 System Solution

The reliability matrix $\mathbf{\Pi}$ is then converted to the multiplicity matrix $\boldsymbol{M}$, for which Algorithm 5.3 is applied. A version of Algorithm 5.3 introducing a stopping rule based on the designed length of output list is presented later in this subsection, as Algorithm 5.7.

In this subsection, the code word score with respect to matrix $\boldsymbol{M}$ is analyzed so as to present the system solution for this soft-decision list decoder. It is shown that the soft-decision scheme provides a higher code word score than the hard-decision scheme.

The resulting multiplicity matrix $\boldsymbol{M}$ can be written as:

$$M = \begin{bmatrix} m_{0,0} & m_{0,1} & \cdots & \cdots & \cdots & m_{0,n-1} \\ m_{1,0} & m_{1,1} & & & & m_{1,n-1} \\ \vdots & & \ddots & & & \vdots \\ \vdots & & & m_{i,j} & & \vdots \\ \vdots & & & & \ddots & \vdots \\ m_{q-1,0} & m_{q-1,1} & \cdots & \cdots & \cdots & m_{q-1,n-1} \end{bmatrix}, \tag{5.107}$$

where entry $m_{i,j}$ represents the multiplicity for unit (p_j, ρ_i). Interpolation builds the minimal polynomial $Q_M \in F_q[x, y, z]$, which has a zero of multiplicity at least $m_{i,j}$

($m_{i,j} \neq 0$) over all the associated units (p_j, ρ_i). Following on from (5.62), with respect to interpolated unit (p_j, ρ_i), Q_M's coefficients Q_{ab} should satisfy:

$$\sum_{a,b \geq v} Q_{ab} \binom{b}{v} \gamma_{a,p_j,u} \rho_i^{b-v} = 0, \ \forall \, u, v \in \mathrm{N} \text{ and } u + v < m_{i,j}. \tag{5.108}$$

For this soft-decision interpolation, the number of interpolated units covered by Q_M is:

$$|\{m_{i,j} \neq 0 | m_{i,j} \in \boldsymbol{M}, i = 0, 1, \ldots, q-1 \text{ and } j = 0, 1, \ldots, n-1\}| \tag{5.109}$$

and the cost C_M of multiplicity matrix $\boldsymbol{M}$ is:

$$C_M = \frac{1}{2} \sum_{i=0}^{q-1} \sum_{j=0}^{n-1} m_{i,j}(m_{i,j} + 1), \tag{5.110}$$

which represents the number of constraints in (5.108) to Q_M's coefficients Q_{ab}. These can be imposed by the iterative polynomial construction algorithm [9, 11, 17] in C_M iterations. Notice that (5.109)) and (5.110)) have the same expression as (5.45) and (5.46), respectively, with the exception that $n = q^{3/2}$.

Based on Lemma 5.1, the following units' multiplicities will contribute to the code word score: $(p_0, c_0), (p_1, c_1), \ldots$, and (p_{n-1}, c_{n-1}). Referring to the multiplicity matrix in (5.107), the interpolated polynomial Q_M can be explained as passing through these units with multiplicity at least $m_0 = m_{i,0}$ ($\rho_i = c_0$), $m_1 = m_{i,1}$ ($\rho_i = c_1$), $\ldots$, and $m_{n-1} = m_{i,n-1}$ ($\rho_i = c_{n-1}$), respectively. If $f \in F_q^{w_z}[x, y]$ is the transmitted message polynomial such that $f(p_i) = c_i$, the total zero order of $Q_M(x, y, f)$ over units $\{(p_0, c_0), (p_1, c_1), \ldots, (p_{n-1}, c_{n-1})\}$ is at least:

$$m_0 + m_1 + \cdots + m_{n-1} = \sum_{j=0}^{n-1} \{m_{i,j} | \rho_i = c_j, i = 0, 1, \ldots, q-1\}. \tag{5.111}$$

Therefore, the code word score $S_M(\bar{c})$ with respect to multiplicity matrix $\boldsymbol{M}$ is:

$$S_M(\bar{c}) = \sum_{j=0}^{n-1} \{m_{i,j} | \rho_i = c_j, i = 0, 1, \ldots, q-1\}. \tag{5.112}$$

If $S_M(\bar{c}) > \deg_{1,w_z}(Q_M(x, y, z))$ then $\sum_{i=0}^{n-1} v_{p_i}(Q_M(x, y, f)) > v_{p_\infty}(Q_M(x, y, f)^{-1})$ and $Q_M(x, y, f) = 0$. f can be found by determining $Q_M(x, y, z)$'s z-roots. It results in the following corollary for successful soft-decision list decoding:

Corollary 5.3 *If the code word score with respect to multiplicity matrix **M** is greater than the interpolated polynomial* $\mathrm{Q_M}$*'s (1,* $\mathrm{w_z}$*)-weighted degree [23]:*

$$S_M(\overline{c}) > \deg_{1,w_z}(Q_M(x, y, z)) \tag{5.113}$$

then $Q_M(x, y, f) = 0$ or $z - f | Q_M(x, y, z)$.

To compare the soft-decision's code word score $S_M(\overline{c})$ with the hard-decision's code word score $S_M(\overline{c})$, denote the index of the maximal element in each column of $\mathbf{\Pi}$ as:

$$i_j = \text{index}\,(\max\{\pi_{i,j} | i = 0, 1, \ldots, q-1\}), \tag{5.114}$$

such that $\pi_{i_j,j} > \pi_i, j(i \neq i_j)$. The hard-decision received word R can be written as:

$$R = (r_0, r_1, \ldots, r_{n-1}) = (\rho_{i_0}, \rho_{i_1}, \ldots, \rho_{i_{n-1}}). \tag{5.115}$$

For hard-decision list decoding, only those entries in the multiplicity matrix from (5.107) that correspond to the reliability value $\pi_{i_j,j}$ will be assigned a multiplicity, as $m_{i_j,j} = m$, and therefore the score for hard-decision can also be written with respect to multiplicity matrix $\boldsymbol{M}$ as:

$$S_m(\overline{c}) = S_M(\overline{c}) = \sum_{j=0}^{n-1} \{m_{i_j,j} | \rho_{i_j} = c_j\}. \tag{5.116}$$

Comparing (5.112) and (5.116), the soft-decision list decoder gains its improvements by increasing its code word score. This is done by increasing the total number of interpolated units (5.111) so that the possibility of covering more interpolated units, including the corresponding code word symbols, is also increased.

If the $(1, w_z)$-weighted degree of interpolated polynomial Q_M is δ^*, based on (5.106), Q_M has at most $N_{1,w_z}(\delta^*)$ nonzero coefficients. The interpolation procedure generates a system of C_M linear equations of type (5.108). The system will be solvable if [5]:

$$N_{1,w_z}(\delta^*) > C_M. \tag{5.117}$$

Based on (5.106), in order to guarantee the solution, the $(1, w_z)$-weighted degree δ^* of the interpolated polynomial Q_M should be large enough that:

$$\deg_{1,w_z}(Q_M(x, y, z)) = d^* = \Delta_{1,w_z}(C_M). \tag{5.118}$$

Therefore, based on Corollary 5.3, given the soft-decision code word score (5.116) and the $(1, w_z)$-weighted degree of the interpolated polynomial Q_M (5.118), the message polynomial f can be found if [23]:

$$S_M(\overline{c}) > \Delta_{1,w_z}(C_M). \tag{5.119}$$

The factorization output list contains the z-roots of polynomial Q_M. Therefore, the maximal length of output list l_M should be equal to polynomial Q_M's z-degree ($\deg_z Q_M$), as:

$$l_M = \deg_z(Q_M(x, y, z)) = \left\lfloor \frac{\deg_{1,w_z}(Q_M(x, y, z))}{w_z} \right\rfloor = \left\lfloor \frac{\Delta_{1,w_z}(C_M)}{w_z} \right\rfloor. \quad (5.120)$$

When converting matrix $\mathbf{\Pi}$ to matrix $\boldsymbol{M}$, based on a designed length of output list l, Algorithm 5.1 will stop once l_M is greater than l. $\Delta_{1,w_z}(C_M)$ can be determined by finding the monomial with (1, w_z)-lexicographic order C_M, as:

$$\Delta_{1,w_z}(C_M) = \deg_{1,w_z}(\phi_a z^b | \mathrm{ord}(\phi_a z^b) = C_M). \quad (5.121)$$

Therefore, in order to assess the soft-decision list decoding algorithm's performance with a designed length of output list l, a large enough value must be set for s when initializing Algorithm 5.3. In the algorithm, after step 5, we can determine the cost C_M (5.110) of the updated matrix $\boldsymbol{M}$ and apply (5.121) to determine $\Delta_{1,w_z}(C_M)$. The maximal length of output list l_M can then be determined by (5.120). Stop Algorithm 5.3 once l_M is greater than l and output the updated matrix $\boldsymbol{M}$.

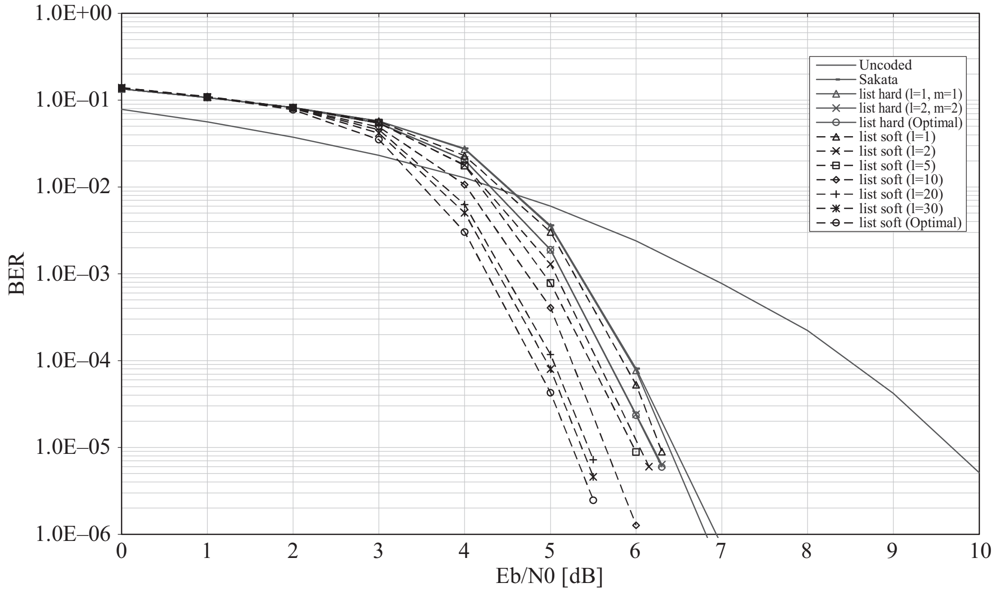
Algorithm 5.7 is based on Algorithm 5.3 and includes this stopping rule. This algorithm is used to obtain the simulation results shown later.

Algorithm 5.7: Convert reliability matrix $\mathbf{\Pi}$ to multiplicity matrix $\boldsymbol{M}$ [14, 23]
Input: Reliability matrix $\mathbf{\Pi}$, a high enough desired value of the sum of multiplicities in matrix $\boldsymbol{M}$:
$s = \sum_{i=0}^{q-1} \sum_{j=0}^{n-1} m_{i,j}$, and designed output length l.
Initialization: Set $\mathbf{\Pi}^* = \mathbf{\Pi}$ and $q \times n$ all-zero multiplicity matrix $\boldsymbol{M}$.

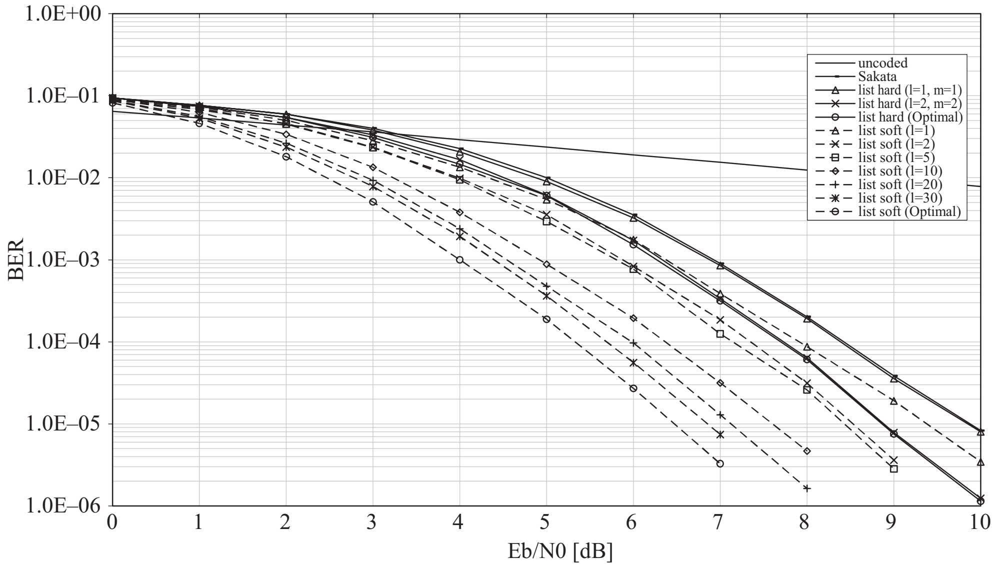
```
1. While (s > 0 or $\lfloor l_M \rfloor < l$)
   {
2. Find the maximal entry $\pi^*_{i,j}$ in $\mathbf{\Pi}^*$ with position $(i, j)$.
3. Update $\pi^*_{i,j}$ in $\mathbf{\Pi}^*$ as $\pi^*_{i,j} = \frac{\pi_{i,j}}{m_{i,j}+2}$.
4. Update $m_{i,j}$ in $\boldsymbol{M}$ as $m_{i,j} = m_{i,j} + 1$.
5. $s = s - 1$.
6. For the updated $\boldsymbol{M}$, calculate its interpolation cost $C_M$ by (5.110).
7. Determine $\Delta_{1,w_z}(C_M)$ by (5.121).
8. Calculate $l_M$ by (5.120).
   }
```

Again, this algorithm gives priority to those interpolated points which correspond to a higher reliability value $\pi_{i,j}$, to be assigned with a higher multiplicity value $m_{i,j}$. For example, if $\pi_{i_1 j_1} < \pi_{i_2 j_2}$ then $m_{i_1 j_1} \leq m_{i_2 j_2}$.

The interpolation and factorization methods given in Algorithms 5.5 and 5.6 are unchanged for soft-decision list decoding of AG codes. Again, as the total number of

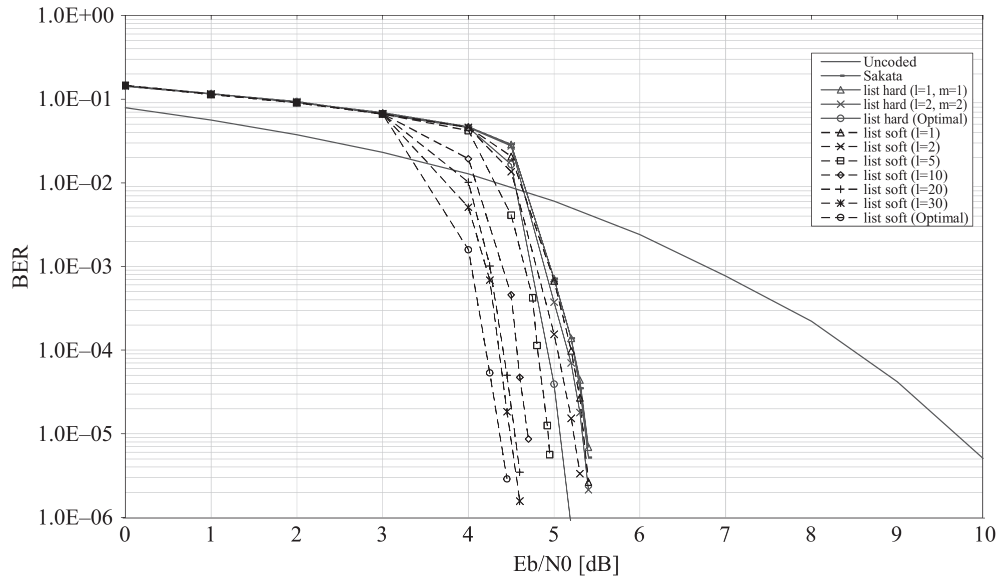


(a) over AWGN channel

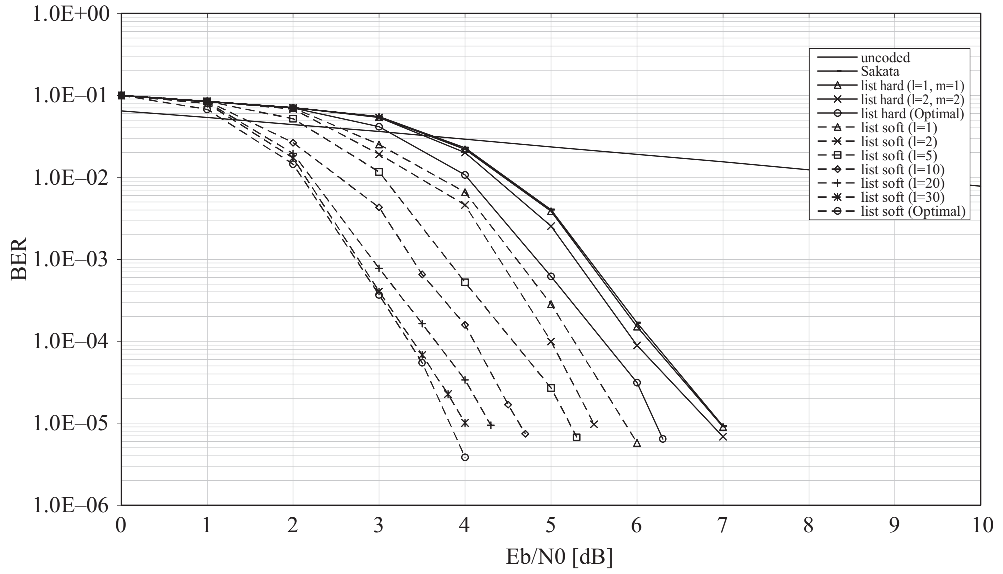


(b) over Rayleigh fading channel

Figure 5.9 Soft-decision list decoding of Hermitian code (64, 39, 20).



(a) over AWGN channel



(b) over Rayleigh fading channel

Figure 5.10 Soft-decision list decoding of Hermitian code (512, 289, 196).

iterations C_M has been determined before running the interpolation, those polynomials with leading order greater than C_M can be eliminated from the group during the iterations [23].

5.8.2 Simulation Results

The performance of hard- and soft-decision list decoding for Hermitian codes is evaluated by simulation results on the AWGN channel and a frequency nonselective Rayleigh fading channel with Doppler frequency 126.67 Hz. The fading profile is generated using Jakes' method [16] and the fading coefficients have a mean value of 1.55 and variance 0.60. During simulation, quasi-static fading is assumed in which the fading amplitude changes for each code word block. For combating the fading effect, 64×64 and 100×512 block interleavers are employed for codes defined in GF(16) and GF(64) respectively.

In Figure 5.9a the soft-decision list decoding performance of the (64, 39, 20) Hermitian code over GF(16) is shown on the AWGN channel for different list lengths. This is compared with the hard-decision list decoding performance with multiplicity values $m = 1$ and 2, and the performance of the Sakata algorithm is included too. We can see that there are small coding gains for soft-decision list decoding over hard-decision list decoding up to approximately 1.3 dB at a BER of 10^{-4}. In Figure 5.9b, more significant coding gains are achieved on the fading channel, with a coding gain greater than 3 dB over hard-decision list decoding.

In Figure 5.10a the soft-decision list decoding performance of the (512, 289, 196) Hermitian code defined over GF(64) is shown on the AWGN channel for different output list lengths. Again, small coding gains are achieved over hard-decision decoding. In Figure 5.10b the soft-decision list decoding performance on the fading channel is shown, with more significant coding gains over hard-decision decoding.

5.9 Conclusions

In this chapter we have introduced the concept of list decoding for Reed–Solomon and AG codes. We can see that list decoding allows more errors to be corrected than with conventional algebraic decoding algorithms, but at a cost of higher complexity. However, a method to reduce this complexity without degrading the performance is given for both Reed–Solomon and AG codes. The Kotter–Vardy algorithm for the soft-decision list decoding of Reed–Solomon codes was presented; this is almost identical to the Guruswami–Sudan algorithm but with an extra process to convert the reliability of the received symbols into a multiplicity matrix. The Kotter–Vardy algorithm was also extended to soft decode AG codes. One interesting observation from the soft-decision list decoding algorithm was that it could achieve the same performance as the hard-decision list decoder but at a *lower* complexity. More significant coding gains can be achieved on a slow fading channel, particularly for AG codes, suggesting that soft-decision list decoding is very suitable for channels where bursts of errors are common.

References

[1] Elias, P. (1957) *List Decoding for Noisy Channels*, Res. Lab. Electron, MIT, Cambridge, MA.

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

[3] Wozencraft, J.M. (1958) *List Decoding*, Res. Lab. Electron, MIT, Cambridge, MA.

[4] McEliece, R.J. (2003) The Guruswami–Sudan Decoding Algorithm for Reed–Solomon Codes, California Institute. Tech, Pasadena, California, IPN Progress Rep, pp. 42–153.

[5] Koetter, R. and Vardy, A. (2003) Algebraic soft-decision decoding of Reed–Solomon codes. *IEEE Trans. Inform. Theory*, **49**, 2809–25.

[6] Guruswami, V. and Sudan, M. (1999) Improved decoding of Reed–Solomon and algebraic–geometric codes. *IEEE Trans. Inform. Theory*, **45**, 1757–67.

[7] Hasse, H. (1936) Theorie der hoheren differentiale in einem algebraishen funcktionenkorper mit vollkommenem konstantenkorper nei beliebeger charakteristic. *J. Reine. Aug. Math*, **175**, 50–4.

[8] Koetter, R. (1996) On algebraic decoding of algebraic–geometric and cyclic codes. Linkoping, Sweden: University Linkoping.

[9] Nielsen, R.R. (2001) List decoding of linear block codes. Lyngby, Denmark: Tech. Univ. Denmark.

[10] Moon, T.K. (2005) *Error Correction Coding – Mathematical and Algorithms*, Wiley Interscience.

[11] Høholdt, T. and Nielsen, R.R. (1999) Decoding Hermitian codes with Sudan's algorithm, in *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes* (eds H.I.N. Fossorier, S. Lin, and A. Pole), (Lecture Notes in Computer Science), Vol. **1719**, Springer-Verlag, Berlin, Germany, pp. 260–70.

[12] Feng, G.-L. and Tzeng, K.K. (1991) A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with application to decoding cyclic codes. *IEEE Trans. Inform. Theory*, **37**, 1274–87.

[13] Chen, L., Carrasco, R.A. and Chester, E.G. (2007) Performance of Reed–Solomon codes using the Guruswami–Sudan algorithm with improved interpolation efficiency. *IET Commun*, **1**, 241–50.

[14] Chen, L. (2007) Design of an efficient list decoding system for Reed–Solomon and algebraic–geometric codes, PhD Thesis *School of electrical, electronic and computer engineering*. Newcastle-upon-Tyne: Newcastle University.

[15] Roth, R. and Ruckenstein, G. (2000) Efficient decoding of Reed–Solomon codes beyond half the minimum distance. *IEEE Trans. Inform. Theory*, **46**, 246–57.

[16] Proakis, J.G. (2000) *Digital Communications*, 4th edn, McGraw-Hill International.

[17] Chen, L., Carrasco, R.A. and Johnston, M. (2006) List decoding performance of algebraic geometric codes. *IET Electronic Letters*, **42**, 986–7.

[18] Chen, L., Carrasco, R.A. and Johnston, M. (XXXX) Reduced complexity interpolation for list decoding Hermitian codes. *IEEE Trans. Wireless Commun*, Accepted for publication.

[19] Chen, L. and Carrasco, R.A. (2007) Efficient list decoder for algebraic–geometric codes. Presented at 9th International Symposium on Communication Theory and Application (ISCTA'07), Ambleside, Lake district, UK.

[20] Wu, X.-W. and Siegel, P. (2001) Efficient root-finding algorithm with application to list decoding of algebraic–geometric codes. *IEEE Trans. Inform. Theory*, **47**, 2579–87.

[21] Chen, L., Carrasco, R.A., Johnston, M. and Chester, E.G. (2007) Efficient factorisation algorithm for list decoding algebraic–geometric and Reed–Solomon codes. Presented at ICC 2007, Glasgow, UK.

[22] Wu, X.-W. (2002) An algorithm for finding the roots of the polynomials over order domains. Presented at ISIT 2002, Lausanne, Switzerland.

[23] Chen, L., Carrasco, R.A. and Johnston, M. (XXXX) Soft-decision list decoding of Hermitian codes. *IEEE Trans. Commun*, Submitted for publication.

6

Non-Binary Low-Density Parity Check Codes

6.1 Introduction

The previous chapters have described non-binary block codes that have algebraic decoders, but now a new type of block code is introduced that is decoded using a graph. This is the well-known low-density parity check (LDPC) code, which was first presented by Gallager in 1962 [1] but only became popular when Mackay rediscovered it in 1995 [2]. It has been shown that with iterative soft-decision decoding LDPC codes can perform as well as or even better than turbo codes [3]. It is claimed that LDPC codes will be chosen in future standards, such as 4G, later versions of WiMax and magnetic storage devices.

In this chapter, binary LDPC codes are first introduced with a discussion on random and structured construction methods. The Belief Propagation algorithm is then presented in detail to decode LDPC codes. We then expand binary LDPC codes to non-binary LDPC codes defined over a finite field GF(q) and again discuss different construction methods. Finally the Belief Propagation algorithm is extended to non-binary symbols, which increases its complexity, but a method to reduce complexity is introduced using fast Fourier transforms (FFT), allowing us to decode non-binary LDPC codes defined over large finite fields.

6.2 Construction of Binary LDPC Codes – Random and Structured Methods

A low-density parity check (LDPC) code is characterized by its sparse parity check matrix, that is a matrix containing mostly zero elements and few nonzero elements. Three important parameters are its code word length, n, its dimension, k, and its number of parity bits, $m = n - k$. The number of nonzero elements in a row of the parity check matrix is called the *row weight*, denoted by ρ. The number of nonzero elements in a column of the parity check matrix is called the *column weight*, denoted by γ. There are two general classes of LDPC code:

- If the row weights and column weights are constant for each row and column in the parity check matrix then the LDPC code is said to be *regular*.
- If the row and column weights vary for each row and column in the parity check matrix then the LDPC code is said to be *irregular*.

In general, irregular LDPC codes have a better performance than regular LDPC codes. An example of a parity check matrix **H** for a small binary LDPC code is given in (6.1):

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{6.1}$$

We can see that **H** has a constant row weight $\rho = 3$ and constant column weight $\gamma = 2$, implying that this is a regular LDPC code. Originally, LDPC codes were constructed by first choosing the row and column weights and then randomly determining where the nonzero elements were located.

6.2.1 Tanner Graph Representation

A parity check matrix can be expressed in the form of a bipartite graph known as a Tanner graph. Each row in the parity check matrix represents a parity check equation z_i, $1 \leq \mathrm{i} \leq m$, and each column represents a coded bit c_j, $1 \leq j \leq n$. For example, the first row of **H** in (6.1) is the parity check equation $z_1 = c_1 \oplus c_2 \oplus c_4$. The Tanner graph connects the coded bits to each parity check equation. For each row of the parity check matrix, if the coded bit is a 1 then there is a connection between that coded bit and the corresponding parity check equation. The Tanner graph for **H** in (6.1) is given in Figure 6.1.

The Tanner graph reveals the presence of *cycles*, that is a path starting and ending at the same coded bit. In Figure 6.1 it can be seen that the smallest cycle has a length of 6. One such cycle is $c_1 \rightarrow z_1 \rightarrow c_4 \rightarrow z_4 \rightarrow c_6 \rightarrow z_3 \rightarrow c_1$. The length of the smallest cycle in a Tanner graph is known as its *girth*. It is desirable to avoid LDPC codes with Tanner graphs with a girth of 4 as this degrades the performance, particularly for short LDPC codes. From the Tanner graph, we introduce two sets:

- M_n is the set of indices of the parity checks connected to coded bit c_n.
- N_m is the set of indices the coded bits that are connected to parity check z_m.

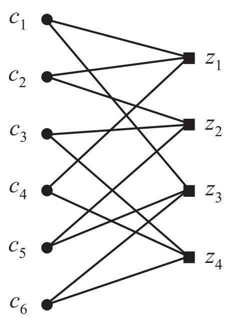


Figure 6.1 Tanner graph for the parity check matrix of (6.1).

So from the Tanner graph of Figure 6.1, the set $M_1 = \{1, 3\}$, since parity checks z_1 and z_3 are connected to coded bit c_1. The set $N_1 = \{1, 2, 4\}$, since coded bits c_1, c_2 and c_4 are connected to parity check z_1. We also introduce the notation M_n/m, which is the set M_n excluding the parity check z_m that is connected to the coded bit c_n. Similarly, N_m/n is the set N_m excluding the coded bit c_n that is connected to the parity check z_m. Therefore, $M_1/1 = \{3\}$ and $N_1/1 = \{2, 4\}$. This notation will be useful when describing the Belief Propagation algorithm for decoding LDPC codes.

6.2.2 Structured Construction Methods

There are many different methods for constructing a binary LDPC code [5–6], but in this chapter we will discuss just one. This method is known as the Balanced Incomplete Block Design (BIBD), which can be used to construct high-rate LDPC codes [4]. Let $X = \{x_1, x_2, \ldots, x_v\}$ be a set of v objects. A BIBD of X is a collection of n γ-subsets of X, denoted by $B_1, B_2, \ldots, B_n$, called blocks, such that the following conditions are satisfied:

1. Each object appears in exactly ρ of the n blocks.
2. Every two objects appear together in exactly λ of the n blocks.
3. The number γ of objects in each block is small compared to the total number of objects in X.

Instead of a list of the blocks, a BIBD can be efficiently described by a $v \times n$ matrix **Q** over GF(2), as follows:

1. The rows of **Q** correspond to the v objects in X.
2. The columns of **Q** correspond to the n blocks of the design.

3. Each element $Q_{i,j}$ in $\mathbf{Q}$ is a 1 if and only if the ith object x_i is contained in the jth block B_j of the design; otherwise it is 0. This matrix is called the *incidence matrix* of the design.

It follows from the structural properties of a BIBD that the incidence matrix $\mathbf{Q}$ has constant row weight ρ and constant column weight γ and any two rows of $\mathbf{Q}$ have exactly λ '1-components' in common.

Example 6.1: Construction of an incidence matrix using BIBD: Let $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ be a set of seven objects. The blocks $\{x_1, x_2, x_4\}$, $\{x_2, x_3, x_5\}$, $\{x_3, x_4, x_6\}$, $\{x_4, x_5, x_7\}$, $\{x_5, x_6, x_1\}$, $\{x_6, x_7, x_2\}$,$\{x_7, x_1, x_3\}$ form a BIBD for the set X. Every block consists of three objects, each object appears in three blocks, and every two objects appear together in exactly one block, that is $\lambda = 1$. The incidence matrix $\mathbf{Q}$ is given below:

$$\mathbf{Q} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Note that each row of $\mathbf{Q}$ is a cyclic right-shift of the row above it and the first row is the cyclic shift of the last row. We also note that each column is a downward cyclic shift of the column on its left and the first column is the downward cyclic shift of the last column. Therefore $\mathbf{Q}$ is a 7×7 square *circulant matrix*. Therefore, the null space of $\mathbf{Q}$ gives a (γ, ρ)-regular LDPC code, called a *BIBD-LDPC* code, of length n. The Tanner graph of a BIBD-LDPC code is free of cycles of length 4 and hence its girth is at least 6. In fact, the second property of a BIBD with $\lambda = 1$ ensures that the girth of the code's Tanner graph is exactly 6. If $\lambda = 2$ then the incidence matrix will have a Tanner graph full of short cycles of length 4, which is undesirable.

6.2.3 LDPC Codes from Class 1 Bose BIBD

There are four classes of Bose BIBD, each with different types producing many different binary LDPC codes. However, in this book we only deal with LDPC constructed from Class 1 Bose BIBDs of type 1 [4]. Choose a finite field GF($2t + 1$), where t is selected so that GF($2t + 1$) is prime, with a primate element x that satisfies $x^{4t} - 1 = x^c$, where c is an odd integer less than $2t + 1$. The resulting LDPC code parameters will have the following values: $n = t(12t + 1)$, $v = 12t + 1$, $\gamma = 4$, $\rho = 4t$, $\lambda = 1$. The base blocks of this BIBD are:

$$\{0, x^{2i}, x^{2i+4t}, x^{2i+8t}\}, \tag{6.2}$$

for $0 \leq i < t$. From each base block B_i, we can form $12t + 1$ blocks by adding each element of the field GF($12t + 1$) in turn to the elements in B_i. This results in $t(12t + 1)$ blocks. The incidence matrix **Q** of this BIBD is a $(12t + 1) \times t(12t + 1)$ matrix. It can be written in cyclic form, which consists of t $(12t + 1) \times (12t + 1)$ circulant submatrices in a row, as shown below:

$$\mathbf{Q} = (\mathbf{Q}_1\ \mathbf{Q}_2 \cdots \mathbf{Q}_t), \tag{6.3}$$

where the ith circulant $\mathbf{Q}_i$ is the incidence matrix formed by adding each element in GF($12t + 1$) in turn to the elements of the ith base block B_i. A circulant is a square matrix which has the following structure:

1. Each row is a right-cyclic shift of the row above it and the first row is the right-cyclic shift of the last row.
2. Each column is a downward cyclic shift of the column on its left and the first column is the downward cyclic shift of the last column. The set of columns is the same as the set of rows, except that each row reads from right to left.

Both the row and the column weight of each circulant $\mathbf{Q}_i$ are 4. Therefore, the row and column weights of **Q** are $4t$ and 4 respectively. The rank of **Q** (or a circulant $\mathbf{Q}_i$) is observed to be $12t$.

Example 6.2: Construction of an LDPC code using a class 1 Bose BIBD: Let $t = 1$. Then $12t + 1 = 13$. The element 6 of GF(13) = $\{0, 1, 2, \ldots, 12\}$ is a primitive element so with $c = 3$ we can easily check that $x^4 - 1 = 64 - 1 = 9 - 1 = 8 = 6^3$, which is less than 13. Therefore, there is a BIBD with parameters, $v = 13$, $n = 13$, $\gamma = 4$, $\rho = 4$ and $\lambda = 1$. The base block for this design is $\{0, 6^0, 6^4, 6^8\} = \{0, 1, 9, 3\}$. By adding each element in GF(13) to the base block we get the following 13 blocks:

$B_0 = \{0, 1, 9, 3\}$, $B_1 = \{1, 2, 10, 4\}$, $B_2 = \{2, 3, 11, 5\}$, $B_3 = \{3, 4, 12, 6\}$, $B_4 = \{4, 5, 0, 7\}$,
$B_5 = \{5, 6, 1, 8\}$, $B_6 = \{6, 7, 2, 9\}$, $B_7 = \{7, 9, 3, 10\}$, $B_8 = \{8, 9, 4, 11\}$, $B_9 = \{9, 10, 5, 12\}$,
$B_{10} = \{10, 11, 6, 0\}$, $B_{11} = \{11, 12, 7, 1\}$, $B_{12} = \{12, 0, 8, 2\}$.

Hence, the circulant matrix **Q** is:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

This example only produces a single circulant matrix, but the next value of t that ensures the finite field is prime is $t = 6$, giving GF(73). This will result in an LDPC code with parameters $v = 73$, $n = 438$, $\gamma = 4$, $\rho = 292$ and $\lambda = 1$. In this case, there will be six base blocks each containing four elements, and six circulant matrices can be formed by adding every element in GF(73) to each of the base blocks. Each circulant matrix will have dimensions 73×73, with $\gamma = 4$, $\rho = 4$. The six circulant matrices are then concatenated to form the final parity check matrix, H, with dimensions 73×438. The null space of this matrix will be the (438, 365) LDPC code, assuming each row in H is independent.

6.2.4 Constructing the Generator Matrix of a Binary LDPC Code

The encoding of a binary message to form an LDPC code word is achieved by multiplying the binary message vector by a systematic generator matrix, **G**. The systematic generator matrix is obtained by first performing Gauss–Jordan elimination on the parity check matrix so that it is of the form $\mathbf{H} = [\mathbf{I_m}|\mathbf{P}]$, where $\mathbf{I_m}$ is the $m \times m$ identity matrix and **P** is a parity matrix. The generator matrix is then $\mathbf{G} = [\mathbf{P^T}|\mathbf{I_k}]$, where $\mathbf{P^T}$ is the matrix transpose of **P**, and $\mathbf{I_k}$ is the $k \times k$ identity matrix. Gauss–Jordan elimination is related to Gaussian elimination in that it eliminates all elements below and also above each pivot element. Gaussian elimination results in a matrix in row echelon form, whereas Gauss–Jordan elimination results in a matrix in reduced row echelon form. It can therefore be seen as a two-stage process, where Gaussian elimination is first performed to transform the matrix into row echelon form, followed by elimination of elements above the pivot elements.

The elimination procedure on a matrix containing finite field elements is performed in the same way as if the matrix contained real numbers. The only difference is that elements are eliminated with modulo-2 addition instead of subtraction.

It is not unusual for a parity check matrix to have dependent rows. When performing Gaussian elimination on the matrix, all dependent rows will appear as rows containing all zeroes located at the bottom of the new matrix. This is the case if we apply Gaussian elimination to the parity check matrix of (6.1):

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \rightarrow \mathbf{H}' \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

We can see that row 4 in **H** is the modulo-2 sum of row 1, row 2 and row 3 and hence it is a dependent row. Therefore, performing Gaussian elimination results in an all-zero row in the new matrix, **H**′. However, a generator matrix can still be obtained by removing the all-zero rows and then eliminating elements above the pivot elements in the resulting submatrix:

$$\mathbf{H}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \rightarrow \mathbf{H}'' = \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right].$$

The systematic parity check matrix **H**″ is simply obtained by adding row 3 to row 2 in **H**′. Finally, the generator matrix is:

$$\mathbf{G} = \left[\begin{array}{ccc:ccc} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right].$$

We can see that dependent rows in the parity check matrix will increase the code rate of the LDPC code. If all the rows were independent then the null space of the parity check matrix of (6.1) would be the (6, 2) LDPC code with code rate of 1/3. However, the dependent row means the null space is the (6, 3) LDPC code with code rate of 1/2.

For larger parity check matrices, the swapping of columns is usually required during Gaussian elimination to ensure that the pivot elements are on the main diagonal of the matrix. However, this will obviously change the parity check equations and the resulting systematic generator matrix will not be the null space of the original parity check matrix. In this case, any column that is swapped must also be swapped on the original parity check matrix, but this will not alter its column and row weights.

6.3 Decoding of Binary LDPC Codes Using the Belief Propagation Algorithm

The decoding of LDPC codes using the sum product algorithm involves finding a code word of length n where each coded bit c_n maximizes the probability of c_n conditioned on the parity check equations associated with c_n being satisfied [7].

$$P(c_n|\{z_m = 0,\ m \in M_n\}). \tag{6.4}$$

The sum product algorithm involves calculating two probabilities. The first is the probability q_{mn}, which is the probability of the nth code bit conditioned on its associated parity checks being satisfied with the exception of the mth parity check:

$$q_{mn}(x) = P(c_n = x \left| \{z_m = 0, m' \in M_n/m\} \right.. \tag{6.5}$$

The second probability is denoted r_{mn}, which is the probability that the mth parity check is satisfied conditioned on all the possible values of the coded bits $\mathbf{c}$:

$$r_{mn}(x) = P(z_m = 0|\mathbf{c}). \tag{6.6}$$

Figure 6.2 shows how these probabilities are exchanged in the Tanner graph of Figure 6.1 for the connections between c_1 and z_1, and c_6 and z_4.

6.3.1 Initialization of the Belief Propagation Algorithm

The probabilities $q_{mn}(x)$ are initialized to the probability $f_n^{(x)}$ that the nth received bit is x, that is $P(c_n = x)$. For the additive white Gaussian noise (AWGN) channel the probability $g_n^{(x)}$ is [7]:

$$g_n^{(0)} \propto \frac{1}{\sigma\sqrt{2\pi}} e^{-(r_n-1)^2/2\sigma^2} \quad \text{and} \quad g_n^{(1)} \propto \frac{1}{\sigma\sqrt{2\pi}} e^{-(r_n+1)^2/2\sigma^2}. \tag{6.7}$$

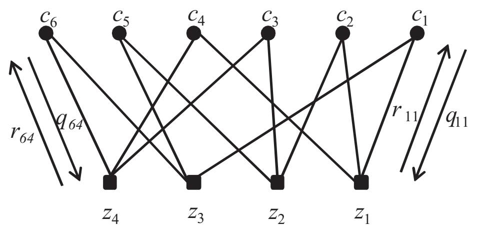
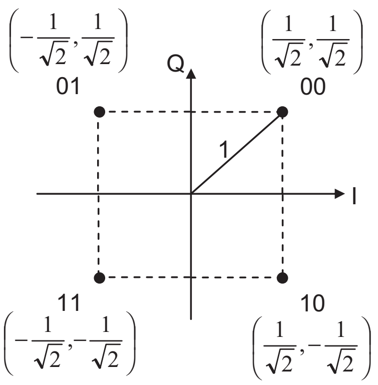


Figure 6.2 Tanner graph showing the exchange of information between the coded bits and their parity checks.



$\left(-\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$
01
Q
$\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$
00
1
I
11
$\left(-\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)$
10
$\left(\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right)$

Figure 6.3 QPSK constellation.

In general, when the modulation scheme has an in-phase and quadrature component the probabilities are determined by calculating the Euclidean distances for the in-phase component from each constellation point. The coordinates of each constellation point in the QPSK constellation are shown in Figure 6.3.

Therefore, the four probabilities $g_n^{(00)}$, $g_n^{(01)}$, $g_n^{(11)}$ and $g_n^{(10)}$ for the QPSK constellation are:

$$\begin{aligned}
g_n^{(00)} &\propto \frac{1}{\sigma\sqrt{2\pi}} e^{-\left[\left(r_n^{(I)}-\frac{1}{\sqrt{2}}\right)^2+\left(r_n^{(Q)}-\frac{1}{\sqrt{2}}\right)^2\right]/2\sigma^2}, \\
g_n^{(01)} &\propto \frac{1}{\sigma\sqrt{2\pi}} e^{-\left[\left(r_n^{(I)}-\frac{1}{\sqrt{2}}\right)^2+\left(r_n^{(Q)}+\frac{1}{\sqrt{2}}\right)^2\right]/2\sigma^2}, \\
g_n^{(11)} &\propto \frac{1}{\sigma\sqrt{2\pi}} e^{-\left[\left(r_n^{(I)}+\frac{1}{\sqrt{2}}\right)^2+\left(r_n^{(Q)}+\frac{1}{\sqrt{2}}\right)^2\right]/2\sigma^2}, \\
g_n^{(10)} &\propto \frac{1}{\sigma\sqrt{2\pi}} e^{-\left[\left(r_n^{(I)}+\frac{1}{\sqrt{2}}\right)^2+\left(r_n^{(Q)}-\frac{1}{\sqrt{2}}\right)^2\right]/2\sigma^2}
\end{aligned} \tag{6.8}$$

where $r_n^{(I)}$ and $r_n^{(Q)}$ are the in-phase and quadrature components of the received symbol. The likelihoods of the received bits are then placed in a vector **f**:

$$\mathbf{f} = \left[\begin{array}{c:c:c:c:c} g_1^{(0)} & g_2^{(0)} & \cdots & g_{n-1}^{(0)} & g_n^{(0)} \\ g_1^{(1)} & g_2^{(1)} & \cdots & g_{n-1}^{(1)} & g_n^{(1)} \end{array}\right]. \tag{6.9}$$

To initialize the sum–product algorithm, the likelihoods in **f** are used to initialize a matrix **Q** defined as:

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c} g_1^{(0)} & g_2^{(0)} & g_3^{(0)} & \cdots & g_{n-1}^{(0)} & g_n^{(0)} \\ g_1^{(1)} & g_2^{(1)} & g_3^{(1)} & \cdots & g_{n-1}^{(1)} & g_n^{(1)} \\ \hdashline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hdashline g_1^{(0)} & g_2^{(0)} & g_3^{(0)} & \cdots & g_{n-1}^{(0)} & g_n^{(0)} \\ g_1^{(1)} & g_2^{(1)} & g_3^{(1)} & \cdots & g_{n-1}^{(1)} & g_n^{(1)} \end{array}\right]. \tag{6.10}$$

The matrix **Q** has partitions containing the likelihoods that the *i*th received bit is a 0, $g_i^{(0)}$ or a 1, $g_i^{(1)}$.

6.3.2 The Horizontal Step

The horizontal step of the sum–product algorithm determines the probability r_{mn}, defined as [7]:

$$r_{mn}(x) = \sum_{\mathbf{c}:c_n=x} P(z_m = 0|\mathbf{c})P(\mathbf{c}|c_n = x), \tag{6.11}$$

where, given that the *n*th coded bit $c_n = x$, **c** is a binary vector of the remaining coded bits of length $\rho - 1$, containing every possible combination of binary sequences. There are therefore $2^{\rho-1}$ binary sequences that (6.9) is summed over. However, not all these sequences will satisfy $z_m = 0$ so we only consider those sequences that do satisfy the parity check. For example, if $x = 0$ then only those binary sequences with an even number of 1s are able to satisfy $z_m = 0$. Therefore, the probability of a sequence containing an odd number of 1s satisfying $z_m = 0$ is zero. Therefore, $r_{mn}(x)$ is the sum of all the probabilities of the binary sequences of length $\rho - 1$ that when summed together equal x:

$$r_{mn}(x) = \sum_{\mathbf{c}:c_n=x} P(z_m = 0\,|\mathbf{c}) \prod_{n' \in N_m \backslash n} q_{mn'}(x), \tag{6.12}$$

where $P(z_m = 0 \mid \mathbf{c})$ is either zero or one. The probabilities $r_{mn}(x)$ are placed in a matrix $\mathbf{R}$:

$$\mathbf{R} = \left[\begin{array}{c:c:c:c:c:c} r_{11}(0) & r_{12}(0) & r_{13}(0) & \cdots & r_{1(n-1)}(0) & r_{1n}(0) \\ r_{11}(1) & r_{12}(1) & r_{13}(1) & \cdots & r_{1(n-1)}(1) & r_{1n}(1) \\ \hdashline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hdashline r_{m1}(0) & r_{m2}(0) & r_{m3}(0) & \cdots & r_{m(n-1)}(0) & r_{mn}(0) \\ r_{m1}(1) & r_{m2}(1) & r_{m3}(1) & \cdots & r_{m(n-1)}(1) & r_{mn}(1) \end{array}\right], \tag{6.13}$$

where each element in the parity check matrix has a corresponding 1×2 column vector containing the probabilities $r_{mn}(0)$ and $r_{mn}(1)$ (or a $1 \times q$ column vector for an LDPC code defined over GF(q)).

6.3.3 The Vertical Step

The vertical step updates the probabilities q_{mn} and is much simpler than the horizontal step. From (6.5), q_{mn} can be written using Baye's Rule as [7]:

$$\begin{aligned} q_{mn}(x) &= P(c_n = x \left| \left\{ z_m = 0, m' \in M_n/m \right\} \right. \\ &= \frac{P(c_n = x)P\left(\left\{z_m = 0, m' \in M_n/m\right\} |c_n = x\right)}{P\left(\{z_m = 0, m' \in M_n/m\}\right)}. \end{aligned} \tag{6.14}$$

Using (6.6) and letting $f_n^x = P(c_n = x)$, (6.10) can be written as [7]:

$$q_{mn}(x) = \beta_{mn} f_n^x \prod_{m' \in M_n/m} r_{m'n}(x), \tag{6.15}$$

where β_{mn} is a normalizing constant such that $\sum q_{mn}(x) = 1$, that is:

$$\beta_{mn} = \frac{1}{\sum\limits_x f_n^x \prod\limits_{m' \in M_n \backslash m} r_{m'n}(x)}. \tag{6.16}$$

The $q_{mn}(x)$ are placed in the matrix $\mathbf{Q}$ as shown in (6.17):

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c} q_{11}(0) & q_{12}(0) & q_{13}(0) & \cdots & q_{1(n-1)}(0) & q_{1n}(0) \\ q_{11}(1) & q_{12}(1) & q_{13}(1) & \cdots & q_{1(n-1)}(1) & q_{1n}(1) \\ \hdashline \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \hdashline q_{m1}(0) & q_{m2}(0) & q_{m3}(0) & \cdots & q_{m(n-1)}(0) & q_{mn}(0) \\ q_{m1}(1) & q_{m2}(1) & q_{m3}(1) & \cdots & q_{m(n-1)}(1) & q_{mn}(1) \end{array}\right], \tag{6.17}$$

where each element in the parity check matrix has a corresponding 1×2 column vector containing the probabilities $q_{mn}(0)$ and $q_{mn}(1)$ (or a $1 \times q$ column vector for an LDPC code defined over GF(q)).

In this step, the pseudo posterior probabilities $q_n(x)$ are also determined by [7]:

$$q_n(x) = \beta_n f_n^x \prod_{m \in M_n} r_{mn}(x), \tag{6.18}$$

where again β_n is a normalizing constant such that $\sum q_n(x) = 1$. The pseudo posterior probabilities are place in a matrix $\mathbf{Q}'$:

$$\mathbf{Q}' = \begin{bmatrix} q_1(0) & q_2(0) & q_3(0) & \cdots & q_{n-1}(0) & q_n(0) \\ q_1(1) & q_2(1) & q_3(1) & \cdots & q_{n-1}(1) & q_n(1) \end{bmatrix}. \tag{6.19}$$

From these pseudo posterior probabilities estimates of the transmitted code word can be found by:

$$\hat{c}_n = \arg\max_x \beta_n f_n^x \prod_{m \in N_m} r_{mn}(x). \tag{6.20}$$

Example 6.3: Decoding a binary LDPC code with the Belief Propagation algorithm: In this example, we use the regular (10, 5) binary LDPC code with row weight $\rho = 6$ and column weight $\gamma = 3$. The parity check matrix $\mathbf{H}$ is given as:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Let us assume that the transmitted code word is $\mathbf{c} = [0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1]$ and the received word is $\mathbf{r} = [0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1]$, with a single error in the fifth position. The associated likelihoods are:

$$\mathbf{f} = \left[\begin{array}{c:c:c:c:c:c:c:c:c:c} 0.78 & 0.84 & 0.81 & 0.52 & 0.45 & 0.13 & 0.82 & 0.21 & 0.75 & 0.24 \\ 0.22 & 0.16 & 0.19 & 0.48 & 0.55 & 0.87 & 0.18 & 0.79 & 0.25 & 0.76 \end{array}\right].$$

We initialize the matrix **Q** with these likelihoods as follows:

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c:c:c:c:c} 0.78 & 0.84 & 0.81 & 0 & 0 & 0.13 & 0.82 & 0 & 0 & 0.24 \\ 0.22 & 0.16 & 0.19 & 0 & 0 & 0.87 & 0.18 & 0 & 0 & 0.76 \\ \hdashline 0.78 & 0 & 0.81 & 0 & 0.45 & 0.13 & 0 & 0.21 & 0.75 & 0 \\ 0.22 & 0 & 0.19 & 0 & 0.55 & 0.87 & 0 & 0.79 & 0.25 & 0 \\ \hdashline 0 & 0 & 0.81 & 0.52 & 0.45 & 0 & 0.82 & 0 & 0.75 & 0.24 \\ 0 & 0 & 0.19 & 0.48 & 0.55 & 0 & 0.18 & 0 & 0.25 & 0.76 \\ \hdashline 0 & 0.84 & 0 & 0.52 & 0.45 & 0.13 & 0 & 0.21 & 0 & 0.24 \\ 0 & 0.16 & 0 & 0.48 & 0.55 & 0.87 & 0 & 0.79 & 0 & 0.76 \\ \hdashline 0.78 & 0.84 & 0 & 0.52 & 0 & 0 & 0.82 & 0.21 & 0.75 & 0 \\ 0.22 & 0.16 & 0 & 0.48 & 0 & 0 & 0.18 & 0.79 & 0.25 & 0 \end{array}\right].$$

The matrix **Q** containing all the probabilities $q_{mn}(x)$ is then used for the horizontal step to determine the matrix **R** containing the probabilities $r_{mn}(x)$. To determine the probability $r_{11}(1)$ we must calculate the probabilities of all the possible binary sequences that satisfy z_1 when $c_1 = 1$. Hence, we must first find all possible binary values of c_2, c_3, c_6, c_7 and c_{10} that satisfy:

$$c_2 + c_3 + c_6 + c_7 + c_{10} = c_1 = 1.$$

Since we are only working with binary values, the above equation can only be satisfied when an odd number of the coded bits are equal to 1. The 16 possible bit sequences and their probabilities are given in Table 6.1.

$$\begin{aligned} r_{11}(1) = {} & \big(q_{12}(1)q_{13}(0)q_{16}(0)q_{17}(0)q_{1,10}(0)\big) + \big(q_{12}(0)q_{13}(1)q_{16}(0)q_{17}(0)q_{1,10}(0)\big) + \\ & + \big(q_{12}(0)q_{13}(0)q_{16}(1)q_{17}(0)q_{1,10}(0)\big) + \big(q_{12}(0)q_{13}(0)q_{16}(0)q_{17}(1)q_{1,10}(0)\big) \\ & + \big(q_{12}(0)q_{13}(0)q_{16}(0)q_{17}(0)q_{1,10}(1)\big) + \big(q_{12}(1)q_{13}(1)q_{16}(1)q_{17}(0)q_{1,10}(0)\big) \\ & + \big(q_{12}(0)q_{13}(1)q_{16}(1)q_{17}(1)q_{1,10}(0)\big) + \big(q_{12}(0)q_{13}(0)q_{16}(1)q_{17}(1)q_{1,10}(1)\big) \\ & + \big(q_{12}(1)q_{13}(1)q_{16}(0)q_{17}(0)q_{1,10}(1)\big) + \big(q_{12}(1)q_{13}(1)q_{16}(0)q_{17}(1)q_{1,10}(0)\big) \\ & + \big(q_{12}(0)q_{13}(1)q_{16}(1)q_{17}(0)q_{1,10}(1)\big) + \big(q_{12}(1)q_{13}(0)q_{16}(1)q_{17}(0)q_{1,10}(1)\big) \\ & + \big(q_{12}(1)q_{13}(0)q_{16}(0)q_{17}(1)q_{1,10}(1)\big) + \big(q_{12}(0)q_{13}(1)q_{16}(0)q_{17}(1)q_{1,10}(1)\big) \\ & + \big(q_{12}(1)q_{13}(0)q_{16}(1)q_{17}(1)q_{1,10}(0)\big) + \big(q_{12}(1)q_{13}(1)q_{16}(1)q_{17}(1)q_{1,10}(1)\big) \\ & = 0.00316 + 0.004038 + 0.116495 + 0.003821 + 0.055123 + 0.005205 \\ & + 0.005998 + 0.080978 + 0.002463 + 0.000171 + 0.086533 + 0.070267 \\ & + 0.002305 + 0.002838 + 0.004871 + 0.0036618 r_{11}(1) = 0.448086. \end{aligned}$$

Table 6.1 Bit sequences and their associated probabilities of determining $r_{11}(1)$.

Bit sequence	Probability
10 000	$q_{12}(1)\,q_{13}(0)\,q_{16}(0)\,q_{17}(0)\,q_{1,10}(0) = 0.16 \times 0.81 \times 0.13 \times 0.82 \times 0.24 = 0.003316$
01 000	$q_{12}(0)\,q_{13}(1)\,q_{16}(0)\,q_{17}(0)\,q_{1,10}(0) = 0.84 \times 0.19 \times 0.13 \times 0.82 \times 0.24 = 0.004083$
00 100	$q_{12}(0)\,q_{13}(0)\,q_{16}(1)\,q_{17}(0)\,q_{1,10}(0) = 0.84 \times 0.81 \times 0.87 \times 0.82 \times 0.24 = 0.116495$
00 010	$q_{12}(0)\,q_{13}(0)\,q_{16}(0)\,q_{17}(1)\,q_{1,10}(0) = 0.84 \times 0.81 \times 0.13 \times 0.18 \times 0.24 = 0.003821$
00 001	$q_{12}(0)\,q_{13}(0)\,q_{16}(0)\,q_{17}(0)\,q_{1,10}(1) = 0.84 \times 0.81 \times 0.13 \times 0.82 \times 0.76 = 0.055123$
11 100	$q_{12}(1)\,q_{13}(1)\,q_{16}(1)\,q_{17}(0)\,q_{1,10}(0) = 0.16 \times 0.19 \times 0.87 \times 0.82 \times 0.24 = 0.005205$
01 110	$q_{12}(0)\,q_{13}(1)\,q_{16}(1)\,q_{17}(1)\,q_{1,10}(0) = 0.84 \times 0.19 \times 0.87 \times 0.18 \times 0.24 = 0.005998$
00 111	$q_{12}(0)\,q_{13}(0)\,q_{16}(1)\,q_{17}(1)\,q_{1,10}(1) = 0.84 \times 0.81 \times 0.87 \times 0.18 \times 0.76 = 0.080978$
11 001	$q_{12}(1)\,q_{13}(1)\,q_{16}(0)\,q_{17}(0)\,q_{1,10}(1) = 0.16 \times 0.19 \times 0.13 \times 0.82 \times 0.76 = 0.002463$
11 010	$q_{12}(1)\,q_{13}(1)\,q_{16}(0)\,q_{17}(1)\,q_{1,10}(0) = 0.16 \times 0.19 \times 0.13 \times 0.18 \times 0.24 = 0.000171$
01 101	$q_{12}(0)\,q_{13}(1)\,q_{16}(1)\,q_{17}(0)\,q_{1,10}(1) = 0.84 \times 0.19 \times 0.87 \times 0.82 \times 0.76 = 0.086533$
10 101	$q_{12}(1)\,q_{13}(0)\,q_{16}(1)\,q_{17}(0)\,q_{1,10}(1) = 0.16 \times 0.81 \times 0.87 \times 0.82 \times 0.76 = 0.070267$
10 011	$q_{12}(1)\,q_{13}(0)\,q_{16}(0)\,q_{17}(1)\,q_{1,10}(1) = 0.16 \times 0.81 \times 0.13 \times 0.18 \times 0.76 = 0.002305$
01 011	$q_{12}(0)\,q_{13}(1)\,q_{16}(0)\,q_{17}(1)\,q_{1,10}(1) = 0.84 \times 0.19 \times 0.13 \times 0.18 \times 0.76 = 0.002838$
10 110	$q_{12}(1)\,q_{13}(0)\,q_{16}(1)\,q_{17}(1)\,q_{1,10}(0) = 0.16 \times 0.81 \times 0.87 \times 0.18 \times 0.24 = 0.004871$
11 111	$q_{12}(1)\,q_{13}(1)\,q_{16}(1)\,q_{17}(1)\,q_{1,10}(1) = 0.16 \times 0.19 \times 0.87 \times 0.18 \times 0.76 = 0.003618$

By adding these probabilities we get $r_{11}(1) = 0.448086$ and therefore $r_{11}(0) = 1 - r_{11}(1) = 0.551914$. The complete matrix **R** is then:

$$
\mathbf{R} = \left[\begin{array}{c:c:c:c:c:c:c:c:c:c}
0.551914 & 0.542753 & 0.546890 & 0 & 0 & 0.460714 & 0.545425 & 0 & 0 & 0.444092 \\
0.448086 & 0.457247 & 0.453110 & 0 & 0 & 0.539286 & 0.454575 & 0 & 0 & 0.555908 \\
\hdashline
0.493347 & 0 & 0.493991 & 0 & 0.537255 & 0.505034 & 0 & 0.506423 & 0.493347 & 0 \\
0506653 & 0 & 0.506009 & 0 & 0.462745 & 0.494966 & 0 & 0.493577 & 0.507451 & 0 \\
\hdashline
0 & 0 & 0.500333 & 0.505158 & 0.497937 & 0 & 0.500322 & 0 & 0.500413 & 0.499603 \\
0 & 0 & 0.499667 & 0.494842 & 0.502063 & 0 & 0.499678 & 0 & 0.499587 & 0.500397 \\
\hdashline
0 & 0.500446 & 0 & 0.507588 & 0.496965 & 0.499590 & 0 & 0.499477 & 0 & 0.499416 \\
0 & 0.499554 & 0 & 0.492412 & 0.503035 & 0.500410 & 0 & 0.500523 & 0 & 0.500584 \\
\hdashline
0.497476 & 0.497921 & 0 & 0.464662 & 0 & 0 & 0.497791 & 0.502437 & 0.497173 & 0 \\
0.502524 & 0.502079 & 0 & 0.535338 & 0 & 0 & 0.502209 & 0.497563 & 0.502827 & 0
\end{array}\right].
$$

The vertical step then updates the matrix **Q** using (6.15). To determine the probability $q_{11}(x)$:

$$
\begin{aligned}
q_{11}(0) &= \beta_{11} f_1^0 \prod_{m' \in M_1 \backslash 1} r_{m'1}(0) \\
&= \beta_{11} \times 0.78 \times (0.493347 \times 0.497476) = 0.191434\beta_{11} \\
q_{11}(1) &= \beta_{11} f_1^1 \prod_{m' \in M_1 \backslash 1} r_{m'1}(1) \\
&= \beta_{11} \times 0.22 \times (0.506653 \times 0.502524) = 0.056013\beta_{11}
\end{aligned}
$$

Since $0.191334\beta_{11} + 0.05601\beta_{11} = 1$, $\beta_{11} = \frac{1}{0.191334+0.056013} = 4.042903$. So:

$$\begin{aligned} q_{11}(0) &= 4.042903 \times 0.78 \times (0.493347 \times 0.497476) = 0.773636 \\ q_{11}(1) &= 4.042903 \times 0.22 \times (0.506653 \times 0.502524) = 0.226364 \end{aligned}.$$

The remaining q_{mn} are shown in $\mathbf{Q}$:

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c:c:c:c:c} 0.773636 & 0.839121 & 0.806481 & 0 & 0 & 0.132106 & 0.818884 & 0 & 0 & 0.239285 \\ 0.226364 & 0.160879 & 0.193519 & 0 & 0 & 0.867894 & 0.181116 & 0 & 0 & 0.760715 \\ \hdashline 0.812140 & 0 & 0.837461 & 0 & 0.444958 & 0.113039 & 0 & 0.211273 & 0.748185 & 0 \\ 0.187860 & 0 & 0.162539 & 0 & 0.555042 & 0.886961 & 0 & 0.788727 & 0.251815 & 0 \\ \hdashline 0 & 0 & 0.833978 & 0.492203 & 0.484126 & 0 & 0.844187 & 0 & 0.742212 & 0.201076 \\ 0 & 0 & 0.166022 & 0.507797 & 0.515874 & 0 & 0.155813 & 0 & 0.257788 & 0.798924 \\ \hdashline 0 & 0.860727 & 0 & 0.489773 & 0.485097 & 0.115241 & 0 & 0.215940 & 0 & 0.201196 \\ 0 & 0.139273 & 0 & 0.510227 & 0.514903 & 0.884759 & 0 & 0.784060 & 0 & 0.798804 \\ \hdashline 0.809608 & 0.861934 & 0 & 0.532711 & 0 & 0 & 0.845514 & 0.213942 & 0.744684 & 0 \\ 0.190392 & 0.138066 & 0 & 0.467289 & 0 & 0 & 0.154486 & 0.786058 & 0.255316 & 0 \end{array}\right].$$

Finally the pseudo posterior probabilities q_n are determined using (6.18). To find $q_1(x)$:

$$\begin{aligned} q_1(0) &= \beta_1 f_1^0 \prod_{m \in M_1} r_{m1}(0) \\ &= \beta_1 \times 0.78 \times (0.551914 \times 0.493347 \times 0.500413) = 0.116898\beta_1 \\ q_1(1) &= \beta_1 f_1^1 \prod_{m \in M_1} r_{m1}(1) \\ &= \beta_1 \times 0.22 \times (0.448086 \times 0.506653 \times 0.502524) = 0.025099\beta_1 \end{aligned}.$$

$$\therefore \beta_1 = \frac{1}{0.116898 + 0.025099} = 7.042402$$

$$\begin{aligned} q_1(0) &= 7.042402 \times 0.78 \times (0.551914 \times 0.493347 \times 0.500413) = 0.808046 \\ q_1(1) &= 7.042402 \times 0.22 \times (0.448086 \times 0.506653 \times 0.502524) = 0.191954 \end{aligned}$$

Applying (6.19), the decoded symbol $\hat{c}_1 = 0$ since $q_1(0)$ has the higher probability. The pseudo posterior probabilities of the received word are:

$$Q' = \begin{bmatrix} 0.808046 & 0.860941 & 0.834162 & 0.497361 & 0.482065 & 0.115074 & 0.844356 & 0.215586 & 0.742528 & 0.200821 \\ 0.191954 & 0.139059 & 0.165838 & 0.502639 & 0.517935 & 0.884926 & 0.155644 & 0.784414 & 0.257472 & 0.799179 \end{bmatrix}$$

and the corresponding decoded code word is $\hat{c} = [0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$. We can see that this does not match the original transmitted code word and so further iterations are required. It turns out that the decoded code word is correct after the third iteration and the final matrices **R**, **Q** and **Q**′ are given along with the correct decoded code word:

$$\mathbf{R} = \left[\begin{array}{c:c:c:c:c:c:c:c:c:c}
0.549960 & 0.540086 & 0.544369 & 0 & 0 & 0.463092 & 0.542650 & 0 & 0 & 0.447890 \\
0.450040 & 0.459914 & 0.455631 & 0 & 0 & 0.536908 & 0.457350 & 0 & 0 & 0.552110 \\
\hdashline
0.493114 & 0 & 0.493650 & 0 & 0.545393 & 0.505532 & 0 & 0.507453 & 0.491301 & 0 \\
0.506886 & 0 & 0.506350 & 0 & 0.454607 & 0.494468 & 0 & 0.492547 & 0.508699 & 0 \\
\hdashline
0 & 0 & 0.499989 & 0.500176 & 0.502649 & 0 & 0.499989 & 0 & 0.499985 & 0.500012 \\
0 & 0 & 0.500011 & 0.499824 & 0.497351 & 0 & 0.500011 & 0 & 0.500015 & 0.499988 \\
\hdashline
0 & 0.499975 & 0 & 0.500415 & 0.503915 & 0.500023 & 0 & 0.500032 & 0 & 0.500030 \\
0 & 0.500025 & 0 & 0.499585 & 0.496085 & 0.499977 & 0 & 0.499968 & 0 & 0.499970 \\
\hdashline
0.496595 & 0.497094 & 0 & 0.457904 & 0 & 0 & 0.496955 & 0.503693 & 0.495668 & 0 \\
0.503405 & 0.502906 & 0 & 0.542096 & 0 & 0 & 0.503045 & 0.496307 & 0.504332 & 0
\end{array}\right]$$

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c:c:c:c:c}
0.772854 & 0.838418 & 0.806053 & 0 & 0 & 0.132534 & 0.818189 & 0 & 0 & 0.240031 \\
0.227146 & 0.161582 & 0.193947 & 0 & 0 & 0.867466 & 0.181811 & 0 & 0 & 0.759969 \\
\hdashline
0.810391 & 0 & 0.835883 & 0 & 0.456507 & 0.114177 & 0 & 0.212482 & 0.746725 & 0 \\
0.189609 & 0 & 0.164117 & 0 & 0.543493 & 0.885823 & 0 & 0.787518 & 0.253275 & 0 \\
\hdashline
0 & 0 & 0.832375 & 0.478244 & 0.499266 & 0 & 0.842265 & 0 & 0.740099 & 0.203955 \\
0 & 0 & 0.167625 & 0.521756 & 0.500734 & 0 & 0.157735 & 0 & 0.259901 & 0.796045 \\
\hdashline
0 & 0.859034 & 0 & 0.478005 & 0.498000 & 0.116425 & 0 & 0.217493 & 0 & 0.203943 \\
0 & 140966 & 0 & 0.521995 & 0.502000 & 0.883575 & 0 & 0.782507 & 0 & 0.796057 \\
\hdashline
0.808242 & 0.860424 & 0 & 0.520590 & 0 & 0 & 0.843871 & 0.215010 & 0.743407 & 0 \\
0.191758 & 0.139576 & 0 & 0.479410 & 0 & 0 & 0.156129 & 0.784990 & 0.256593 & 0
\end{array}\right]$$

$$\mathbf{Q}' = \begin{bmatrix} 0.806122 & 0.859023 & 0.832369 & 0.478419 & 0.501915 & 0.116434 & 0.842260 & 0.217514 & 0.740088 & 0.203963 \\ 0.193878 & 0.140977 & 0.167631 & 0.521581 & 0.498085 & 0.883566 & 0.157740 & 0.782486 & 0.259913 & 0.796037 \end{bmatrix}$$

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

The majority of the complexity of the sum–product algorithm is due to the horizontal step, since the number of possible non-binary sequences can become very large. However, it has been shown that fast Fourier transforms (FFTs) can be used to replace this step, resulting in a significant reduction in complexity [8].

6.3.4 Reducing the Decoding Complexity Using Fast Fourier Transforms

The horizontal step described previously involves finding all possible binary sequences that satisfy a parity check equation, determining the probability of each sequence and adding them all together, as defined in (6.10).

If we take the simple parity check equation $z_1 = c_1 \oplus c_2 \oplus c_3 = 0$ then to determine $r_{11}(x)$ we first need to find all the solutions of $c_2 \oplus c_3 = c_1 = x$. For $x = 0$ the solutions are $c_2 = 0$, $c_3 = 0$ and $c_2 = 1$, $c_3 = 1$. Therefore:

$$r_{11}(0) = q_{12}(0)q_{13}(0) + q_{12}(1)q_{13}(1).$$

For $x = 1$ the solutions are $c_2 = 0$, $c_3 = 1$ and $c_2 = 1$, $c_3 = 0$. Therefore:

$$r_{11}(1) = q_{12}(0)q_{13}(1) + q_{12}(1)q_{13}(0).$$

In general, we can write this as the *convolution* operation:

$$r_{11}(x) = \sum_{v=0}^{1} q_{12}(v)q_{13}(x - v),$$

where $v \in \mathrm{GF}(2)$.

This implies that the same result can be achieved by replacing the convolution operation with Fourier transforms. Therefore, to determine r_{mn} we must first calculate the product of the Fourier transforms of the other $q_{mn'}$ and then apply the inverse Fourier transform:

$$r_{mn}(x) = F^{-1}\left(\prod_{n' \in N_m/n} F\left(q_{mn'}(x)\right)\right), \tag{6.21}$$

where $F(\,)$ is the Fourier transform and F^{-1} is the inverse Fourier transform. Since all elements belong to the additive group $\mathbb{Z}_2$, the Fast Fourier transform reduces to the Hadamard transform [9] $\mathbf{W}_2$, where:

$$\mathbf{W}_2 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{6.22}$$

A property of the Hadamard matrix is that its inverse is also the Hadamard matrix, that is:

$$\mathbf{W}_2\mathbf{W}_2 = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \mathbf{I},$$

where $\mathbf{I}$ is the identity matrix. Consequently, the inverse Fourier transform can be obtained by also multiplying by the Hadamard matrix. We will now use FFTs to determine $r_{11}(0)$ and $r_{11}(1)$ for the first iteration of Example 3.2. From (6.20):

$$\begin{aligned} r_{11}(x) &= F^{-1}\left(\prod_{n' \in N_0 \backslash 0} F\left(q_{1n'}(x)\right)\right) \\ &= F^{-1}\left(\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} q_{13}(0) \\ q_{13}(1) \end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} q_{14}(0) \\ q_{14}(1) \end{bmatrix}\right.\end{aligned}$$

$$
\begin{aligned}
&\times \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}q_{16}(0)\\ q_{16}(1)\end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}q_{17}(0)\\ q_{17}(1)\end{bmatrix}\\
&\times \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}q_{1,10}(0)\\ q_{1,10}(1)\end{bmatrix}\Bigg)\\
&= F^{-1}\left(\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0.84\\ 0.16\end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0.81\\ 0.19\end{bmatrix}\right.\\
&\times \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0.13\\ 0.87\end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0.82\\ 0.18\end{bmatrix}\\
&\left.\times \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0.24\\ 0.76\end{bmatrix}\right)\\
&= F^{-1}\left(\frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 0.68\end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 0.62\end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -0.74\end{bmatrix}\right.\\
&\left.\times \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 0.64\end{bmatrix} \times \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -0.52\end{bmatrix}\right).\\
&= \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0.103828\end{bmatrix}\\
&= \begin{bmatrix}0.551914\\ 0.448086\end{bmatrix}
\end{aligned}
$$

Therefore, $r_{11}(0) = 0.551914$ and $r_{11}(1) = 0.448086$, which is identical to the result obtained in the previous example. It is quite obvious by comparing these two methods that FFTs are much simpler to perform. FFTs will also be applied to the decoding of non-binary LDPC codes, explained later in this chapter.

6.4 Construction of Non-Binary LDPC Codes Defined Over Finite Fields

A non-binary LDPC code is simply an LDPC code with a sparse parity check matrix containing elements that could be defined over groups, rings or fields. In this book, we concentrate solely on LDPC codes defined over finite fields GF(2^i), where i is a positive integer greater than 1. In 1998, Mackay presented the idea of LDPC codes over finite fields [10], proving that they can achieve increases in performance over their binary counterparts with increasing finite field size. Mackay also showed how the sum–product algorithm could be extended to decode non-binary LDPC codes, but the overall complexity was much higher. At that time it was therefore only feasible to decode non-binary LDPC codes over small finite fields.

There are only a few papers in the literature on non-binary LDPC codes and most of these codes are constructed by taking a known binary LDPC code and replacing its nonzero elements with randomly-generated finite field elements. Shu Lin has presented several structured methods to construct good non-binary LDPC codes using a technique known as *array dispersion* [11], one of which we now describe.

6.4.1 Construction of Non-Binary LDPC Codes from Reed–Solomon Codes

Shu Lin has demonstrated several array dispersion methods using tools such as Euclidean and finite geometries and also using a single code word from a very low-rate Reed–Solomon code [11]. Since this book is concerned with non-binary codes, this section explains how to construct non-binary LPDC codes from a Reed–Solomon code word.

In the context of a non-binary code defined over finite fields GF(q), array dispersion is an operation applied to each nonzero element in a matrix whereby each element is transformed to a location vector of length $q - 1$. For an element $\alpha^i \in$ GF(q), $0 \leq i \leq q - 2$, it will be placed in the ith index of the location vector. For example, the element α^5 in GF(8) would be placed in the fifth index of the location vector. The element in this location vector is used to build an array with each row defined as the previous row cyclically shifted to the right and multiplied by the primitive element in GF(q), resulting in a $(q - 1) \times (q - 1)$ array. Performing array dispersion on a matrix with dimensions $a \times b$ would result in a larger matrix with dimensions $a(q - 1) \times b(q - 1)$. Therefore, after array dispersion the element α^5 would become:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \alpha^5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \alpha^6 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \alpha^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha^3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \alpha^4 & 0 & 0 \end{pmatrix}$$

where the top row is the original location vector. For the zero element, array dispersion will result in a $(q - 1) \times (q - 1)$ zero matrix.

These non-binary LDPC codes are constructed from Reed–Solomon codes with message length $k = 2$. Since these code are maximum distance separable (MDS) their minimum Hamming distance is $d = n - k + 1 = n - 1$. Since a Reed–Solomon code defined over GF(q) has a block length of $n = q - 1$, the Reed–Solomon code will be a $(q - 1, 2, q - 2)$ Reed–Solomon code. Since $d = q - 2$, this means that the minimum weight of the code word is also $q - 2$, implying that the code word will contain $q - 2$ nonzero elements and a single zero. It can be shown that the vector containing all 1s, $[1, 1, 1, \ldots, 1]$, and the vector $[1, \alpha, \alpha^2, \ldots, \alpha^{q-2}]$ are both valid code words. As a Reed–Solomon code is linear, adding these two code words will result in another code word:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \end{bmatrix} + \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{q-2} \end{bmatrix} = \begin{bmatrix} 0 & 1+\alpha & 1+\alpha^2 & \cdots & 1+\alpha^{q-2} \end{bmatrix},$$

which has a weight of $q - 2$. This code word is then used to build a $(q - 1) \times (q - 1)$ array by cyclically shifting it to the right to form the next row of the array. Observe that the main diagonal of this array will be zeroes.

Example 6.4: Array dispersion of a (7, 2, 6) Reed–Solomon code word over GF(8): The code word of weight $q - 2 = 6$ is obtained by adding the two vectors:

$$\begin{aligned}&\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix} + \begin{bmatrix}1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6\end{bmatrix}\\ &= \begin{bmatrix}0 & \alpha^3 & \alpha^6 & \alpha & \alpha^5 & \alpha^4 & \alpha^2\end{bmatrix}.\end{aligned}$$

This then is used to build the 7×7 array:

$$\begin{pmatrix}
0 & \alpha^3 & \alpha^6 & \alpha & \alpha^5 & \alpha^4 & \alpha^2 \\
\alpha^2 & 0 & \alpha^3 & \alpha^6 & \alpha & \alpha^5 & \alpha^4 \\
\alpha^4 & \alpha^2 & 0 & \alpha^3 & \alpha^6 & \alpha & \alpha^5 \\
\alpha^5 & \alpha^4 & \alpha^2 & 0 & \alpha^3 & \alpha^6 & \alpha \\
\alpha & \alpha^5 & \alpha^4 & \alpha^2 & 0 & \alpha^3 & \alpha^6 \\
\alpha^6 & \alpha & \alpha^5 & \alpha^4 & \alpha^2 & 0 & \alpha^3 \\
\alpha^3 & \alpha^6 & \alpha & \alpha^5 & \alpha^4 & \alpha^2 & 0
\end{pmatrix}$$

Since Reed–Solomon codes are cyclic, each row is a code word and each column, when read from bottom to top, is also a code word, all with weights equal to 6. After applying array dispersion we will obtain a much larger 49×49 array. This is too large to reproduce here but a small part of it is shown below, corresponding to the 2×2 subarray circled.

$$\left[\begin{array}{ccccccc:ccccccc:c}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^3 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^4 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^5 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^6 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^2 & 0 & 0 & 0 & 0 & \cdots \\
\hdashline
0 & 0 & \alpha^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & \alpha^3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & \alpha^4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & \alpha^5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & \alpha^6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\hdashline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}\right].$$

6.5 Decoding Non-Binary LDPC Codes with the Sum–Product Algorithm

The sum–product algorithm for binary LDPC codes can be extended to decode non-binary LDPC codes, but with an increase in decoding complexity. Firstly, for a non-binary LDPC code defined over GF(q), each received symbol can be one of q different elements in GF(q). Secondly, the horizontal step becomes more complicated as there are now more possible non-binary sequences to satisfy the parity check equations. The matrices **Q** and **R** used in the horizontal and vertical steps of the sum–product algorithm are defined in (6.23) and (6.24) respectively:

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c}
q_{11}(0) & q_{12}(0) & q_{13}(0) & \cdots & q_{1,n-1}(0) & q_{1n}(0) \\
q_{11}(1) & q_{12}(1) & q_{13}(1) & \cdots & q_{1,n-1}(1) & q_{1n}(1) \\
q_{11}(\alpha) & q_{12}(\alpha) & q_{13}(\alpha) & \cdots & q_{1,n-1}(\alpha) & q_{1n}(\alpha) \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
q_{11}(\alpha^{q-2}) & q_{12}(\alpha^{q-2}) & q_{13}(\alpha^{q-2}) & \cdots & q_{1,n-1}(\alpha^{q-2}) & q_{1n}(\alpha^{q-2}) \\
\hdashline
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\hdashline
q_{m1}(0) & q_{m2}(0) & q_{m3}(0) & \cdots & q_{m,n-1}(0) & q_{mn}(0) \\
q_{m1}(1) & q_{m2}(1) & q_{m3}(1) & \cdots & q_{m,n-1}(1) & q_{mn}(1) \\
q_{m1}(\alpha) & q_{m2}(\alpha) & q_{m3}(\alpha) & \cdots & q_{m,n-1}(\alpha) & q_{mn}(\alpha) \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
q_{m1}(\alpha^{q-2}) & q_{m2}(\alpha^{q-2}) & q_{m3}(\alpha^{q-2}) & \cdots & q_{m,n-1}(\alpha^{q-2}) & q_{mn}(\alpha^{q-2})
\end{array}\right]. \tag{6.23}$$

$$\mathbf{R} = \left[\begin{array}{c:c:c:c:c:c}
r_{11}(0) & r_{12}(0) & r_{13}(0) & \cdots & r_{1,n-1}(0) & r_{1n}(0) \\
r_{11}(1) & r_{12}(1) & r_{13}(1) & \cdots & r_{1,n-1}(1) & r_{1n}(1) \\
r_{11}(\alpha) & r_{12}(\alpha) & r_{13}(\alpha) & \cdots & r_{1,n-1}(\alpha) & r_{1n}(\alpha) \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
r_{11}(\alpha^{q-2}) & r_{12}(\alpha^{q-2}) & r_{13}(\alpha^{q-2}) & \cdots & r_{1,n-1}(\alpha^{q-2}) & r_{1n}(\alpha^{q-2}) \\
\hdashline
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
\hdashline
r_{m1}(0) & r_{m2}(0) & r_{m3}(0) & \cdots & r_{m,n-1}(0) & r_{mn}(0) \\
r_{m1}(1) & r_{m2}(1) & r_{m3}(1) & \cdots & r_{m,n-1}(1) & r_{mn}(1) \\
r_{m1}(\alpha) & r_{m2}(\alpha) & r_{m3}(\alpha) & \cdots & r_{m,n-1}(\alpha) & r_{mn}(\alpha) \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\
r_{m1}(\alpha^{q-2}) & r_{m2}(\alpha^{q-2}) & r_{m3}(\alpha^{q-2}) & \cdots & r_{m,n-1}(\alpha^{q-2}) & r_{mn}(\alpha^{q-2})
\end{array}\right]. \tag{6.24}$$

It can be observed that each nonzero element defined over GF(q) in the parity check matrix $\mathbf{H}$ has q probabilities associated with it, instead of two probabilities as in the binary case.

6.5.1 Received Symbol Likelihoods

In (6.8) it was shown how to determine the likelihoods of a demodulated symbol for the AWGN channel. Assuming a non-binary LDPC code defined over GF(q) and that M-PSK modulation is chosen, then provided that $q = M$ the likelihoods of the demodulated symbols are also the likelihoods of the received symbols. However, for the case where $q > M$ and M divides q we must concatenate the likelihoods of the demodulated symbols. As an example, take an LDPC code defined over GF(16) and QPSK modulation. We know that a finite field element in GF(16) is made up of four bits and a QPSK modulated symbol is made up of two bits. Therefore, two consecutive QPSK modulated symbols contain a finite field element. The element α in GF(16) is represented as 0010 in binary. Therefore, the likelihood of a received symbol being α is the product of the likelihood that one demodulated symbol is 00 and the likelihood that the neighbouring demodulated symbol is 10, that is, from (6.8), $P(r_i = \alpha) = f^{\alpha} = g^{00}g^{10}$.

6.5.2 Permutation of Likelihoods

In general, the parity check equations are of the form:

$$z_i = \sum_{j=1}^{n} h_{ij}\, c_j, \tag{6.25}$$

where h_{ij} and $c_j \in \mathrm{GF}(q)$. The parity check equation is satisfied when:

$$h_{11}c_1 + h_{12}c_2 + \cdots + h_{1n}c_n = 0.$$

For the horizontal step we calculate the probabilities $r_{ij}(x)$ from (6.10) by first substituting all possible non-binary elements into the coded symbols that satisfy the parity check equation when $c_j = x$, that is:

$$h_{i1}c_1 + h_{i2}c_2 + \cdots + h_{in}c_n = h_{ij}\, c_j,$$

and then determining the probability of each sequence. This is more complicated than the binary case since we must now consider the non-binary parity check matrix elements. Each coded symbol has q likelihoods associated with it. When the coded symbol is multiplied by a non-binary parity check element we can compensate for this by cyclically shifting downwards this column vector of likelihoods, with the exception of the first likelihood, corresponding to the probability of the coded symbol being zero. The number of cyclic shifts is equal to the power of the primitive element that is multiplied with the coded symbol. This is illustrated below:

$$c_j \rightarrow \begin{bmatrix} q_{ij}(0) \\ q_{ij}(1) \\ \vdots \\ q_{ij}(\alpha^{q-2}) \end{bmatrix} \quad \alpha c_j \rightarrow \begin{bmatrix} q_{ij}(0) \\ q_{ij}(\alpha^{q-2}) \\ q_{ij}(1) \\ \vdots \end{bmatrix} \alpha^2 c_j \rightarrow \begin{bmatrix} q_{ij}(0) \\ \vdots \\ q_{ij}(\alpha^{q-2}) \\ q_{ij}(1) \end{bmatrix} \quad \text{etc.}$$

This cyclic shift of the likelihoods is known as a *permutation* [8] and transforms the parity check equation from (6.25) to:

$$c_1 + c_2 + \cdots + c_n = c_j,$$

which is more similar to the binary parity check equations. The inverse of a permutation is a *depermutation*, where the likelihoods are cyclically shifted upwards, again with the exception of the first likelihood.

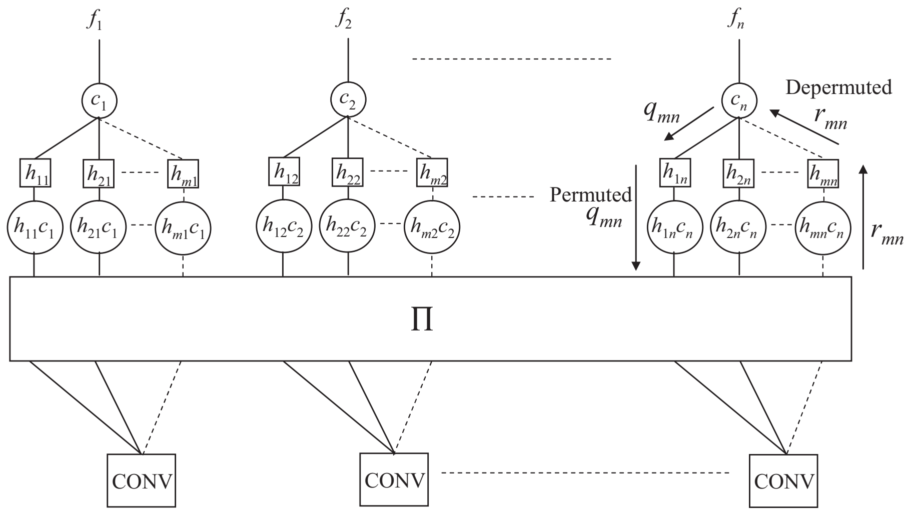


Figure 6.4 Generalized factor graph of a non-binary LDPC code [8].

6.5.3 Factor Graph of a Non-Binary LDPC Code

The factor graph of a non-binary LDPC code graphically shows the operation of the sum–product decoding algorithm (Figure 6.4).

The factor graph for non-binary LPDC codes is similar to that for binary LPDC codes, but we must now take into account the non-binary elements in the parity check matrix, which are denoted as h_{ij}, $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. In Figure 6.4 the number of parity check matrix elements connected to a coded symbol c_j is the column weight of the code and the number of connections to each parity check z_i is the row weight of the code. The likelihoods of each coded symbol f_j are column vectors containing the q likelihoods of the coded symbol being an element in GF(q). The block labelled Π connects the non-binary elements in each row to the parity checks. In Figure 6.5 a generalized factor graph is shown, with FFT blocks replacing the convolutional blocks, which reduces complexity.

A factor graph of the parity check matrix in (6.26) is show in Figure 6.6.

$$\mathbf{H} = \begin{bmatrix} \alpha & 0 & 1 & \alpha & 0 & 1 \\ \alpha^2 & \alpha & 0 & 1 & 1 & 0 \\ 0 & \alpha & \alpha^2 & 0 & \alpha^2 & 1 \end{bmatrix}. \tag{6.26}$$

6.5.4 The Fast Fourier Transform for the Decoding of Non-Binary LDPC Codes

In Section 6.3.4, the FFT was used to reduce the complexity of the horizontal step in the sum–product algorithm. For binary decoding of LDPC codes, the matrix **Q** contains

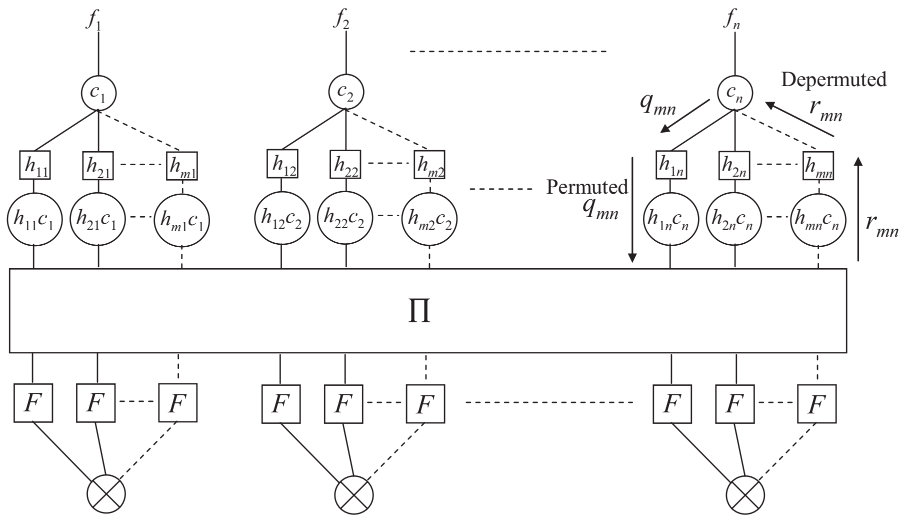

f_1
f_2
f_n
c_1
c_2
c_n
q_{mn}
Depermuted
r_{mn}
h_{11} h_{21} h_{m1}
h_{12} h_{22} h_{m2}
h_{1n} h_{2n} h_{mn}
Permuted
q_{mn}
r_{mn}
$h_{11}c_1$ $h_{21}c_1$ $h_{m1}c_1$
$h_{12}c_2$ $h_{22}c_2$ $h_{m2}c_2$
$h_{1n}c_n$ $h_{2n}c_n$ $h_{mn}c_n$
Π
F

Figure 6.5 Generalized factor graph of a non-binary LDPC code showing the replacement of the convolution operations with FFTs.

1×2 column vectors containing two probabilities, $q_{mn}(0)$ and $q_{mn}(1)$. The Fourier transform of this column vector is then obtained by multiplying by the Hadamard matrix defined in (6.21).

However, for non-binary LDPC codes defined over GF(q), $\mathbf{Q}$ contains $1 \times q$ column vectors and the Fourier transform is obtained by multiplying by the tensor product of

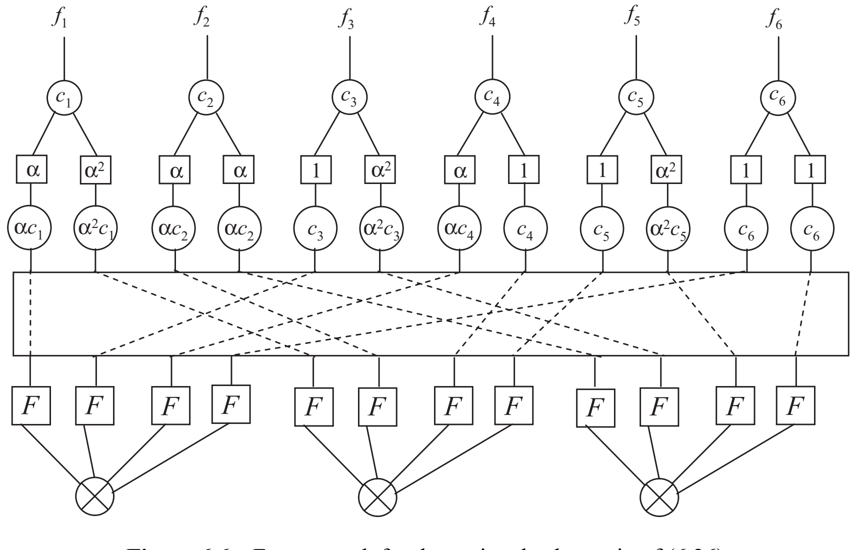

f_1 f_2 f_3 f_4 f_5 f_6
c_1 c_2 c_3 c_4 c_5 c_6
α α^2 α α 1 α^2 α 1 1 α^2 1 1
αc_1 $\alpha^2 c_1$ αc_2 αc_2 c_3 $\alpha^2 c_3$ αc_4 c_4 c_5 $\alpha^2 c_5$ c_6 c_6
F

Figure 6.6 Factor graph for the parity check matrix of (6.26).

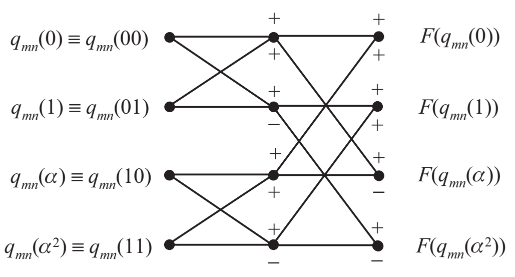

$q_{mn}(0) \equiv q_{mn}(00)$
$q_{mn}(1) \equiv q_{mn}(01)$
$q_{mn}(\alpha) \equiv q_{mn}(10)$
$q_{mn}(\alpha^2) \equiv q_{mn}(11)$
$F(q_{mn}(0))$
$F(q_{mn}(1))$
$F(q_{mn}(\alpha))$
$F(q_{mn}(\alpha^2))$
+
−

Figure 6.7 Radix-2 butterfly diagram for GF(4).

Hadamard matrices. An example of the tensor product of two Hadamard matrices is given below:

$$\mathbf{W_4} = \mathbf{W_2} \otimes \mathbf{W_2} = \begin{bmatrix} \mathbf{W_2} & \mathbf{W_2} \\ \mathbf{W_2} & -\mathbf{W_2} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}.$$

Therefore, for an LDPC code defined over GF(4), the Fourier transform of the 1 × 4 column vector of probabilities $q_{mn}(0)$, $q_{mn}(1)$, $q_{mn}(\alpha)$ and $q_{mn}(\alpha^2)$ is:

$$\begin{bmatrix} F\left(q_{mn}(0)\right) \\ F\left(q_{mn}(1)\right) \\ F\left(q_{mn}(\alpha)\right) \\ F\left(q_{mn}(\alpha^2)\right) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} q_{mn}(0) \\ q_{mn}(1) \\ q_{mn}(\alpha) \\ q_{mn}(\alpha^2) \end{bmatrix}.$$

As before, the inverse FFT is achieved by multiplying by the Hadamard matrix, that is $\mathbf{W_4W_4} = \mathbf{I_4}$. The FFT operation can be expressed in terms of its radix-2 butterfly diagram. An example of the radix-2 butterfly diagram for GF(4) is shown in Figure 6.7.

The ordering of the probabilities $q_{mn}(x)$ is very important when determining the FFT. It is essential that for each pair of $q_{mn}(x)$, the binary representation of the finite field elements x differ by *one bit*. In Figure 6.8, the radix-2 butterfly diagram for GF(16) is also given.

Example 6.5: Decoding a non-binary LDPC code with the FFT sum–product algorithm: In this example we decode the (6, 3) LDPC code defined over GF(4) with a parity check matrix given by (6.23). We assume that the transmitted code word was $\mathbf{c} = [2\ \ 1\ \ 0\ \ 3\ \ 0\ \ 2\ ]$ and the received word is $\mathbf{r} = [2\ \ 1\ \ 0\ \ \mathbf{2}\ \ \mathbf{2}\ \ 2]$, with two errors highlighted. Furthermore, it is assumed that

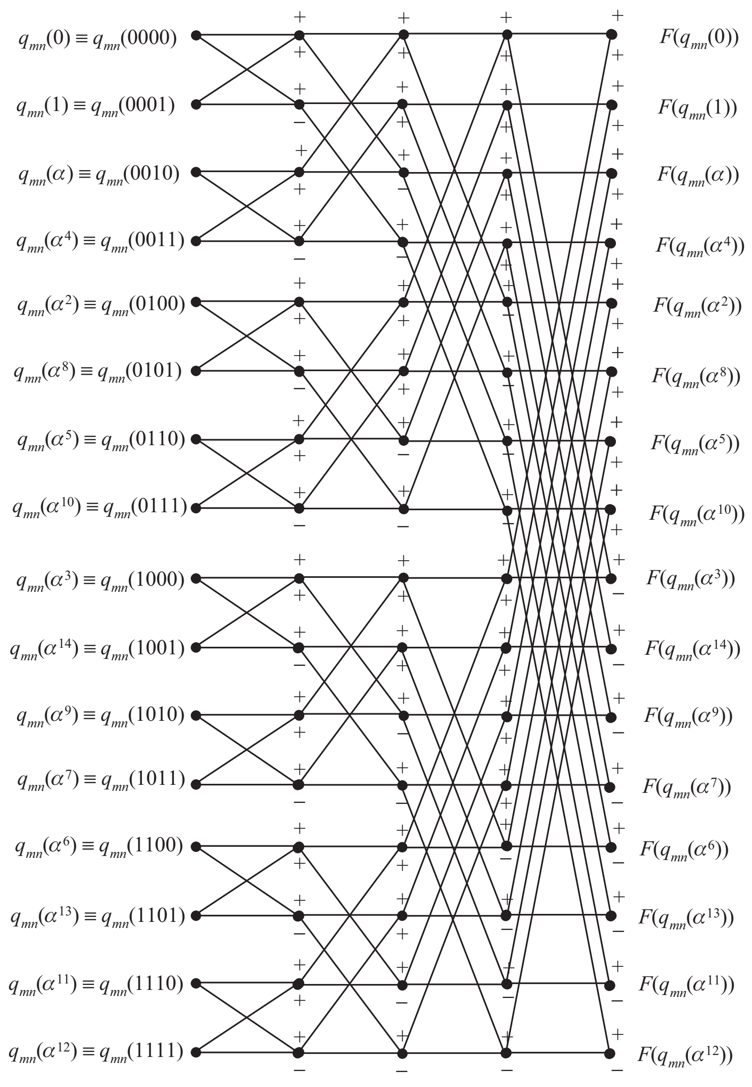

$q_{mn}(0) \equiv q_{mn}(0000)$
$q_{mn}(1) \equiv q_{mn}(0001)$
$q_{mn}(\alpha) \equiv q_{mn}(0010)$
$q_{mn}(\alpha^4) \equiv q_{mn}(0011)$
$q_{mn}(\alpha^2) \equiv q_{mn}(0100)$
$q_{mn}(\alpha^8) \equiv q_{mn}(0101)$
$q_{mn}(\alpha^5) \equiv q_{mn}(0110)$
$q_{mn}(\alpha^{10}) \equiv q_{mn}(0111)$
$q_{mn}(\alpha^3) \equiv q_{mn}(1000)$
$q_{mn}(\alpha^{14}) \equiv q_{mn}(1001)$
$q_{mn}(\alpha^9) \equiv q_{mn}(1010)$
$q_{mn}(\alpha^7) \equiv q_{mn}(1011)$
$q_{mn}(\alpha^6) \equiv q_{mn}(1100)$
$q_{mn}(\alpha^{13}) \equiv q_{mn}(1101)$
$q_{mn}(\alpha^{11}) \equiv q_{mn}(1110)$
$q_{mn}(\alpha^{12}) \equiv q_{mn}(1111)$
$F(q_{mn}(0))$
$F(q_{mn}(1))$
$F(q_{mn}(\alpha))$
$F(q_{mn}(\alpha^4))$
$F(q_{mn}(\alpha^2))$
$F(q_{mn}(\alpha^8))$
$F(q_{mn}(\alpha^5))$
$F(q_{mn}(\alpha^{10}))$
$F(q_{mn}(\alpha^3))$
$F(q_{mn}(\alpha^{14}))$
$F(q_{mn}(\alpha^9))$
$F(q_{mn}(\alpha^7))$
$F(q_{mn}(\alpha^6))$
$F(q_{mn}(\alpha^{13}))$
$F(q_{mn}(\alpha^{11}))$
$F(q_{mn}(\alpha^{12}))$

Figure 6.8 Radix-2 butterfly diagram for GF(16).

the received symbols have the following likelihoods:

$$\mathbf{f} = \begin{bmatrix} 0.182512 & 0.149675 & 0.444988 & 0.044678 & 0.412355 & 0.320530 \\ 0.046118 & 0.723055 & 0.324187 & 0.030350 & 0.073539 & 0.079952 \\ 0.615774 & 0.021827 & 0.133538 & 0.550805 & 0.436298 & 0.479831 \\ 0.155596 & 0.105443 & 0.097286 & 0.374167 & 0.077809 & 0.119687 \end{bmatrix}, \tag{6.27}$$

where each column in **f** contains the likelihoods of a received symbol being 0, 1, α or α^2. The matrix **Q** is then initialized as shown in (6.21). However, as described earlier, the elements in **Q** must be permuted due to the received coded symbols being multiplied by a non-binary element in the parity check matrix. The permuted version of **Q** is given in (6.28):

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c}
0.182512 & 0 & 0.444988 & 0.044678 & 0 & 0.320530 \\
0.046118 & 0 & 0.324187 & 0.030350 & 0 & 0.079952 \\
0.615774 & 0 & 0.133538 & 0.550805 & 0 & 0.479831 \\
0.155596 & 0 & 0.097286 & 0.374167 & 0 & 0.119687 \\
\hdashline
0.182512 & 0.149675 & 0 & 0.044678 & 0.412355 & 0 \\
0.046118 & 0.723055 & 0 & 0.030350 & 0.073539 & 0 \\
0.615774 & 0.021827 & 0 & 0.550805 & 0.436298 & 0 \\
0.155596 & 0.105443 & 0 & 0.374167 & 0.077809 & 0 \\
\hdashline
0 & 0.149675 & 0.444988 & 0 & 0.412355 & 0.320530 \\
0 & 0.723055 & 0.324187 & 0 & 0.073539 & 0.079952 \\
0 & 0.021827 & 0.133538 & 0 & 0.436298 & 0.479831 \\
0 & 0.105443 & 0.097286 & 0 & 0.077809 & 0.119687
\end{array}\right]. \tag{6.28}$$

Permuted **Q**

$$= \left[\begin{array}{c:c:c:c:c:c}
0.182512 & 0 & 0.444988 & 0.044678 & 0 & 0.320530 \\
0.155596 & 0 & 0.324187 & 0.374167 & 0 & 0.079952 \\
0.046118 & 0 & 0.133538 & 0.030350 & 0 & 0.479831 \\
0.615774 & 0 & 0.097286 & 0.550805 & 0 & 0.119687 \\
\hdashline
0.182512 & 0.149675 & 0 & 0.044678 & 0.412355 & 0 \\
0.615774 & 0.105443 & 0 & 0.030350 & 0.073539 & 0 \\
0.155596 & 0.723055 & 0 & 0.550805 & 0.436298 & 0 \\
0.046118 & 0.021827 & 0 & 0.374167 & 0.077809 & 0 \\
\hdashline
0 & 0.149675 & 0.444988 & 0 & 0.412355 & 0.320530 \\
0 & 0.105443 & 0.133538 & 0 & 0.436298 & 0.079952 \\
0 & 0.723055 & 0.097286 & 0 & 0.077809 & 0.479831 \\
0 & 0.021827 & 0.324187 & 0 & 0.073539 & 0.119687
\end{array}\right]. \tag{6.29}$$

We now perform the FFT operation on each of the 1×4 column vectors. For example, the Fourier transform of the column vector in the top-left-hand corner of the permuted **Q** matrix is determined by:

$$F\begin{pmatrix} q_{11}(0) \\ q_{11}(1) \\ q_{11}(\alpha) \\ q_{11}(\alpha^2) \end{pmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 0.182512 \\ 0.155596 \\ 0.046118 \\ 0.615774 \end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} 0.182512 + 0.155596 + 0.046118 + 0.615774 \\ 0.182512 - 0.155596 + 0.046118 - 0.615774 \\ 0.182512 + 0.155596 - 0.046118 - 0.615774 \\ 0.182512 - 0.155596 - 0.046118 + 0.615774 \end{bmatrix}$$

$$F\begin{pmatrix} q_{11}(0) \\ q_{11}(1) \\ q_{11}(\alpha) \\ q_{11}(\alpha^2) \end{pmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ -0.542740 \\ -0.323783 \\ 0.596571 \end{bmatrix}.$$

The FFT of each column vector in the permuted **Q** matrix is given in (6.30):

$$F(\mathbf{Q}) = \left[\begin{array}{c:c:c:c:c:c} 1 & 0 & 1 & 1 & 0 & 1 \\ -0.542740 & 0 & 0.157053 & -0.849944 & 0 & 0.600723 \\ -0.323783 & 0 & 0.538351 & -0.162310 & 0 & -0.199036 \\ 0.596571 & 0 & 0.084549 & 0.190965 & 0 & -0.119566 \\ \hdashline 1 & 1 & 0 & 1 & 1 & 0 \\ -0.323783 & 0.745461 & 0 & 0.190965 & 0.697305 & 0 \\ 0.596571 & -0.489765 & 0 & -0.849944 & -0.028213 & 0 \\ -0.542740 & -0.656996 & 0 & -0.162310 & -0.019673 & 0 \\ \hdashline 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0.745461 & 0.084549 & 0 & -0.019673 & 0.600723 \\ 0 & -0.489765 & 0.157053 & 0 & 0.697305 & -0.199036 \\ 0 & -0.656996 & 0.538351 & 0 & -0.0282143 & -0.119566 \end{array}\right]. \tag{6.30}$$

To complete the horizontal step we apply (6.20) to determine the probabilities $r_{mn}(x)$. To determine $r_{11}(x)$ we take the product of the Fourier transformed column vectors in $F(\mathbf{Q})$ corresponding to the third, fourth and sixth received symbols and then apply the inverse FFT to obtain a 1×4 column vector containing $r_{11}(x)$:

$$
\begin{aligned}
r_{11}(x) &= F^{-1}\left(\prod_{n' \in N_1/1} F\left(q_{1n'}(x)\right)\right) \\
&= F^{-1}\left(\frac{1}{2}\begin{bmatrix} 1 \\ 0.157053 \\ 0.538351 \\ 0.084549 \end{bmatrix} \times \begin{bmatrix} 1 \\ -0.849944 \\ -0.162310 \\ 0.190965 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0.600723 \\ -0.199036 \\ -0.119566 \end{bmatrix}\right) \\
&= F^{-1}\left(\frac{1}{2}\begin{bmatrix} 1 \\ -0.080188 \\ 0.0173917 \\ -0.003222 \end{bmatrix}\right) \\
&= \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -0.080188 \\ 0.0173917 \\ -0.003222 \end{bmatrix} \\
&= \frac{1}{4}\begin{bmatrix} 0.933982 \\ 1.100802 \\ 0.905642 \\ 1.059574 \end{bmatrix} = \begin{bmatrix} 0.233818 \\ 0.274878 \\ 0.226088 \\ 0.265216 \end{bmatrix}
\end{aligned}
$$

The remaining probabilities $r_{mn}(x)$ are given in (6.31):

$$
\mathbf{R} = \left[\begin{array}{c:c:c:c:c:c}
0.233818 & 0 & 0.313258 & 0.244365 & 0 & 0.277593 \\
0.274878 & 0 & 0.181512 & 0.272982 & 0 & 0.236553 \\
0.226088 & 0 & 0.325298 & 0.230033 & 0 & 0.258631 \\
0.265216 & 0 & 0.179931 & 0.252620 & 0 & 0.227223 \\
\hdashline
0.271356 & 0.242364 & 0 & 0.208230 & 0.286092 & 0 \\
0.222772 & 0.264788 & 0 & 0.295891 & 0.338076 & 0 \\
0.278277 & 0.236078 & 0 & 0.207616 & 0.190862 & 0 \\
0.227595 & 0.256769 & 0 & 0.288262 & 0.184970 & 0 \\
\hdashline
0 & 0.244755 & 0.264237 & 0 & 0.273865 & 0.238776 \\
0 & 0.244347 & 0.269750 & 0 & 0.233789 & 0.234406 \\
0 & 0.254745 & 0.231358 & 0 & 0.245066 & 0.260604 \\
0 & 0.256153 & 0.234655 & 0 & 0.247279 & 0.266214
\end{array}\right]. \tag{6.31}
$$

The matrix **R** is now complete, but it must be depermuted before it can be used in the vertical step. The depermuted **R** is given in (6.32):

$$\text{Depermuted } \mathbf{R} = \left[\begin{array}{c:c:c:c:c:c} 0.233818 & 0 & 0.313258 & 0.244365 & 0 & 0.277593 \\ 0.226088 & 0 & 0.181512 & 0.230033 & 0 & 0.236553 \\ 0.265216 & 0 & 0.325298 & 0.252620 & 0 & 0.258631 \\ 0.274878 & 0 & 0.179931 & 0.272982 & 0 & 0.227223 \\ \hdashline 0.271356 & 0.242364 & 0 & 0.208230 & 0.286092 & 0 \\ 0.227595 & 0.236078 & 0 & 0.295891 & 0.338076 & 0 \\ 0.222772 & 0.256769 & 0 & 0.207616 & 0.190862 & 0 \\ 0.278277 & 0.264788 & 0 & 0.288262 & 0.184970 & 0 \\ \hdashline 0 & 0.244755 & 0.264237 & 0 & 0.273865 & 0.238776 \\ 0 & 0.254745 & 0.234655 & 0 & 0.247279 & 0.234406 \\ 0 & 0.256153 & 0.269750 & 0 & 0.233789 & 0.260604 \\ 0 & 0.244347 & 0.231358 & 0 & 0.245066 & 0.266214 \end{array}\right]. \tag{6.32}$$

The vertical step is carried out in the same way as for the binary case. To determine $q_{11}(x)$ we apply (6.13):

$$q_{11}(0) = \beta_{11} f_1^0 \prod_{m' \in M_1/1} r_{m'1}(0) = \beta_{11} \times 0.182512 \times 0.271356 = 0.049526\beta_{11}$$

$$q_{11}(1) = \beta_{11} f_1^1 \prod_{m' \in M_1/1} r_{m'1}(1) = \beta_{11} \times 0.046118 \times 0.227595 = 0.010496\beta_{11}$$

$$q_{11}(\alpha) = \beta_{11} f_1^{\alpha} \prod_{m' \in M_1/1} r_{m'1}(\alpha) = \beta_{11} \times 0.615774 \times 0.222772 = 0.137177\beta_{11}$$

$$q_{11}(\alpha^2) = \beta_{11} f_1^{\alpha^2} \prod_{m' \in M_1/1} r_{m'1}(\alpha^2) = \beta_{11} \times 0.155596 \times 0.278277 = 0.043299\beta_{11}$$

$$\beta_{11} = \frac{1}{0.049256 + 0.010496 + 0.137177 + 0.043299} = 4.162712$$

$$\begin{aligned} q_{11}(0) &= 0.049526\beta_{11} = 0.205930 \\ q_{11}(1) &= 0.010496\beta_{11} = 0.043644 \\ q_{11}(\alpha) &= 0.137177\beta_{11} = 0.570388 \\ q_{11}(\alpha^2) &= 0.043299\beta_{11} = 0.180039 \end{aligned}.$$

The complete matrix **Q** is given in (6.33):

$$\mathbf{Q} = \left[\begin{array}{c:c:c:c:c:c}
0.205930 & 0 & 0.46625 & 0.038683 & 0 & 0.303488 \\
0.043644 & 0 & 0.301653 & 0.037341 & 0 & 0.074315 \\
0.570388 & 0 & 0.142839 & 0.475497 & 0 & 0.495852 \\
0.180039 & 0 & 0.089252 & 0.448479 & 0 & 0.126345 \\
\hdashline
0.164650 & 0.145266 & 0 & 0.042123 & 0.447806 & 0 \\
0.040229 & 0.730398 & 0 & 0.026937 & 0.072108 & 0 \\
0.630104 & 0.022170 & 0 & 0.536854 & 0.404473 & 0 \\
0.165017 & 0.102165 & 0 & 0.394068 & 0.075612 & 0 \\
\hdashline
0 & 0.150837 & 0.537825 & 0 & 0.490530 & 0.343296 \\
0 & 0.709767 & 0.227035 & 0 & 0.103376 & 0.072970 \\
0 & 0.023304 & 0.167601 & 0 & 0.346251 & 0.478806 \\
0 & 0.116092 & 0.067538 & 0 & 0.059844 & 0.104928
\end{array}\right]. \tag{6.33}$$

Finally, the pseudo posterior probabilities are determined using (6.18) and are given in (6.34):

$$\mathbf{Q}' = \begin{bmatrix}
0.205930 & 0.145266 & 0.466255 & 0.038683 & 0.447806 & 0.303488 \\
0.043644 & 0.730398 & 0.301653 & 0.037341 & 0.072108 & 0.074315 \\
0.570388 & 0.022170 & 0.142839 & 0.475497 & 0.404473 & 0.495852 \\
0.180039 & 0.102165 & 0.089252 & 0.448479 & 0.075612 & 0.126345
\end{bmatrix}. \tag{6.34}$$

From (6.19), taking the highest likelihood in each column of (6.34) gives a decoded code word of $\hat{\mathbf{c}} = [\alpha \quad 1 \quad 0 \quad \alpha \quad 0 \quad \alpha]$. The first iteration of the sum–product algorithm has corrected the fifth received symbol but there is still the error in the fourth received symbol. It turns out that only one more iteration is required to correct this error too, and the relevant matrices from the second iteration are given. The updated matrix **Q** in (6.33) is permuted and the FFT is applied to each column vector, as shown in (6.35). The matrix **R** is depermuted and shown in (6.36), and the updated matrix **Q** and pseudo posterior probabilities are given in

(6.37) and (6.38) respectively:

$$F(\mathbf{Q}) = \left[\begin{array}{c|c|c|c|c|c}
1 & 0 & 1 & 1 & 0 & 1 \\
-0.500852 & 0 & 0.218190 & -0.847952 & 0 & 0.598679 \\
-0.228062 & 0 & 0.535817 & -0.025675 & 0 & -0.244394 \\
0.552635 & 0 & 0.111015 & 0.028361 & 0 & -0.140334 \\
\hline
1 & 1 & 0 & 1 & 1 & 0 \\
-0.340666 & 0.751328 & 0 & 0.157955 & 0.704559 & 0 \\
0.589508 & -0.505138 & 0 & -0.861880 & -0.039829 & 0 \\
-0.590243 & -0.665128 & 0 & -0.127581 & -0.046837 & 0 \\
\hline
0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0.721208 & 0.210727 & 0 & 0.100747 & 0.644204 \\
0 & -0.466142 & 0.410854 & 0 & 0.673561 & -0.167468 \\
0 & -0.651719 & 0.529721 & 0 & 0.187811 & -0.103553
\end{array}\right]. \tag{6.35}$$

$$\text{Depermuted } \mathbf{R} = \left[\begin{array}{c|c|c|c|c|c}
0.223039 & 0 & 0.312657 & 0.238958 & 0 & 0.274386 \\
0.221579 & 0 & 0.186628 & 0.228330 & 0 & 0.227183 \\
0.276740 & 0 & 0.314472 & 0.256737 & 0 & 0.271947 \\
0.278642 & 0 & 0.186243 & 0.275975 & 0 & 0.226484 \\
\hline
0.276232 & 0.236345 & 0 & 0.206548 & 0.291534 & 0 \\
0.225755 & 0.244699 & 0 & 0.287521 & 0.336792 & 0 \\
0.232438 & 0.265419 & 0 & 0.203285 & 0.188251 & 0 \\
0.265575 & 0.253537 & 0 & 0.302645 & 0.183422 & 0 \\
\hline
0 & 0.239258 & 0.278016 & 0 & 0.291432 & 0.205369 \\
0 & 0.267581 & 0.228322 & 0 & 0.226443 & 0.230132 \\
0 & 0.255591 & 0.248275 & 0 & 0.224605 & 0.302287 \\
0 & 0.237570 & 0.245388 & 0 & 0.257521 & 0.262212
\end{array}\right]. \tag{6.36}$$

$$
\mathbf{Q} = \left[\begin{array}{c|c|c|c|c|c}
0.205545 & 0 & 0.485609 & 0.037950 & 0 & 0.252543 \\
0.042447 & 0 & 0.290544 & 0.035886 & 0 & 0.070589 \\
0.583537 & 0 & 0.130139 & 0.460471 & 0 & 0.556467 \\
0.168472 & 0 & 0.093708 & 0.465692 & 0 & 0.120401 \\
\hline
0.153792 & 0.137779 & 0 & 0.040705 & 0.471531 & 0 \\
0.038606 & 0.744379 & 0 & 0.026422 & 0.065340 & 0 \\
0.643804 & 0.021464 & 0 & 0.539167 & 0.384507 & 0 \\
0.163797 & 0.096378 & 0 & 0.393706 & 0.078622 & 0 \\
\hline
0 & 0.144486 & 0.535638 & 0 & 0.498018 & 0.333508 \\
0 & 0.722661 & 0.232931 & 0 & 0.102603 & 0.068878 \\
0 & 0.023662 & 0.161675 & 0 & 0.340255 & 0.494822 \\
0 & 0.109191 & 0.069757 & 0 & 0.059124 & 0.102792
\end{array}\right]. \tag{6.37}
$$

$$
\mathbf{Q}' = \begin{bmatrix}
0.205545 & 0.137779 & 0.485609 & 0.037950 & 0.471531 & 0.252543 \\
0.042447 & 0.744379 & 0.290544 & 0.035886 & 0.065340 & 0.070589 \\
0.583537 & 0.021464 & 0.130139 & 0.460471 & 0.384507 & 0.556467 \\
0.168472 & 0.096378 & 0.093708 & 0.465692 & 0.078622 & 0.120401
\end{bmatrix}. \tag{6.38}
$$

From the matrix $\mathbf{Q}'$, the decoded code word is now $\hat{\mathbf{c}} = [\alpha \quad 1 \quad 0 \quad \alpha^2 \quad 0 \quad \alpha]$, which matches the original transmitted code word.

6.6 Conclusions

A very important class of block code known as the low-density parity check (LDPC) code has been explained, with discussions on some structured construction methods for binary and non-binary LDPC codes. Additionally, a decoding algorithm called the sum–product algorithm has been studied in detail for use with binary and non-binary LDPC codes. A method to reduce the complexity of the sum–product algorithm has been given, using fast Fourier transforms based on the Hadamard matrix, which replaces the original horizontal step of the decoding algorithm.

The study of LDPC codes is very important as they are fast becoming one of the more popular coding schemes for a number of future applications in wireless communications and eventually magnetic storage. The binary LDPC codes are well known and perform as well as turbo codes, and in some cases can outperform turbo codes for large block lengths. Non-binary LDPC codes are less well known, but Mackay showed that these codes outperform binary LDPC codes with further increases in performance as the size of the finite field increases [10]. This is of course at the expense of higher complexity, but non-binary LDPC codes also have better convergence properties, requiring less iterations to decode a received word. With the inclusion of the FFT proposed by Barnault *et al.* [8], the complexity is reduced and non-binary LDPC codes can now be used practically in many future applications.

References

[1] Gallager, R.G. (1962) Low-density parity-check codes. *IRE Transactions on Information Theory*, **IT-8**, 21–8.

[2] Mackay, D.J. and Neal, R.M. (1995) Good codes based on very sparse matrices. *Cryptography and Coding*, 5th IMA Conference (ed. C. Boyd), Vol. **1025**, pp. 100–11 (Lecture Notes in Computer Science, Springer).

[3] Chung, S.Y., Forney, G.D. Jr., Richardson, T.J. and Urbanke, R. (2001) On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, **5** (2), 58–60.

[4] Ammar, B., Honary, B., Kou, Y. *et al.* (2004) Construction of low-density parity-check codes based on balanced incomplete block designs. *IEEE Transactions on Information Theory*, **50** (6), 1257–69.

[5] Hu, X.-Y., Eleftheriou, E. and Arnold, D.-M. (2001) Progressive edge-growth tanner graphs. Proceedings of IEEE GlobeCom, San Antonio, Texas, pp. 995–1001.

[6] Kou, Y., Lin, S. and Fossorier, M.P.C. (2001) Low-density parity-check codes based on finite geometries: a rediscovery and new results. *IEEE Transactions on Information Theory*, **47** (7), 2711–36.

[7] Moon, T.K. (2005) *Error Correction Coding. Mathematical Methods and Algorithms*, Wiley Interscience, ISBN 0-471-64800-0.

[8] Barnault, L. and Declerq, D. (2003) Fast decoding algorithm for LDPC over GF(2^q). IEEE Information Theory Workshop, Paris, France, pp. 70–3.

[9] Sylvester, J.J. (1867) Thoughts on orthogonal matrices, simultaneous sign-successions, and tesselated pavements in two or more colours, with applications to newton's rule, ornamental tile-work, and the theory of numbers. *Phil. Mag.*, **34**, 461–75.

[10] Davey, M.C. and Mackay, D.J. (1998) Low density parity check codes over GF(q). IEEE Information Theory Workshop, Killarney, Ireland, pp. 70–1.

[11] Lin, S., Song, S., Zhou, B. *et al.* (2007) Algebraic constructions of non-binary quasi-cyclic LDPC codes: array masking and dispersion. 9th International Symposium on Communication Theory and Applications (ISCTA), Ambleside, Lake District, UK.