Chapter 9: BCH, Reed-Solomon, and Related Codes

9.1 Introduction.

In Chapter 7 we gave one useful generalization of the $(7, 4)$ Hamming code of the Introduction: the family of $(2^m - 1, 2^m - m - 1)$ single error-correcting Hamming Codes. In Chapter 8 we gave a further generalization, to a class of codes capable of correcting a single *burst* of errors. In this Chapter, however, we will give a far more important and extensive generalization, the multiple-error correcting *BCH* and *Reed-Solomon* Codes.

To motivate the general definition, recall that the parity-check matrix of a Hamming Code of length $n = 2^m - 1$ is given by (see Section 7.4)

$$H = [\, \mathbf{v}_0 \quad \mathbf{v}_1 \quad \cdots \quad \mathbf{v}_{n-1} \,], \tag{9.1}$$

where $(\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{n-1})$ is some ordering of the $2^m - 1$ nonzero (column) vectors from $V_m = GF(2)^m$. The matrix H has dimensions $m \times n$, which means that it takes m parity-check bits to correct one error. If we wish to correct *two* errors, it stands to reason that m more parity checks will be required. Thus we might guess that a matrix of the general form

$$H_2 = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{n-1} \\ \mathbf{w}_0 & \mathbf{w}_1 & \cdots & \mathbf{w}_{n-1} \end{bmatrix},$$

where $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{n-1} \in V_m$, will serve as the parity-check matrix for a two-error-correcting code of length n. Since however the $\mathbf{v}_i$'s are distinct, we may view the correspondence $\mathbf{v}_i \to \mathbf{w}_i$ as a *function* from V_m into itself, and write H_2 as

$$H_2 = \begin{bmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \cdots & \mathbf{v}_{n-1} \\ \mathbf{f}(\mathbf{v}_0) & \mathbf{f}(\mathbf{v}_1) & \cdots & \mathbf{f}(\mathbf{v}_{n-1}) \end{bmatrix}. \tag{9.3}$$

But how should the function $\mathbf{f}$ be chosen? According to the results of Section 7.3, H_2 will define a two-error-correcting code iff the syndromes of the $1 + n + \binom{n}{2}$ error pattern of weights $0, 1$ and 2 are all distinct. Now any such syndrome is a sum of a (possible empty) subset of columns of H_2, and so is a vector in V_{2m}. But to be consistent with our present viewpoint let us break the syndrome $\mathbf{s} = (s_1, \ldots, s_{2m})$ in two halves: $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2)$, where $\mathbf{s}_1 = (s_1, \ldots, s_m)$ and $\mathbf{s}_2 = (s_{m+1}, \ldots, s_{2m})$ are both in V_m. With this convention, the syndrome of the all-zero pattern is $(\mathbf{0}, \mathbf{0})$; a single error in position i has $\mathbf{s} = (\mathbf{v}_i, \mathbf{f}(\mathbf{v}_i))$; a pair of errors at positions i and j gives $\mathbf{s} = (\mathbf{v}_i + \mathbf{v}_j, \mathbf{f}(\mathbf{v}_i) + \mathbf{f}(\mathbf{v}_j))$. We can unify these three cases by defining $\mathbf{f}(\mathbf{0}) = \mathbf{0}$ (notice that since $\mathbf{0}$ is not a column of H, $\mathbf{f}$ has not yet been defined at $\mathbf{0}$); then the condition that these syndromes are all distinct is that the system of equations

$$\begin{aligned} \mathbf{u} + \mathbf{v} &= \mathbf{s}_1 \\ \mathbf{f}(\mathbf{u}) + \mathbf{f}(\mathbf{v}) &= \mathbf{s}_2 \end{aligned} \tag{9.3}$$

has at most one solution $(\mathbf{u}, \mathbf{v})$ for each pair of vectors from V_m. (Naturally we do not regard the solution $(\mathbf{u}, \mathbf{v})$ as distinct from $(\mathbf{v}, \mathbf{u})$.)

Now we must try to find a function $\mathbf{f} : V_m \to V_m$, $\mathbf{f}(\mathbf{0}) = \mathbf{0}$, with the above property. We could try a linear mapping $\mathbf{f}(\mathbf{v}) = T\mathbf{v}$ for some linear transformation T, but this doesn't work (see Prob 9.1); so $\mathbf{f}$ must be nonlinear. To describe nonlinear functions of vectors $\mathbf{v} \in V_m$, we need to know that it is possible to define a multiplication on the vectors of V_m, which when combined with the vector addition makes V_m into a field. (The field is the Galois field $GF(2^m)$; the properties of finite fields that we shall need are stated in Appendix C.) Using this fact, it is easy to see (cf. Prob 9.2) that every function $\mathbf{f} : V_m \to V_m$ can be represented by a polynomial. Polynomials of degree ≤ 2 don't work (see Prob. 9.1); but $\mathbf{f}(\mathbf{v}) = \mathbf{v}^3$ does, as we shall shortly see. Hence (we change notation to emphasize that from now on we regard the elements of V_m not as

m-dimensional vectors over $GF(2)$, but as scalars from $GF(2^m)$) if $(\alpha_0, \alpha_1, \ldots \alpha_{n-1})$ is an arbitrary ordering of the nonzero elements of $GF(2^m)$, then the matrix

$$H_2 = \begin{bmatrix} \alpha_0 & \alpha_1 & \cdots & \alpha_{n-1} \\ \\ \alpha_0^3 & \alpha_1^3 & \cdots & \alpha_{n-1}^3 \end{bmatrix} \tag{9.4}$$

is the parity-check matrix of a two-error-correcting binary code of length $n = 2^m - 1$. Equivalently, $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1}) \in V_n$ is a codeword in the code with parity-check matrix H_2 iff $\sum_{i=0}^{n} C_i \alpha_i = \sum_{i=0}^{n} C_i \alpha_i^3 = 0$. Since as a matrix over $GF(2)$, H_2 has $2m$ rows (which are linearly independent for $m \geq 3$; see Prob. 9.5), the dimension of the code is $\geq n - 2m = 2^m - 1 - 2m$.

The *proof* that the matrix H_2 in (9.4) does indeed define a two-error-correcting code, as well as the generalization to t error-correcting codes, is given in the following celebrated theorem.

`Theorem 9.1.` *Let $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ be an ordering of the nonzero elements of $GF(2^m)$, and let t be a positive integer $\leq 2^{m-1} - 1$. Then the $t \times n$ matrix*

$$H = \begin{bmatrix} \alpha_0 & \alpha_1 & \cdots & \alpha_{n-1} \\ \alpha_0^3 & \alpha_1^3 & \cdots & \alpha_{n-1}^3 \\ \alpha_0^5 & \alpha_1^5 & \cdots & \alpha_{n-1}^5 \\ \vdots & & & \\ \alpha_0^{2t-1} & \alpha_1^{2t-1} & \cdots & \alpha_{n-1}^{2t-1} \end{bmatrix}$$

is the parity-check matrix of a binary (n, k) code capable of correcting all error patterns of weight $\leq t$, with dimension $k \geq n - mt$.

Proof. A vector $\mathbf{C}= (C_0, \ldots, C_{n-1}) \in V_n$ will be a codeword iff $H\mathbf{C}^T = \mathbf{0}$, which is equivalent to the following system of t linear equations in the C_i's:

$$\sum_{i=0}^{n-1} C_i \alpha_i^j = 0, \qquad j = 1, 3, \ldots, 2t-1. \tag{9.5}$$

Squaring the jth equation in (9.5), we get $0 = (\sum C_i \alpha_i^j)^2 = \sum C_i^2 \alpha_i^{2j} = \sum C_i \alpha_i^{2j}$ (since $(x+y)^2 = x^2 + y^2$ in characteristic 2 and $x^2 = x$ in $GF(2)$). Hence an equivalent definition of a codeword is the following system of $2t$ equations:

$$\sum_{i=0}^{n-1} C_i \alpha_i^j = 0, \qquad j = 1, 2, \ldots, 2t. \tag{9.6}$$

It follows that we could equally well use the $2t \times n$ parity-check matrix

$$H' = \begin{bmatrix} \alpha_0 & \alpha_1 & \cdots & \alpha_{n-1} \\ \alpha_0^2 & \alpha_1^2 & \cdots & \alpha_{n-1}^2 \\ \vdots & & & \\ \alpha_0^{2t} & \alpha_1^{2t} & \cdots & \alpha_{n-1}^{2t} \end{bmatrix}$$

to describe the code. According to Theorem 7.3, H, will be the parity-check matrix of a t-error-correcting code iff every subset of $2t$ or fewer columns of H' is linearly independent. Now a subset of r columns from H', where $r \leq 2t$, will have the form

$$B = \begin{bmatrix} \beta_1 & \cdots & \beta_r \\ \beta_1^2 & \cdots & \beta_r^2 \\ \vdots & & \\ \beta_1^{2t} & \cdots & \beta_r^{2t} \end{bmatrix},$$

where $\beta_1, \beta_2, \ldots, \beta_r$ are distinct nonzero elements of $GF(2^m)$. Now consider the matrix $\mathbf{B}'$ formed from the first r rows of β:

$$B' = \begin{bmatrix} \beta_1 & \cdots & \beta_r \\ \vdots & & \\ \beta_1^r & \cdots & \beta_r^r \end{bmatrix}.$$

The matrix B' is nonsingular, since its determinant is

$$\begin{aligned} \det(\mathbf{B}') &= \beta_1 \cdots \beta_r \det \begin{bmatrix} 1 & \cdots & 1 \\ \beta_1 & \cdots & \beta_r \\ \vdots & & \\ \beta_1^{r-1} & \cdots & \beta_r^{r-1} \end{bmatrix} \\ &= \beta_1 \cdots \beta_r \prod_{i<j} (\beta_j - \beta_i) \neq 0 \end{aligned}$$

by the Vandermonde determinant theorem (see Prob. 9.3). Hence the columns of $\mathbf{B}'$, let alone those of $\mathbf{B}$, cannot be linearly dependent, and so the code does correct all error patterns of weight $\leq t$. To verify the bound $k \geq n - mt$ on the dimension, observe that the original parity-check matrix H, viewed as a matrix with entries from $GF(2)$ rather than $GF(2^m)$, has dimensions $mt \times n$. And by the results of Section 7.1, this means that the dual code has dimension $\leq mt$, and so the code itself has dimension $\geq n - mt$. ∎

The codes described in Theorem 9.1 are called *BCH codes*, in honor of their inventors Bose, Ray-Chaudhuri, and Hocquenghem.* These codes are important, not so much because of Theorem 9.1 itself (other codes can have higher rates and larger minimum distances), but rather because there are efficient encoding and, especially, decoding algorithms for them. In the next section, we will see that if we choose exactly the right ordering $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$, BCH codes magically become cyclic codes, and so by the results of Chapter 8, the encoding automatically becomes simple. Additionally, this "cyclic" view of BCH codes will allow us to refine our estimates of the codes' dimensions. Then in Sections 9.3-9.5, we will fully describe one version of Berlekamp's famous decoding algorithm for BCH codes.

* Of course, it should be *BRH* codes, but an early reader of the paper by Bose and Ray-Chaudhuri mistakenly supposed that the second author's first name was Raymond, and so we are stuck with this innacurate acronym.

9.2. BCH Codes as cyclic codes.

Recall the definition of a t-error correcting BCH code of length $n = 2^m - 1$: $\mathbf{C} = (C_0, \ldots, C_{n-1})$ is a codeword iff $\sum_{i=0}^{n-1} C_i \alpha_i^j = 0$ for $j = 1, 3, \ldots, 2t-1$ (equivalently, for $j = 1, 2, 3, \ldots, 2t$), where $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ is an arbitrary but fixed ordering of the nonzero elements of $GF(2^m)$. Although changing from one such ordering to another cannot affect the error-correcting abilities of the code on a memoryless channel, nevertheless there is a "best" ordering for purposes of implementation. That ordering is to set $\alpha_i = \alpha^i$, where α is a *primitive root* (see Appendix C) of $GF(2^m)^*$. With this ordering, the definition becomes: $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ is a codeword iff

$$\sum_{i=0}^{n-1} C_i \alpha^{ij} = 0, \quad \text{for } j = 1, 3, \ldots, 2t-1 \text{ (or } j = 1, 2, 3, \ldots, 2t.) \tag{9.7}$$

In this realization, the BCH code becomes a *cyclic code*, in the sense of Chapter 8. To see that this is so, let $C(x) = C_0 + C_1 x + \cdots + C_{n-1} x^{n-1}$ be the generating function for the codeword $\mathbf{C}$; then (9.7) becomes

$$C(\alpha^j) = 0, \qquad j = 1, 2, \ldots, 2t. \tag{9.8}$$

Now let $\mathbf{C}^R$ be the right cyclic shift of the codeword $\mathbf{C}$; its generating function is, by Theorem 8.1, $C^R(x) = xC(x) \bmod (x^n - 1)$, which means that $C^R(x) = xC(x) + M(x)(x^n - 1)$ for some polynomial $M(x)$. Thus for $j = 1, 2, \ldots, 2t$,

$$C^R(\alpha^j) = \alpha^j C(\alpha^j) + M(\alpha^j)(\alpha^{jn} - 1).$$

But $C(\alpha^j) = 0$ by (9.8), and $\alpha^{jn} - 1 = 0$ since $\alpha^n = 1$. It follows that $C^R(\alpha^j) = 0$ for $j = 1, 2, \ldots, 2t$, so that $\mathbf{C}^R$ is also in the BCH code defined by (9.7), which means that the code is cyclic.

It now follows from Theorem 8.3 that every BCH code is characterized by its generator polynomial $g(x)$. But how can we compute $g(x)$? According to the definition, $g(x)$ is the least degree polynomial in the code, i.e., the least degree polynomial satisfying $g(\alpha) = g(\alpha^3) = \cdots = g(\alpha^{2t-1})$. Now the coefficients of $g(x)$ are in $GF(2)$, but the various powers of α are in the larger field $GF(2^m)$. Thus (see Appendix C) $g(x)$ is the **minimal polynomial** over $GF(2)$ of the subset of $A = \{\alpha, \alpha^3, \cdots, \alpha^{2t-1}\}$ of $GF(2^m)$. Hence if A^* is defined to be the set of all $GF(2)$- conjugates of elements in A, i.e. $A^* = \{\beta^{2^i} : \beta \in A, i \geq 0\}$, then

$$g(x) = \prod_{\beta \in A^*} (x - \beta). \tag{9.9}$$

We summarize these results in the following theorem.

`Theorem 9.2.` *If we define the t-error correcting BCH code of length $n = 2^m - 1$ by (9.7) or (9.8), then the code is cyclic, with generator polynomial given by (9.9). Thus the dimension of the code is given by $n - deg(g)$, i.e., $k = n - |A^*|$, where A^* is the set of $GF(2)$-conjugates of $A = \{\alpha, \alpha^3, \ldots, \alpha^{2t-1}\}$ in $GF(2^m)$.*
∎

Example 9.1. Consider the 3-error correcting BCH code of length 15. Let α be a primitive root in $GF(16)$; then by Theorem 9.2, the generator polynomial is the minimal polynomial of the set $A = \{\alpha, \alpha^3, \alpha^5\}$. The conjugates of α are $(\alpha, \alpha^2, \alpha^4, \alpha^8)$; of α^3: $(\alpha^3, \alpha^6, \alpha^{12}, \alpha^9)$; of α^5: (α^5, α^{10}). Hence

$$A^* = \{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6 \alpha^8, \alpha^9, \alpha^{10}, \alpha^{12}\},$$

and so by Theorem 9.2, the dimension is $15 - 10 = 5$.

To actually compute $g(x)$ for this example, we need a concrete realization of $GF(16)$. Let's represent $GF(16)$ according to powers of a primitive root α that satisfies $\alpha^4 = \alpha + 1$. In Table 9.1 the element α^j is given as a polynomial of degree ≤ 3 in a α; for example, $\alpha^{11} = \alpha^3 + \alpha^2 + \alpha$. The generator polynomial $g(x)$ is the product of the minimal polynomials of α, α^3, and α^5. The minimal polynomial of α is by definition $x^4 + x + 1$. The minimal polynomial of α^3 — call it $g_3(x) = g_{30} + g_{31}x + g_{32}x^2 + g_{33}x^3 + g_{34}x^4$ — must satisfy $g_3(\alpha^3) = 0$. From Table 9.1 this is equivalent to $g_{30}[0001] + g_{31}[1000] + g_{32}[1100] + g_{33}[1010] + g_{34}[1111] = [0000]$. The

* This ordering is still not unique, since there are many (to be exact, $\phi(2^m - 1)$) primitive roots in $GF(2^m)$.

only nontrivial solution to this set of 4 homogeneous equations in the 5 unknowns is $[g_{30}, g_{31}, g_{32}, g_{33}, g_{34}] = [11111]$, and so $g_3(x) = x^4 + x^3 + x^2 + x + 1$. Similarily, $g_5(x) = g_{50} + g_{51}x + g_{52}x^2$ (we already know that α^5 has only two conjugates, α^5 and α^{10}) turns out to be $x^2 + x + 1$. Hence the generator polynomial of the three-error-correcting BCH code of length 15 is $g(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1) = x^{10}+x^8+x^5+x^4+x^2+x+1$. Similarly, the parity-check polynomial is $h(x) = (x^{15}+1)/g(x) = x^5+x^3+x+1$. (We emphasize, however, that $g(x)$ depends on the particular realization of $GF(16)$ given in Table 9.1. See Problem 9.6.) ∎

i	α^i
0	0000
1	0010
2	0100
3	1000
4	0011
5	0110
6	1100
7	1011
8	0101
9	1010
10	0111
11	1110
12	1111
13	1101
14	1001

Table 9.1. The field $GF(16)$ represented as powers of α, where $\alpha^4 = \alpha + 1$.

Let us summarize what we know about BCH codes so far: they can be designed to correct any desired number of errors up to about half the code's block length (Theorem 9.1), and they have a very nice algebraic characterization as cyclic codes. However, their practical importance is due almost wholly to the fact that they have a remarkably efficient *decoding* algorithm. We will begin our discussion of this algorithm in the following section.

9.3. Decoding BCH codes, Part I: The Key Equation.

In this section, we will derive the so-called *key equation*, which is the basis for the BCH decoding algorithm. Before we get to the key equation, however, we must present some preliminary material. We shall present this material more generally than is strictly necessary, so that we can refer to it later, when we discuss the decoding *erasures* as well as errors, both for BCH codes and for *Reed-Solomon* codes.

Thus let F be a field which contains a primitive nth root of unity α.* We first note that

$$1 - x^n = \prod_{i=0}^{n-1} (1 - \alpha^i x). \tag{9.10}$$

This is because the polynomials on both sides of (9.10) have degree n, constant term 1, and roots α^{-i}, for $i = 0, 1, \ldots, n-1$. Next, let

$$\mathbf{V} = (V_0, V_1, \ldots, V_{n-1})$$

be an n dimensional vector over F, and let

$$\widehat{\mathbf{V}} = (\widehat{V}_0, \widehat{V}_1, \ldots, \widehat{V}_{n-1})$$

be its *discrete Fourier transform* (DFT), whose components are defined as follows.

$$\widehat{V}_j = \sum_{i=0}^{n-1} V_i \alpha^{ij}, \qquad \text{for } j = 0, 1, \ldots, n-1. \tag{9.11}$$

We sometimes call the V_i's the "time domain" coordinates, and the $\widehat{V}_j$'s the "frequency domain" coordinates, of the vector $\mathbf{V}$. The time domain components can be recovered from the frequency domain components via the so-called "inverse DFT":

$$V_i = \frac{1}{n} \sum_{j=0}^{n-1} \widehat{V}_j \alpha^{-ij}, \qquad \text{for } i = 0, 1, \ldots, n-1. \tag{9.12}$$

(In (9.12) the "$1/n$" factor in front of the sum must be interpreted with some care, in view of the possibly finite characteristic of F. The number "n" is the sum $1 + 1 + \cdots + 1$ (n terms), and "$1/n$" is the inverse of this number. For example, if F has characteristic 2 and n is odd, then $1/n = 1$. Apart from this small subtlety, however, the proof of (9.12) is identical to the usual proof of the inverse DFT formula, and we leave it as Problem 9.8.) If we interpret the components of $\mathbf{V}$ and $\widehat{\mathbf{V}}$ as the coefficients of polynomials, i.e., if we define generating functions $V(x)$ and $\widehat{V}(x)$ by

$$V(x) = V_0 + V_1 x + \cdots + V_{n-1} x^{n-1} \tag{9.13}$$

and

$$\widehat{V}(x) = \widehat{V}_0 + \widehat{V}_1 x + \cdots + \widehat{V}_{n-1} x^{n-1}, \tag{9.14}$$

then the DFT and IDFT relationships (9.11) and (9.12) become

$$\widehat{V}_j = V(\alpha^j) \tag{9.15}$$

and

$$V_i = \frac{1}{n} \widehat{V}(\alpha^{-i}). \tag{9.16}$$

There are many interesting and useful relationships between the time domain and frequency domain coordinates of a given vector. One of them is that a "phase shift" in the time domain corresponds to a "time

* If the characteristic of F is finite, we assume that the characteristic does not divide n.

shift" in the frequency domain, in the following sense. If we multiply the ith component of $\mathbf{V}$ by $\alpha^{\mu i}$, i.e., if we define a new vector $\mathbf{V}_\mu$ as

$$\mathbf{V}_\mu = (V_0, V_1\alpha^\mu, \ldots, V_{n-1}\alpha^{\mu(n-1)}) \tag{9.17}$$

then its DFT is

$$\widehat{\mathbf{V}}_\mu = (\widehat{V}_\mu, \widehat{V}_{\mu+1}, \ldots, \widehat{V}_{\mu+n-1}), \tag{9.18}$$

where in (9.18) the subscripts are taken mod n. We leave the proof of (9.18) as Problem 9.10.

As coding theorists, we are always interested in the *weight* of a vector. The following classical theorem tells us how to estimate the weight in the time domain if we know something about the vector in the frequency domain.

`Theorem 9.3.` *(The BCH Argument). Suppose* $\mathbf{V}$ *is a nonzero vector with the property that* $\widehat{\mathbf{V}}$ *has* m *consecutive* 0 *components, i.e.,* $\widehat{V}_{j+1} = \widehat{V}_{j+2} = \cdots = \widehat{V}_{j+m} = 0$. *Then the weight of* $\mathbf{V}$ *is* $\geq m+1$.

`Proof:` Let $\widehat{\mathbf{W}}$ be the vector obtained by cyclically shifting $\widehat{\mathbf{V}}$ until its m consecutive 0s appear in positions $n-m, n-m+1, \ldots, n-1$, i.e., $\widehat{\mathbf{W}} = [* * \cdots * \overbrace{00\cdots0}^{m}]$. By (9.17) and (9.18), $\widehat{\mathbf{W}}$ is the DFT of a vector $\mathbf{W}$ whose weight is the same as the weight of $\mathbf{V}$. However, by (9.12), $W_i = \frac{1}{n}\widehat{W}(\alpha^{-i})$, where $\widehat{W}(x) = \widehat{W}_0 + \widehat{W}_1 x + \cdots + \widehat{W}_{n-m-1}x^{n-m-1}$. Since $\widehat{W}(x)$ is a nonzero polynomial of degree $\leq n-m-1$, it follows that $W_i = 0$ for *at most* $n-i-m$ values of i, and so $W_i \neq 0$ for *at least* $m+1$ values of i. Thus $\mathrm{wt}(\mathbf{V}) = \mathrm{wt}(\mathbf{W}) \geq m+1$. ∎

We are almost ready to introduce the key equation, but we need a few more definitions. With the vector $\mathbf{V}$ fixed, we define its *support set* I as follows:

$$I = \{i : 0 \leq i \leq n-1 \text{ and } V_i \neq 0\}. \tag{9.19}$$

We now define several polynomials associated with $\mathbf{V}$, the locator polynomial, the punctured locator polynomials, and the evaluator polynomial. The *locator polynomial* for $\mathbf{V}$ is

$$\sigma_{\mathbf{V}}(x) = \prod_{i\in I}(1-\alpha^i x). \tag{9.20}$$

For each value of $i \in I$ we also define the ith *punctured locator polynomial* $\sigma^{(i)}_{\mathbf{V}}(x)$:

$$\begin{aligned}\sigma^{(i)}_{\mathbf{V}}(x) &= \sigma_{\mathbf{V}}(x)/(1-\alpha^i x)\\ &= \prod_{\substack{j\in I\\ j\neq i}}(1-\alpha^j x).\end{aligned} \tag{9.21}$$

Finally, we define the *evaluator polynomial* for $\mathbf{V}$ as

$$\omega_{\mathbf{V}}(x) = \sum_{i\in I} V_i \sigma^{(i)}_{\mathbf{V}}(x). \tag{9.22}$$

We will need the following lemma later on, for example, in Sections 9.5 and 9.7 when we discuss the RS/BCH decoding algorithms.

`Lemma 9.1.` $\gcd(\sigma_{\mathbf{V}}(x), \omega_{\mathbf{V}}(x)) = 1$.

`Proof:` By (9.20), $\gcd(\sigma_{\mathbf{V}}(x), \omega_{\mathbf{V}}(x)) = \prod_{i\in J}(1-\alpha^i x)$, where $J = \{i \in I : \omega_{\mathbf{V}}(\alpha^{-i}) = 0\}$. By (9.22), if $i \in I$, $\omega_{\mathbf{V}}(\alpha^{-i}) = V_i\sigma^{(i)}_{\mathbf{V}}(\alpha^{-i})$. But by the definition of I, if $i \in I$, $V_i \neq 0$, and by (9.21), $\sigma^{(i)}_{\mathbf{V}}(\alpha^{-i}) = \prod_{\substack{j\in I\\ j\neq i}}(1-\alpha^{j-i}) \neq 0$. Hence the set J is empty, and so $\gcd(\sigma_{\mathbf{V}}(x), \omega_{\mathbf{V}}(x)) = 1$, as asserted. ∎

We now come to the promised "key equation."

`Theorem 9.4.` *(The Key Equation). For a fixed vector* $\mathbf{V}$*, the polynomials* $\widehat{V}(x)$*,* $\sigma_{\mathbf{V}}(x)$*, and* $\omega_{\mathbf{V}}(x)$ *satisfy*

$$\sigma_{\mathbf{V}}(x)\widehat{V}(x) = \omega_{\mathbf{V}}(x)(1-x^n). \tag{9.23}$$

`Proof:` Using the definitions (9.11), (9.14) and (9.22), we find that

$$\widehat{\mathbf{V}}(x) = \sum_{i\in I} V_i \sum_{j=0}^{n-1} x^j \alpha^{ij}. \tag{9.24}$$

According to (9.21), $\sigma_{\mathbf{V}}(x) = \sigma_{\mathbf{V}}^{(i)}(x)(1-\alpha^i x)$ for all $i \in I$, and so from (9.24) we have

$$\begin{aligned}\sigma_I(x)\widehat{\mathbf{V}}(x) &= \sum_{i\in I} V_i \sigma_{\mathbf{V}}^{(i)}(x)(1-\alpha^i x)\sum_{j=0}^{n-1} x^j\alpha^{ij} \\ &= \sum_{i\in I} V_i \sigma_{\mathbf{V}}^{(i)}(x)(1-x^n) \\ &= \omega_{\mathbf{V}}(x)(1-x^n). \qquad \blacksquare\end{aligned}$$

The following Corollary to Theorem 9.3 tells us how to reconstruct the nonzero components of $\mathbf{V}$ from $\sigma_{\mathbf{V}}(x)$ and $\omega_{\mathbf{V}}(x)$. It involves the *formal derivative* $\sigma'_{\mathbf{V}}(x)$ of the polynomial $\sigma_{\mathbf{V}}(x)$. (See Problem 9.17).

Corollary 9.1. . For each $i \in I$, we have

$$V_i = -\alpha^i \frac{\omega_{\mathbf{V}}(\alpha^{-i})}{\sigma'_{\mathbf{V}}(\alpha^{-i})}. \tag{9.25}$$

`Proof:` If we differentiate the the key equation (9.23) we get

$$\sigma_{\mathbf{V}}(x)\widehat{V}'(x) + \sigma'_{\mathbf{V}}(x)\widehat{V}(x) = \omega_{\mathbf{V}}(x)(-nx^{n-1}) + \omega'_{\mathbf{V}}(x)(1-x^n). \tag{9.26}$$

Note that if $x = \alpha^{-i}$ with $i \in I$, from (9.20) and (9.10) we see that both $\sigma_{\mathbf{V}}(x)$ and $1-x^n$ vanish. Thus if $x = \alpha^{-i}$, (9.26) becomes

$$\sigma'_{\mathbf{V}}(\alpha^{-i})\widehat{V}(\alpha^{-i}) = -n\alpha^i \omega_{\mathbf{V}}(\alpha^{-i}). \tag{9.27}$$

But from (9.16), $\widehat{V}(\alpha^{-i}) = nV_i$. This fact, combined with (9.27), completes the proof. $\blacksquare$

Corollary 9.1 says, in effect, that the time-domain coordinates of $\mathbf{V}$ can be recovered from $\sigma_{\mathbf{V}}(x)$ and $\omega_{\mathbf{V}}(x)$. The next Corollary says that if the first few frequency-domain coordinates of $\mathbf{V}$ are known, the rest can be recovered from $\sigma_{\mathbf{V}}(x)$ alone, via a simple recursion. In the statement of the Corollary, we suppose that the coefficients of $\sigma_{\mathbf{V}}(x)$ are given by

$$\sigma_{\mathbf{V}}(x) = 1 + \sigma_1 x + \cdots + \sigma_d x^d.$$

Corollary 9.2. For $j = d, d+1, \ldots, n-1$, we have

$$\widehat{V}_j = -\sum_{i=1}^{d} \sigma_i \widehat{V}_{j-i}. \tag{9.28}$$

`Proof:` The key equation implies that

$$\sigma_{\mathbf{V}}(x)\widehat{V}(x) \equiv 0 \pmod{1-x^n}. \tag{9.29}$$

What (9.29) says is that for each j in the range $0 \le j \le n-1$, the coefficient of x^j in the polynomial $\sigma_{\mathbf{V}}(x)\widehat{V}(x) \bmod (1-x^n)$ is 0. But this coefficient is $\sum_{i=0}^{d} \sigma_i \widehat{V}_{(j-i) \bmod n}$, so that for each j in the range $0 \le j \le n-1$, we have

$$\sum_{i=0}^{d} \sigma_i \widehat{V}_{j-i} = 0, \tag{9.30}$$

where subscripts are to be taken mod n and we have defined $\sigma_0 = 1$. But now the equations (9.30) for $j = d, d+1, \ldots, n-1$ are equivalent to the equations (9.28). $\blacksquare$

Example 9.2. We illustrate this material using the field $GF(16)$, in which the nonzero elements are represented by the powers of a primitive root α satisfying the equation $\alpha^4 = \alpha + 1$. We consider the vector

$$\mathbf{V} = (0, 0, \alpha^2, 0, 0, 0, 0, \alpha^7, 0, 0, 0, 0, 0, 0, 0).$$

Then the polynomial $V(x)$ defined in (9.13) is

$$V(x) = \alpha^2 x^2 + \alpha^7 x^7.$$

Using (9.11) or (9.15) we can calculate the DFT of $\mathbf{V}$:

$$\widehat{\mathbf{V}} = (\alpha^{12}, \alpha^9, 0, \alpha^3, 1, 0, \alpha^9, \alpha^6, 0, 1, \alpha^{12}, 0, \alpha^6, \alpha^3, 0).$$

Thus $\widehat{V}(x)$, as defined in (9.14), is

$$\begin{aligned}
\widehat{V}(x) &= \alpha^{12} + \alpha^9 x + \alpha^3 x^3 + x^4 + \alpha^9 x^6 + \alpha^6 x^7 + x^9 + \alpha^{12} x^{10} + \alpha^6 x^{12} + \alpha^3 x^{13} \\
&= (\alpha^{12} + \alpha^9 x)(1 + \alpha^6 x^3 + \alpha^{12} x^6 + \alpha^3 x^9 + \alpha^9 x^{12}) \\
&= (\alpha^{12} + \alpha^9 x)\frac{1 + x^{15}}{1 + \alpha^6 x^3} \\
&= \alpha^{12}\frac{1 + x^{15}}{1 + \alpha^{12} x + \alpha^9 x^2}.
\end{aligned} \tag{9.31}$$

The support set for $\mathbf{V}$ is $I = \{2, 7\}$, and so the locator polynomial for $\mathbf{V}$ is

$$\sigma_{\mathbf{V}}(x) = (1 + \alpha^2 x)(1 + \alpha^7 x) = 1 + \alpha^{12} x + \alpha^9 x^2. \tag{9.32}$$

The polynomials $\sigma_{\mathbf{V}}^{(i)}(x)$ defined in (9.21) are in this case

$$\sigma_{\mathbf{V}}^{(2)} = (1 + \alpha^7 x), \quad \sigma_{\mathbf{V}}^{(7)} = (1 + \alpha^2 x).$$

The evaluator polynomial $\omega_{\mathbf{V}}(x)$ defined in (9.22) is

$$\omega_{\mathbf{V}}(x) = \alpha^2(1 + \alpha^7 x) + \alpha^7(1 + \alpha^2 x) = \alpha^{12}. \tag{9.33}$$

Combining (9.31), (9.32), and (9.33), we see that the key equation indeed holds in this case. To check Corollary 1, we note that from (9.32), $\sigma'_{\mathbf{V}}(x) = \alpha^{12} = \omega_{\mathbf{V}}(x)$, so that Corollary 1 becomes simply $V_i = \alpha^i$, for $i \in I$, which is true ($V_2 = \alpha^2$ and $V_7 = \alpha^7$). Finally, note that Corollary 2 says in this case that

$$\widehat{V}_j = \alpha^{12}\widehat{V}_{j-1} + \alpha^9 \widehat{V}_{j-2} \qquad \text{for } j = 2, 3, \ldots, 14,$$

so that (using $\widehat{V}_0 = \alpha^{12}$ and $\widehat{V}_1 = \alpha^9$ as initial conditions)

$$\begin{aligned}
\widehat{V}_2 &= \alpha^{12} \cdot \alpha^9 + \alpha^9 \cdot \alpha^{12} = 0 \\
\widehat{V}_3 &= \alpha^{12} \cdot 0 + \alpha^9 \cdot \alpha^9 = \alpha^3 \\
\widehat{V}_4 &= \alpha^{12} \cdot \alpha^3 + \alpha^9 \cdot 0 = 1 \\
&\vdots \\
\widehat{V}_{14} &= \alpha^{12} \cdot \alpha^3 + \alpha^9 \cdot \alpha^6 = 0,
\end{aligned}$$

which agrees with our direct calculation of $\widehat{\mathbf{V}}$. ∎

With the preliminary material about the key equation out of the way, we can begin a serious discussion of the problem of decoding BCH codes. Suppose then that $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ is a codeword from the

t-error correcting BCH code of length n defined by (9.6), which is transmitted over a noisy channel, and that $\mathbf{R} = (R_0, R_1, \ldots, R_{n-1})$ is received. (We assume that the components of $\mathbf{R}$ are 0's and 1's, i.e., are elements of $GF(2)$.) We define the *error pattern* as the vector $\mathbf{E} = (E_0, E_1, \ldots, E_n) = \mathbf{R} - \mathbf{C}$. The decoder's first step is to compute the *syndromes* $S_1, S_2, \ldots, S_{2t}$, which are defined by

$$S_j = \sum_{i=0}^{n-1} R_i \alpha^{ij}, \qquad \text{for } j = 1, 2, \ldots, 2t. \tag{9.34}$$

Since $\mathbf{R} = \mathbf{C} + \mathbf{E}$, and $\mathbf{C}$ is a codeword, it follows that

$$S_j = \sum_{i=0}^{n-1} E_i \alpha^{ij}, \qquad \text{for } j = 1, 2, \ldots, 2t, \tag{9.35}$$

so that, as expected, the syndromes depend only on the error pattern and not on the transmitted codeword. Note also that on comparing (9.35) with (9.11), we see that S_j is the jth component of the DFT of the error pattern; in other words, the syndrome lets us see $2t$ consecutive components (the first, second,..., $2t$th) of $\widehat{\mathbf{E}}$. If we now define the *twisted* error pattern $\mathbf{V}$ as

$$\mathbf{V} = (E_0, E_1\alpha, E_2\alpha^2, \ldots, E_{n-1}\alpha^{n-1}), \tag{9.36}$$

it follows from (9.17) and (9.18) that $(S_1, S_2, \ldots, S_{2t}) = (\widehat{V}_0, \widehat{V}_1, \ldots, \widehat{V}_{2t})$.

The key equation applies to the vector $\mathbf{V}$ defined in (9.36); however, since we only know the the first $2t$ coefficients of $\widehat{V}(x)$ (i.e., $\widehat{V}_0, \widehat{V}_1, \ldots, \widehat{V}_{2t}$), we focus instead on the key equation *reduced mod* x^{2t}:

$$\sigma(x)\widehat{V}(x) \equiv \omega(x) \pmod{x^{2t}}. \tag{9.37}$$

(In (9.37) we have dropped the subscript $\mathbf{V}$'s on $\sigma(x)$ and $\omega(x)$.) From (9.19) and (9.36) we see that the support set I for $\mathbf{V}$ is the set of indices such that $E_i \neq 0$, i.e., the set of *error locations*. For this reason, the polynomial $\sigma(x)$ in (9.37) is called the *error locator polynomial*. Similarly, the polynomial $\omega(x)$ in (9.37) is called the *error evaluator polynomial*. Equation (9.37) is called the *BCH key equation*.

Now observe that if, given the syndrome of the received word $\mathbf{R}$, or equivalently, $\widehat{V}(x) \bmod x^{2t}$, we could somehow "solve" the BCH key equation (9.37) for the polynomials $\sigma(x)$ and $\omega(x)$, we could then easily recover the error pattern $\mathbf{E}$, and thus also the transmitted codeword $\mathbf{C} = \mathbf{R} - \mathbf{E}$. We could do this by first computing the n values $\sigma(\alpha^{-i})$, for $i = 0, 1, \ldots, n-1$, which would identify the support set I of $\mathbf{V}$ defined in (9.19). Then the nonzero components of $\mathbf{V}$ could be computed by (9.25), and this would give us the complete vector $\mathbf{V}$, or equivalently, $\mathbf{E}$ (see (9.36)). Alternatively, knowing $(\widehat{V}_0, \widehat{V}_1, \ldots, \widehat{V}_{2t})$, we could complete the vector $\widehat{\mathbf{V}}$ via (9.28), and then recover $\mathbf{V}$ via an inverse DFT. In the next section, we will see that there is a remarkably efficient algorithm for computing $\sigma(x)$ and $\omega(x)$ from the BCH key equation, provided we make the additional assumption that the actual number of errors that occurred is at most t. (This assumption is necessary, since a t error-correcting BCH code is not designed to correct more than t errors.)

9.4. Euclid's Algorithm for Polynomials

This section does not deal directly with the problem of decoding BCH codes. The reader should bear in mind, however, that our goal is to solve the BCH key equation (Eq. 9.37)) for $\sigma(x)$ and $\omega(x)$, given $\widehat{V}(x) \bmod x^{2t}$.

Throughout this section $a(x)$ and $b(x)$ will be fixed polynomials over a field F, with $\deg a(x) \geq \deg b(x)$.* Later $a(x)$ will be replaced by x^{2t}, and $b(x)$ by the syndrome polynomial $S(x)$.

Euclid's algorithm is a recursive procedure for finding the greatest common divisor (gcd for short) $d(x)$ of $a(x)$ and $b(x)$, and for producing an equation

$$u(x)a(x) + v(x)b(x) = d(x) \tag{9.38}$$

that expresses $d(x)$ as a linear combination of $a(x)$ and $b(x)$. The algorithm involves four sequences of polynomials: $(u_i(x)), (v_i(x)), (r_i(x)), (q_i(x))$. The initial conditions are

$$\begin{aligned} u_{-1}(x) = 1, \quad v_{-1}(x) = 0, \quad r_{-1}(x) = a(x), \\ u_0(x) = 0, \quad v_0(x) = 1, \quad r_0(x) = b(x). \end{aligned} \tag{9.39}$$

($q_{-1}(x)$ and $q_0(x)$ are not defined). For $i \geq 1$, $q_i(x)$ and $r_i(x)$ are defined to be the *quotient* and *remainder*, respectively, when $r_{i-2}(x)$ is divided by $r_{i-1}(x)$:

$$r_{i-2}(x) = q_i(x)r_{i-1}(x) + r_i(x), \qquad \deg r_i \leq \deg r_{i-1}. \tag{9.40}$$

The polynomials $u_i(x)$ and $v_i(x)$ are then defined by

$$u_i(x) = u_{i-2}(x) - q_i(x)u_{i-1}(x), \tag{9.41}$$

$$v_i(x) = v_{i-2}(x) - q_i(x)v_{i-1}(x). \tag{9.42}$$

Since the degrees of the remainders r_i are strictly decreasing, there will be a last nonzero one; call it $r_n(x)$. It turns out that $r_n(x)$ is the gcd of $a(x)$ and $b(x)$, and furthermore that the desired equation expressing the gcd as a linear combination of the original two polynomials (see Eq. (9.38)) is

$$u_n(x)a(x) + v_n(x)b(x) = r_n(x). \tag{9.43}$$

Since this particular aspect of Euclid's algorithm is not our main concern, we leave the proof of these facts to Prob. 9.18b.

What is more interesting to us at present is the list shown in Table 9.2 of intermediate relationships among the polynomials of Euclid's algorithm. It is not difficult to prove these properties by induction on i; see Prob. 9.18a.

Table 9.2. Properties of Euclid's Algorithm

A.	$v_i r_{i-1} - v_{i-1} r_i = (-1)^i a$	$0 \leq i \leq n+1$.
B.	$u_i r_{i-1} - u_{i-1} r_i = (-1)^{i+1} b$	$0 \leq i \leq n+1$.
C.	$u_i v_{i-1} - u_{i-1} v_i = (-1)^{i+1}$	$0 \leq i \leq n+1$.
D.	$u_i a + v_i b = r_i$	$-1 \leq i \leq n+1$
E.	$\deg(u_i) + \deg(r_{i-1}) = \deg(b)$	$1 \leq i \leq n+1$.
F.	$\deg(v_i) + \deg(r_{i-1}) = \deg(a)$	$0 \leq i \leq n+1$.

Example 9.3. Let $F = GF(2)$, $a(x) = x^8$, $b(x) = x^6 + x^4 + x^2 + x + 1$. The behavior of Euclid's algorithm is given in Table 9.3.

* By convention the degree of the zero polynomial is $-\infty$. This is done so that basic facts like $\deg(ab) = \deg(a) + \deg(b)$, $\deg(a+b) \leq \max(\deg(a), \deg(b))$, etc., will hold even if one of a or b is the zero polynomial.

Table 9.3. An Example of Euclid's Algorithm

i	u_i	v_i	r_i	q_i
-1	1	0	x^8	$\cdots$
0	0	1	$x^6+x^4+x^2+x+1$	$\cdots$
1	1	x^2+1	x^3+x+1	x^2+1
2	x^3+1	$x^5+x^3+x^2$	x^2	x^3+1
3	x^4+x+1	$x^6+x^4+x^3+x^2+1$	$x+1$	x
4	$x^5+x^4+x^3+x^2$	$x^7+x^6+x^3+x+1$	1	$x+1$
5	$x^6+x^4+x^2+x+1$	x^8	0	$x+1$

The $i = 4$ line of Table 9.3 shows that $\gcd(a(x), b(x)) = 1$ (which is obvious anyway), and with Property D from Table 9.2 yields the equation $(x^5+x^4+x^3+x^2)a(x) + (x^7+x^6+x^3+x+1)b(x) = 1$. This example is continued in Example 9.4. ∎

We now focus our attention on Property D in Table 9.2, which can be rewritten as

$$v_i(x)b(x) \equiv r_i(x) \pmod{a(x)}. \tag{9.44}$$

Using Property F and the fact that $\deg r_{i-1} \geq \deg r_i$, we get the estimate

$$\deg v_i + \deg r_i < \deg a. \tag{9.45}$$

The main result of this section (Theorem 9.4) is a kind of converse to (9.44) and (9.45). We begin with a lemma.

`Lemma 9.2.` *Suppose Euclid's algorithm, as described above, is applied to the two polynomials $a(x)$ and $b(x)$. Given two integers $\mu \geq 0$ and $\nu \geq 0$ with $\mu + \nu = \deg a - 1$, there exists a unique index j, $0 \leq j \leq n$, such that:*

$$\deg(v_j) \leq \mu, \tag{9.46}$$
$$\deg(r_j) \leq \nu. \tag{9.47}$$

`Proof:` Recall that $\deg r_i$ is a strictly decreasing function of i until $r_n = \gcd(a, b)$, and define the index j uniquely by requiring

$$\deg r_{j-1} \geq \nu + 1, \tag{9.48}$$
$$\deg r_j \leq \nu. \tag{9.49}$$

Then by Property F we also have

$$\deg v_j \leq \mu, \tag{9.50}$$
$$\deg v_{j+1} \geq \mu + 1 \tag{9.51}$$

Equations (9.49) and (9.50) show the existence of an index j satisfying (9.46) and (9.47); Eqs. (9.48) and (9.51) show uniqueness. ∎

The following theorem is the main result of this section.

`Theorem 9.5.` *Suppose $a(x)$, $b(x)$, $v(x)$ and $r(x)$ are nonzero polynomials satisfying*

$$v(x)b(x) \equiv r(x) \pmod{a(x)}, \tag{9.52}$$
$$\deg v(x) + \deg r(x) < \deg a(x). \tag{9.53}$$

Suppose further that $v_j(x)$ and $r_j(x)$, $j = -1, 0, \ldots, n+1$, are the sequences of polynomials produced when Euclid's algorithm is applied to the pair $(a(x), b(x))$. Then there exists a unique index j, $0 \le j \le n$, and a polynomial $\lambda(x)$ such that

$$v(x) = \lambda(x)v_j(x), \tag{9.54}$$
$$r(x) = \lambda(x)r_j(x). \tag{9.55}$$

`Proof:` Let j be the index satisfying (9.46) and (9.47) with $\nu = \deg r$, $\mu = \deg a - \deg r - 1$. Thus from (9.53) $\deg(v(x)) \le \mu$. Then according to (9.51) and (9.48), $\deg v_{j+1} \ge \mu + 1 \ge \deg v + 1$, and $\deg r_{j-1} \ge \nu + 1 = \deg r + 1$. Hence if there is an index such that (9.54) and (9.55) hold, it must be unique.

Now rewrite Property D and Eq. (9.52) as follows:

$$u_j a + v_j b = r_j, \tag{9.56}$$
$$ua + vb = r, \tag{9.57}$$

where u is some unspecified polynomial. Multiply (9.56) by v and (9.57) by v_j:

$$u_j va + v_j vb = r_j v, \tag{9.58}$$
$$uv_j a + vv_j b = rv_j. \tag{9.59}$$

Together (9.58) and (9.59) imply $r_j v \equiv rv_j \pmod{a}$. But by (9.47) and (9.53), $\deg(r_j v) = \deg r_j + \deg v \le \nu + \mu < \deg a$. Similarly, by (9.46) and (9.53), $\deg(rv_j) = \deg r + \deg v_j \le \nu + \mu < \deg a$. It follows that $r_j v = rv_j$. This fact, combined with (9.58) and (9.59), implies that $u_j v = uv_j$. But since Property C guarantees that u_j and v_j are relatively prime, this means that

$$u(x) = \lambda(x)u_j(x),$$
$$v(x) = \lambda(x)v_j(x),$$

for some polynomial $\lambda(x)$. Then Equation (9.57) becomes $\lambda u_j a + \lambda v_j b = r$; comparing this to Eq. (9.58), we conclude that $r(x) = \lambda(x)r_j(x)$. ∎

The results of this section will be used constantly in our forthcoming discussion of decoding algorithms for BCH and Reed-Solomon codes. To facilitate these discussions, we now introduce the algorithmic procedure "`Euclid`$(a(x), b(x), \mu, \nu)$".

`Definition.` If $(a(x), b(x))$ is a pair of nonzero polynomials with $\deg a(x) \ge \deg b(x)$, and if (μ, ν) is a pair of nonnegative integers such that $\mu + \nu = \deg a(x) - 1$, `Euclid`$(a(x), b(x), \mu, \nu)$ is the procedure that returns the unique pair of polynomials $(v_j(x), r_j(x))$ with $\deg v_j(x) \le \mu$ and $\deg r_j(x) \le \nu$, when Euclid's algorithm is applied to the pair $(a(x), b(x))$.

The following theorem summarizes the results of this section.

`Theorem 9.6.` *Suppose $v(x)$ and $r(x)$ are nonzero polynomials satisfying*

$$v(x)b(x) \equiv r(x) \pmod{a(x)} \tag{9.60}$$
$$\deg v(x) \le \mu \tag{9.61}$$
$$\deg r(x) \le \mu, \tag{9.62}$$

where μ and ν are nonnegative integers such that $\mu + \nu = \deg r(x) - 1$. Then if $(v_j(x), r_j(x))$ is the pair of polynomials returned by `Euclid`$(a(x), b(x), \mu, \nu)$*, there is a polynomial $\lambda(x)$ such that*

$$v(x) = \lambda(x)v_j(x) \tag{9.63}$$
$$r(x) = \lambda(x)r_j(x) \tag{9.64}$$

`Proof:` Theorem 9.4 guarantees that there exists a unique index j such that (9.63) and (9.64) hold. Furthermore the procedure `Euclid`$(a(x), b(x), \mu, \nu)$ must return this pair, since by (9.63) and (9.64), $\deg v_j(x) \le \deg v(x) \le \mu$ and $\deg r_j(x) \le \deg r(x) \le \nu$. ∎

Example 9.4. Let $a(x) = x^8$, $b(x) = x^6 + x^4 + x^2 + x + 1$, $F = GF(2)$, as in Example 9.5. Using Table 9.2, we can tabulate the output of `Euclid` for the eight possible pairs (μ, ν):

(μ, ν)	$Euclid(x^8, x^6 + x^4 + x^2 + x + 1, \mu, \nu)$
$(0, 7)$	$(1, x^6 + x^4 + x^2 + x + 1)$
$(1, 6)$	$(1, x^6 + x^4 + x^2 + x + 1)$
$(2, 5)$	$(x^2 + 1, x^3 + x + 1)$
$(3, 4)$	$(x^2 + 1, x^3 + x + 1)$
$(4, 3)$	$(x^2 + 1, x^3 + x + 1)$
$(5, 2)$	$(x^5 + x^3 + x^2, x^2)$
$(6, 1)$	$(x^6 + x^4 + x^3 + x^2 + 1, x + 1)$
$(7, 0)$	$(x^7 + x^6 + x^3 + x + 1, 1)$

Now suppose we wished to "solve" the congruence $x^6 + x^4 + x^2 + x + 1)\sigma(x) \equiv \omega(x) \pmod{x^8}$, subject to the restriction that $\deg \sigma(x) \leq 3$, $\deg \omega(x) \leq 4$. Accordingg to Theorem 9.5, we invoke `Euclid`$(x^8, x^6 + x^4 + x^2 + x + 1, 4, 3)$ which by the above table returns the pair $(x^2 + 1, x^3 + x + 1)$, so that all solutions to the given problem are of the form $\sigma(x) = \lambda(x)(x^2 + 1)$, $\omega(x) = \lambda(x)(x^3 + x + 1)$, with $\deg \lambda(x) \leq 1$. If we further required $\gcd(\sigma(x), \omega(x)) = 1$, then the *only* solution would be $\sigma(x) = x^2 + 1$, $\omega(x) = x^3 + x + 1$. ∎

At this point the application of Theorem 9.4 to the problem of solving the key equation for BCH codes should be apparent. In any event we spell it out in the next section.

9.5. Decoding BCH Codes, Part II: The Algorithms.

Let us recapitulate the BCH decoding problem, which we abandoned temporarily at the end of Section 9.3. We are given a received vector $\mathbf{R} = (R_0, R_1, \ldots, R_{n-1})$, which is a noisy version of an unknown codeword $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ from the t-error connecting BCH code defined by (9.7), i.e., $\mathbf{R} = \mathbf{C} + \mathbf{E}$, where $\mathbf{E}$ is the error pattern. Our goal is to recover $\mathbf{C}$ from $\mathbf{R}$. The first step in the decoding process is to compute the *syndrome polynomial* $S(x)$, defined by

$$S(x) = S_1 + S_2 + \cdots + S_{2t}x^{2t-1}, \tag{9.65}$$

where $S_j = \Sigma_{i=0}^{n-1} R_i \alpha^{ij}$, for $j = 1, 2, \ldots, 2t$. We saw at the end of Section 9.3 that $S(x) = \widehat{V}(x) \bmod x^{2t}$, where $\widehat{V}(x)$ is the generating function for the Fourier transform of the vector $\mathbf{V}$ defined in (9.36), so that the key equation (9.37) becomes

$$\sigma(x)S(x) \equiv \omega(x) \pmod{x^{2t}}, \tag{9.66}$$

where $\sigma(x)$ is the error locator polynomial and $\omega(x)$ is the error evaluator polynomial.

The next step in the decoding process is to use Euclid's algorithm, and in particular the procedure `Euclid`$(a(x), b(x), \mu, \nu)$ defined in Section 9.4, to solve the key equation for $\sigma(x)$ and $\omega(x)$. This is possible, since if the number of errors that actually occurred is $\leq t$, then by (9.20) and (9.22),

$$\deg \sigma(x) \leq t$$

$$\deg \omega(x) \leq t - 1,$$

and by Corollary 9.1, $\gcd(\sigma(x), \omega(x)) = 1$. Thus the hypotheses of Theorem 9.5 are met with $a(x) = x^{2t}$, $b(x) = S(x)$, $v(x) = \sigma(x)$, $r(x) = \omega(x)$, $\mu = t$, $\nu = t-1$, so that if the procedure `Euclid`$(x^{2t}, S(x), t, t-1)$ is called it will return the polynomial pair $(v(x), r(x))$ where $v(x) = \lambda\sigma(x), r(x) = \lambda\omega(x)$, and λ is a nonzero scalar. The scalar λ can be determined by the fact that $\sigma(0) = 1$ (see (9.20)), i.e, $\lambda = v(0)^{-1}$, and so

$$\sigma(x) = v(x)/v(0)$$

$$\omega(x) = r(x)/v(0).$$

The final step in the decoding algorithm is to use $\sigma(x)$ and $\omega(x)$ to determine the error pattern $\mathbf{E} = (E_0, E_1, \ldots, E_{n-1})$, and hence the corrected codeword $\mathbf{C} = \mathbf{R} - \mathbf{E}$. As we observed at the end of Section 9.3, there are two ways to do this, which we shall call the *time-domain approach* and the *frequency-domain approach*.

The time domain approach is based on the fact that

$$\sigma(x) = \prod_{i \in I} (1 - \alpha^i x)$$

where I is the *error locator set*, i.e. $I = \{i : E_i \neq 0\}$ (see (9.20) and (9.36)). Thus in order to find the error locations, one needs to find the *reciprocals of the roots of the equation* $\sigma(x) = 0$. Since there are only n possibilities for the roots, viz., $1, \alpha^{-1}, \alpha^{-2}, \ldots, \alpha^{-(n-1)}$, a simple "trial and error" algorithm can be used to find $\mathbf{E}$. Thus the so-called "time-domain completion" can be described by the following pseudocode fragment. It takes as input $\sigma(x)$ and produces the error vector $(E_0, E_1, \ldots, E_{n-1})$.

```
/* Time Domain Completion */
{
  for (i = 0 to n - 1)
  {
    if (σ(α^{-i}) == 0)
      E_i = 1;
    else
      E_i = 0;
  }
}
```

A complete decoding algorithm using the time domain completion is shown in Figure 9.1. Note that the error evaluator polynomial $\omega(x)$ is not needed — its significance will become apparent only when we consider *Reed-Solomon codes* in the next section.

The *frequency domain approach* is based on Corollary 9.2, which says that the components of $\widehat{V} = (\widehat{V}_0, \ldots, \widehat{V}_{n-1})$ can be computed recursively, via the formula $\widehat{V}_j = \sum_{i=1}^{d} \sigma_i \widehat{V}_{j-i}$, where $\sigma(x) = 1 + \sigma_1 x + \cdots + \sigma_d x^d$, provided at least d "initial values" of the vector $\widehat{V}$ are known. Since the syndrome provides $2t$ components of $\widehat{V}$ viz., $\widehat{V}_1$, $\widehat{V}_2$, $\cdots$, $\widehat{V}_{2t}$, and since $\texttt{Euclid}(x^{2t}, s(x), t, t-1)$ is guaranteed to return a polynomial $v(x)$ of degree $\leq t$, the syndrome values $S_1, S_2, \ldots, S_{2t}$ are more than enough to get the recursion started, so that the following "frequency domain completion" will successfully calculate the error vector $\mathbf{E}$:

```
/* Frequency Domain Completion */
{
  for (j = 2t + 1 to n)
    S_j = Σ_{i=1}^{d} σ_i S_{j-i};)
  for (i= 0 to n-1)
    E_i = Σ_{j=0}^{n-1} S_j α^{-ij};
}
```

A complete decoding algorithm using the frequency domain completion is shown in Figure 9.2.

```
/* ''Time Domain'' BCH Decoding Algorithm */
{
  for (j = 1 to 2t)
    S_j = Σ_{i=0}^{n-1} R_i α^{ij};
  S(x) = S_1 + S_2 x + ··· + S_{2t} x^{2t-1};
  if (S(x) ==0)
    print ''no errors occurred'';
  else
  {
    Euclid (x^{2t}, S(x), t, t − 1);
    σ(x) = v(x)/v(0);
    for (i = 0 to n-1)
    {
      if (σ(α^{-i}) == 0)
        E_i = 1;
      else
        E_i = 0;
    }
    for (i = 0 to n-1)
      Ĉ_i = R_i + E_i;
    print ''corrected codeword:(Ĉ_0, Ĉ_1, ..., Ĉ_{n-1})'';
  }
}
```

Figure 9.1. A Time Domain BCH Decoding Algorithm

```
/* ‘‘Frequency Domain’’ BCH Decoding Algorithm */
{
  for (j = 1 to 2t)
    S_j = Σ_{i=0}^{n-1} R_i α^{ij};
  S(x) = S_1 + S_2 x + ··· + S_{2t} x^{2t-1};

  if (S(x) == 0)
    print ‘‘no errors occurred’’;
  else
  {
    Euclid (x^{2t}, S(x), t, t-1);
    σ(x) = v(x)/v(0);
    for (j = 2t +1 to n)
      S_j = Σ_{i=0}^{d} σ_i S_{j-i};

    for (i = 0 to n - 1)
      E_i = Σ_{j=0}^{n-1} S_j α^{-ij};
    for (i = 0 to n-1)
      Ĉ_i = R_i + E_i;
    print ‘‘corrected codeword:(Ĉ_0, Ĉ_1, ..., Ĉ_{n-1})’’;
  }
}
```

Figure 9.2. A Frequency Domain BCH Decoding Algorithm

Example 9.5. Consider the 3-error correcting BCH code of length 15, with generator polynomial $g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$ (see Example 9.1). Suppose the vector $\mathbf{R} = (110000110110101)$ is received. Then the syndrome components S_j are given by $S_j = 1 + \alpha^j + \alpha^{6j} + \alpha^{7j} + \alpha^{9j} + \alpha^{10j} + \alpha^{12j} + \alpha^{14j}$, where α is a primitive root in $GF(16)$. Using Table 9.1, together with the fact that $S_{2j} = S_j^2$ (Problem 9.17), we find that $S_1 = \alpha^{12}, S_2 = \alpha^9, S_3 = 0, S_4 = \alpha^3, S_5 = 1, S_6 = 0$, and so $S(x) = x^4 + \alpha^3 x^3 + \alpha^9 x + \alpha^{12}$. Applying Euclid's algorithm to the pair $(x^6, S(x))$, we get the following table:

i	u_i	v_i	r_i	q_i
-1	1	0	x^6	$--$
0	0	1	$x^4 + \alpha^3 x^3 + \alpha^9 x + \alpha^{12}$	$--$
1	1	$x^2 + \alpha^3 x + \alpha^6$	α^3	$x^2 + \alpha^3 x + \alpha^6$

Thus the procedure Euclid $(x^6, S(x), 3, 2)$ returns the pair $(x^2 + \alpha^3 x + \alpha^6, \alpha^3)$. Multiplying both of these polynomials by α^{-6}, we therefore find that $\sigma(x) = 1 + \alpha^{12} x + \alpha^9 x^2$, and $\omega(x) = \alpha^{12}$. If we choose the time domain completion, we find that $\sigma(\alpha^{-i}) = 0$ for $i = 2$ and 7, so that the error pattern is $\mathbf{E} = [00100001000000]$, and the corrected codeword is $\widehat{C} = [111000100110101]$. On the other hand, if we choose the frequency domain completion, we use the initial conditions $S_1 = \alpha^{12}, S_2 = \alpha^9, S_3 = 0, S_4 = \alpha^3, S_5 = 1, S_6 = 0$ and the recursion $S_j = \alpha^{12} S_{j-1} + \alpha^9 S_{j-2}$ to complete the syndrome vector, and find

$$\mathbf{S} = (S_0, S_1, \ldots, S_{15}) = (0, \alpha^{12}, \alpha^9, 0, \alpha^3, 1, 0, \alpha^9, \alpha^6, 0, 1, \alpha^{12}, 0, \alpha^6, \alpha^3).$$

Performing an inverse DFT on the vector $\mathbf{S}$ we find that $\mathbf{E} = [00100001000000]$, and $\widehat{C} = [111000100110101]$ as before. ∎

The algorithms in Figures 9.1 and 9.2 will work perfectly if the number of errors that occurs is no more than t. If, however, *more* than t errors occur, certain problems can arise. For example, the procedure "Euclid $(x^{2t}, S(x), t, t-1)$" could return a polynomial $v(x)$ with $v(0) = 0$, thereby causing a division by 0 in the step "$\sigma(x) = v(x)/v(0)$". Also, the decoder output $\widehat{C} = (\widehat{C}_0, \widehat{C}_1, \ldots, \widehat{C}_{n-1})$ may turn out not to be a codeword. Therefore in any practical implementation of the decoding algorithms, it will be necessary to test for these abnormal conditions, and print a warning, like "more than t errors" if they occur.

9.6. Reed-Solomon Codes.

In the first five sections of this chapter we have developed an elaborate theory for BCH codes. They are multiple error-correcting linear codes over the binary field GF(2), whose decoding algorithm requires computations in the larger field GF(2^m). Thus for BCH codes there are *two* fields of interest: the codeword *symbol field* $GF(2)$, and the decoder's *computation field* $GF(2^m)$.

It turns out that almost the same theory can be used to develop another class of codes, the *Reed-Solomon* codes (RS codes for short). The main *theoretical* difference between RS codes and BCH codes is that for RS codes, the symbol field and the computation field are the same. The main *practical* difference between the two classes of codes is that RS codes lend themselves naturally to the transmission of information characters other than bits. In this section we will define and study Reed-Solomon codes.

Thus let F be any field which contains a primitive nth root of unity α.* If r is a fixed integer between 0 and n, the set of all vectors $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ with components in F such that

$$\sum_{i=0}^{n-1} C_i \alpha^{ij} = 0, \quad \text{for} \quad j = 1, 2, \ldots, r \tag{9.67}$$

is called a *Reed-Solomon code* of *length* n and *redundancy* r over F. The vectors $\mathbf{C}$ belonging to the code are called its *codewords*. The following theorem gives the basic facts about RS codes.

`Theorem 9.7.` *The code defined by (9.67) is an $(n, n-r)$ cyclic code over F with generator polynomial $g(x) = \prod_{j=1}^{r}(x - \alpha^j)$, and minimum distance $d_{\min} = r + 1$.*

`Proof:` Let $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ be an arbitrary vector of length n over F and let $C(x) = C_0 + C_1 x + \cdots + C_{n-1}x^{n-1}$ be corresponding the generating function. Then (9.67) says that $\mathbf{C}$ is a codeword if and only if $C(\alpha^j) = 0$, for $j = 1, 2, \ldots, r$, which is the same as saying that $C(x)$ is a multiple of $g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^r)$. But since $x^n - 1 = \prod_{j=1}^{n}(x - \alpha^j)$, it follows that $g(x)$ is a divisor of $x^n - 1$, and so by Theorem 8.3b the code is an $(n, n-r)$ cyclic code with generator polynomial $g(x)$. To prove the assertion about $d_{\min}$, observe that (9.67) says that if $\widehat{C} = (\widehat{C}_0, \widehat{C}_1, \ldots, \widehat{C}_{n-1})$ is the DFT of a codeword, then $\widehat{C}_1 = \widehat{C}_2 = \cdots = \widehat{C}_r = 0$ (cf. Eq. (9.11)). Thus by the BCH argument (Theorem 9.3), the weight of any nonzero codeword is $\geq r + 1$. On the other hand, the generator polynomial $g(x) = x^r + g_{r-1}x^{r-1} + \cdots + g_0$, when viewed as a codeword, has weight $\leq r + 1$. Thus $d_{\min} = r + 1$ as asserted. ∎

Example 9.6. Consider the (7,3) Reed-Solomon code over GF(8). If α is a primitive root in GF(8) satisfying $\alpha^3 = \alpha + 1$, the generator polynomial for the code is $g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3$. If $g(x)$ is viewed as a codeword, it is $[\alpha^3, \alpha, 1, \alpha^3, 1, 0, 0]$, which is of weight 5, the minimum weight of the code. ∎

We note that the $(7, 3)$ RS code over $GF(8)$ in Example 9.6 has $d_{\min} = 5$, whereas the (7,3) code over GF(2) given in Example 8.2 (and elsewhere in Chapter 8) has only $d_{\min} = 4$. The following theorem shows that for a given n and k, RS codes have the largest possible $d_{\min}$, independent of the field F.

`Theorem 9.8.` *(The Singleton Bound) If C is an (n, k) linear code over a field F, then $d_{\min} \leq n - k + 1$.*

`Proof:` We begin by recalling that if T is a linear transformation mapping a finite-dimensional vector space U to another vector space V, then

$$\texttt{rank}(T) + \texttt{nullity}(T) = \texttt{dim}(U) \tag{9.68}$$

We apply this to the linear transformation T mapping the code C to the space F^{k-1} by projecting each codeword onto the first $k-1$ coordinates:

$$T(C_0, C_1, \ldots, C_{n-1}) = (C_0, C_1, \ldots, C_{k-2}).$$

* In almost, but not all, applications, the field F will be $GF(2^m)$ for some $m \geq 1$. However, the underlying theory goes through equally well for any field, finite or not, and we shall make no unnecessary restrictions in F.

We know that $\texttt{rank}(T) \le k-1$, since the image F^{k-1} has dimension $k-1$. Also, $\dim(C) = k$ by assumption. Thus (9.68) implies that $\texttt{nullity}(T) \ge 1$. Thus there exists at least one nonzero codeword $\mathbf{C}$ such that $T(\mathbf{C}) = 0$. Such a codeword has at least $k-1$ zero components, and so has weight at most $n-k+1$. ∎

Theorem 9.8 says that $d_{\min} \le n-k+1$ for any (n,k) linear code. On the other hand, Theorem 9.7 says that $d_{\min} = n-k+1$ for any (n,k) Reed-Solomon code, and so Reed-Solomon codes are *optimal* in the sense of having the largest possible minimum distance for a given length and dimension. There is a special name given to linear codes with $d_{\min} = n-k+1$; they are called *maximum distance separable (MDS)* codes. (Some other MDS codes are described in Problems 9.23–25.) All MDS codes share some very interesting mathematical properties; among the most interesting is the following, called the *interpolation property* of MDS codes.

`Theorem 9.9.` *Let C be an (n,k) MDS code over the field F, and let $I \subseteq \{0,1,\ldots,n-1\}$ be any subset of k coordinate positions. Then for any set $\{\alpha_i : i \in I\}$ of k elements from F, there exists a unique codeword $\mathbf{C}$ such that $C_i = \alpha_i$ for all $i \in I$.*

`Proof:` We consider the linear transformation P_I mapping the code C to F^k by *projecting* each codeword onto the index set I; i.e., $P_I(C_0, C_1, \ldots, C_{n-1}) = (C_{i_1}, C_{i_2}, \ldots, C_{i_k})$, where $I = \{i_1, i_2, \ldots, i_k\}$. Applying (9.68), which in this case says that $\texttt{rank}(P_I) + \texttt{nullity}(P_I) = \texttt{dim}(C)$ we see that $\texttt{dim}(C) = k$, since C is a k-dimensional code. Also, $\texttt{nullity}(P_I) = 0$, since if there were a nonzero codeword $\mathbf{C}$ with $P_I(\mathbf{C}) = 0$, that codeword would have weight at most $n-k$, contradicting the fact that $\mathbf{C}$ is an MDS code. Hence by (9.68) $\texttt{rank}(P_I) = k$, and so the mapping $P_I : C \to F^k$ is nonsingular, i.e. one-to-one and onto. Thus every vector in F^k appears exactly once as the projection of a codeword onto I, which is what the theorem promises. ∎

We summarize the result of Theorem 9.9 by saying that any subset of k coordinate positions of a k-dimensional MDS code is an *information set* (see also Problem 7.13). The proof we have given is short but nonconstructive; however, for RS codes there is an efficient *interpolation algorithm*, which is closely related to the Lagrange interpolation formula of numerical analysis. The next theorem spells this out.

`Theorem 9.10.` *Consider the (n,k) Reed-Solomon code over the field F defined by (9.67), where $k = n-r$. There is a one-to-one correspondence between the codewords $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ of this code, and the set of all polynomials $P(x) = P_0 + P_1 x + \cdots + P_{k-1}x^{k-1}$ of degree $k-1$ or less over F, given by*

$$C_i = \alpha^{-i(r+1)} P(\alpha^{-i}).$$

Thus apart from the scaling factors $\alpha^{-i(r+1)}$, the components of a given RS codeword are the values of a certain $(k-1)$st degree polynomial.

`Proof:` Let $\mathbf{C} = [C_1, \ldots, C_{n-1}]$ be a fixed codeword. We define a "twisted" version of $\mathbf{C}$, called $\mathbf{D} = [D_1, \ldots, D_{n-1}]$, by

$$D_i = \alpha^{-i(r+1)} C_i, \qquad \text{for } i = 0, 1, \ldots, n-1. \tag{9.69}$$

Since by (9.67) we have $\widehat{C}_1 = \widehat{C}_2 = \cdots = \widehat{C}_r = 0$, it follows from (9.17) and (9.18) that $\widehat{D}_{n-r} = \cdots = \widehat{D}_{n-1} = 0$. Thus the DFT polynomial for $\mathbf{D}$, denoted by $\widehat{D}(x)$, is a polynomial of degree $n-r-1 = k-1$ or less:

$$\widehat{D}(x) = \widehat{D}_0 + \widehat{D}_1 x + \cdots + \widehat{D}_{k-1} x^{k-1}.$$

Let us define the polynomial $P(x)$ as follows:

$$P(x) = \frac{1}{n} \widehat{D}(x).$$

Then by (9.16) we have $D_i = P(\alpha^{-i})$, for $i = 0, 1, \ldots, n-1$. Combining this with (9.69), we obtain $C_i = \alpha^{-i(r+1)} P(\alpha^{-i})$, which is what we wanted. ∎

The following Example illustrates Theorem 9.10.

Example 9.7. Consider the (7,3) RS code described in Example 9.6. According to Theorem 9.9, there is a unique codeword $\mathbf{C}$ such that $C_1 = \alpha^3, C_4 = \alpha$, and $C_6 = \alpha^4$. Let us construct this codeword.

We begin by observing that if $I = \{1, 4, 6\}$, Theorem 9.9 guarantees, in essence, the existence of a 3 x 7 generator matrix for C of the form

$$G_{146} = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{pmatrix} * & 1 & * & * & 0 & * & 0 \\ * & 0 & * & * & 1 & * & 0 \\ * & 0 & * & * & 0 & * & 1 \end{pmatrix} \end{array}$$

where the *'s are unknown elements of GF(8) which must be determined. Once G_{146} is known, the desired codeword $\mathbf{C}$ is given by $\mathbf{C} = [\alpha^3, \alpha, \alpha^4] \cdot G_{146}$. So let's construct the three rows of G_{146}, which we shall call C_1, C_4, and C_6.

By Theorem 9.9, any codeword $\mathbf{C}$ from the (7,3) RS code can be represented as $C_i = \alpha^{-5i} P(\alpha^{-i})$, where $P(x) = P_0 + P_1 x + P_2 x^2$ is a polynomial of degree 2 or less. Thus for example, if $P_1(x)$ denotes the polynomial corresponding to the first row $\mathbf{C}_1$ of G_{146}, we have

$$P_1(\alpha^{-1}) = \alpha^5, P_1(\alpha^{-4}) = 0, P_1(\alpha^{-6}) = 0. \tag{9.70}$$

It follows from the conditions $P_1(\alpha^{-4}) = P_1(\alpha^{-6}) = 0$ in (9.65) that $P(x) = A(1 + \alpha^4 x)(1 + \alpha^6 x)$ for some constant A, which can be determined by the condition $P_1(\alpha^{-1}) = \alpha^5$. Indeed $P_1(\alpha^{-1}) = \alpha^5$ implies $A(1 + \alpha^3)(1 + \alpha^5) = \alpha^5$, i.e., $A = \alpha^5/(1 + \alpha^3)(1 + \alpha^5) = 1$. Thus $P_1(x) = (1 + \alpha^4 x)(1 + \alpha^6 x)$, and so

$$\begin{aligned} \mathbf{C}_1 &= [P_1(1), \alpha^2 P_1(\alpha^{-1}), \alpha^4 P_1(\alpha^{-2}), \alpha^6 P_1(\alpha^{-3}), \alpha^1 P_1(\alpha^{-4}), \alpha^3 P_1(\alpha^{-5}), \alpha^5 P_1(\alpha^{-6})] \\ &= [1, 1, \alpha, \alpha^3, 0, \alpha, 0] \end{aligned}$$

Similarly, if $P_4(x)$ and $P_6(x)$ denote the quadratic polynomials corresponding to the rows C_4 and C_6 of the generator matrix G_{146}, then we find that $P_4(x) = \alpha^2(1 + \alpha x)(1 + \alpha^6 x)$ and $P_6(x) = \alpha^6(1 + \alpha x)(1 + \alpha^4 x)$. Thus we compute

$$\begin{aligned} \mathbf{C}_4 &= [1, 0, \alpha^6, \alpha^6, 1, \alpha^2, 0] \\ \mathbf{C}_6 &= [1, 0, \alpha^4, \alpha^5, 0, \alpha^5, 1]. \end{aligned}$$

Combining $\mathbf{C}_1, \mathbf{C}_4$, and $\mathbf{C}_6$, we find that the generator matrix G_{146} is

$$G_{146} = \begin{pmatrix} 1 & 1 & \alpha & \alpha^3 & 0 & \alpha & 0 \\ 1 & 0 & \alpha^6 & \alpha^6 & 1 & \alpha^2 & 0 \\ 1 & 0 & \alpha^4 & \alpha^5 & 0 & \alpha^5 & 1 \end{pmatrix},$$

and so, finally, the unique codeword $\mathbf{C}$ with $C_1 = \alpha^3, C_4 = \alpha, C_6 = \alpha^4$ is

$$\mathbf{C} = [\alpha^3, \alpha, \alpha^4] \cdot G_{146} = [\alpha^5, \alpha^3, \alpha^6, 0, \alpha, 1, \alpha^4].$$

∎

This concludes our theoretical discussion of RS codes; now let's consider the practical issues of *encoding* and *decoding* them.

Since by Theorem 9.7, an (n, k) RS code is cyclic, it can be encoded using the shift-register techniques developed in Chapter 8. In particular, the general encoding circuit of Figure 8.5(a) can be used. However, since a RS code is defined over an arbitrary field F – which in practice will never be the binary field $GF(2)$ (Problem 9.24) – the three basic components (flip-flops, adders, and multipliers), will typically not be "off the shelf" items. Although the design of these components over the important fields $GF(2^m)$ is an important and interesting topic, it is beyond the scope of this book, and we will conclude our discussion of RS encoders

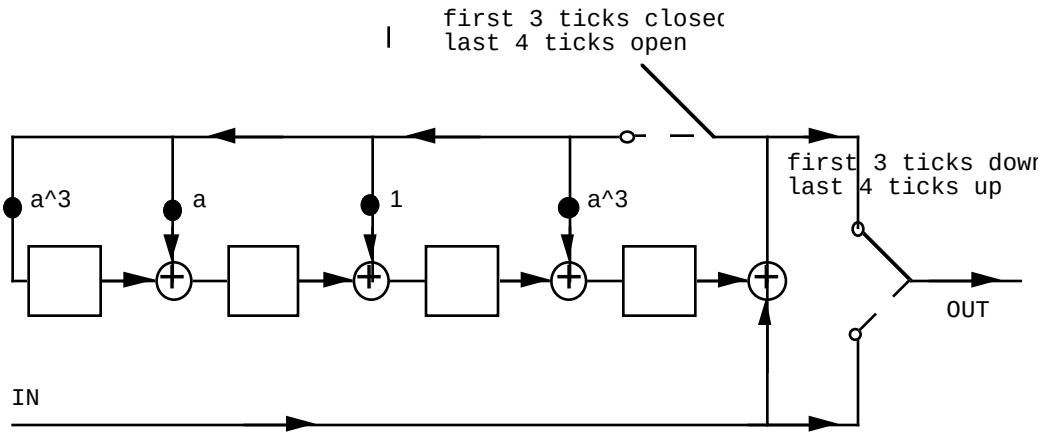

Figure 9.3. A systematic shift-register encoder for the (7,3) RS code over GF(8) with $g(x) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3$.

with Figure 9.3, which shows a systematic shift-register encoder for the (7,3) RS code over $GF(8)$ with $g(x) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3$ (see Examples 9.8 and 9.9).

We turn now to the problem of *decoding* RS codes, which turns out to be quite similar to the decoding of BCH codes. In view of the similarity of their definitions (compare (9.7)) with (9.67)), this should not be surprising.

Let us begin by formally stating the RS decoding problem. We are given a received vector $\mathbf{R} = (R_0, R_1, \ldots, R_{n-1})$, which is a noisy version of an unknown codeword $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ from the (n, k) RS code defined by (9.67), i.e., $\mathbf{R} = \mathbf{C} + \mathbf{E}$, where $\mathbf{E}$ is the error pattern. Since by Theorem 9.7, $d_{\min} = r + 1$, we cannot hope to correctly identify $\mathbf{C}$ unless $\mathrm{wt}(\mathbf{E}) \leq \lfloor r/2 \rfloor$, and so for the rest of the discussion we shall let $t = \lfloor r/2 \rfloor$, and assume that $\mathrm{wt}(\mathbf{E}) \leq t$.

The first step in the decoding process is to compute the *syndrome polynomial*

$$S(x) = S_1 + S_2 x + \cdots + S_r x^r, \tag{9.71}$$

where $S_j = \sum_{i=0}^{n-1} R_i \alpha^{ij}$, for $j = 1, 2, \ldots, r$. By the results of Section 9.3, if we define the "twisted error pattern" by

$$\mathbf{V} = (E_0, E_1\alpha, E_2\alpha^2, \ldots, E_{n-1}\alpha^{n-1}),$$

then $S(x) = \widehat{V}(x) \bmod x^r$, and the key equation (9.23), reduced $\bmod x^r$, becomes

$$\sigma(x)S(x) \equiv w(x) (\bmod\, x^r),$$

where $\sigma(x)$ is the locator polynomial, and $w(x)$ is the evaluator polynomial, for the vector $\mathbf{V}$.

At this point the decoding problem is almost exactly the same as it was for BCH codes as described in Section 9.5. In particular, if the procedure `Euclid`$(x^r, S(x), t, t-1)$ is called, it will return the pair of polynomials $(v(x), r(x))$, where $v(x) = \lambda\sigma(x)$ and $r(x) = \lambda\omega(x)$, for some nonzero constant λ.

The final step in the decoding algorithm is to use $\sigma(x)$ and $\omega(x)$ to determine the error pattern $\mathbf{E} = (E_0, E_1, \ldots, E_{n-1})$, and hence the original codeword $\mathbf{C} = \mathbf{R} - \mathbf{E}$. As with BCH codes, there are two essentially different ways to do this, the *time domain approach* and the *frequency domain approach*.

The *time domain approach* for RS decoding is similar to the time domain approach for BCH decoding, with one important exception. For BCH codes, when the errors are located, their values are immediately known. This is because BCH codes are binary, so that $E_i = 0$ or 1 for all i. Thus if there is an error in

position i, i.e., $E_i \neq 0$, then necessarily $E_i = 1$. However, for RS codes, the E_i's lie in an arbitrary field F, so that simply knowing that $E_i \neq 0$ is not enough to evaluate E_i. In order to evaluate an error whose location is known, we use Corollary 9.1, which says that if $E_i \neq 0$, i.e., $\sigma(\alpha^{-i}) = 0$, then $V_i = \alpha^i E_i = -\alpha^i \omega(\alpha^{-i})/\sigma'(\alpha^{-i})$, i.e.,

$$E_i = -\frac{\omega(\alpha^{-i})}{\sigma'(\alpha^{-i})}. \tag{9.72}$$

Thus the time domain completion of the RS decoding algorithm can be written as follows:

```
/* Time Domain Completion */
{
  for (i = 0 to n - 1)
  {
      if (σ(α^{-i}) == 0)
        E_i = -ω(α^{-i})/σ'(α^{-i});
      else
        E_i = 0;
  }
}
```

A complete time domain decoding algorithm for RS codes is shown in Figure 9.4.

```
/*Time Domain RS Decoding Algorithm*/
{
  for (j = 1 to r)
    S_j = Σ_{i=0}^{n-1} R_i α^{ij};
  S(x) = S_1 + S_2 x + ··· + S_r x^{r-1};
  if (S(x) = = 0)
    print ‘‘no errors occurred’’;
  else
  {
    Euclid(x^r, S(x), t, t - 1);
    σ(x) = v(x)/v(0);
    ω(x) = r(x)/v(0);
    for (i = 0 to n - 1)
    {
      if (σ(α^{-i}) == 0
        E_i = -ω(α^{-i})/σ'(α^{-i});
      else
        E_i = 0;
    }
    for (i = 0 to n - 1)
      Ĉ_i = R_i - E_i;
    print ‘‘corrected codeword:  [Ĉ_0, Ĉ_1, ..., Ĉ_{n-1}]’’;
  }
}
```

Figure 9.4. A Time Domain RS Decoding Algorithm

The *frequency domain approach* to RS decoding is nearly identical to the frequency domain approach to BCH decoding, since the idea of recursive completion of the error vector works for an arbitrary field F. Here is a pseudocode listing for a frequency domain completion.

```
/*Frequency Domain Completion*/
{
  for (j = r + 1 to n)
    S_{j mod n} = -Σ_{i=1}^{d} σ_i S_{j-i};
  for (i = 0 to n - 1)
    E_i = (1/n) Σ_{j=0}^{n-1} S_j α^{-ij};
}
```

A complete RS decoding algorithm using the frequency domain completion is given in Figure 9.5.

```
/*Frequency Domain RS Decoding Algorithm*/
{
  for (j = 1 to r)
    S_j = Σ_{i=0}^{n-1} R_i α^{ij};
  S(x) = S_1 + S_2 x + ··· + S_r x^{r-1};
  if (S(x) = = 0)
    print ''no errors occured'';
  else
  {
    Euclid(x^r, S(x), t, t-1);
    σ(x) = v(x)/v(0);
    ω(x) = r(x)/v(0);
    for (j = r + 1 to n)
      S_{j mod n} = -Σ_{i=0}^{d} σ_i S_{j-i};
    for (i = 0 to n -1)
      E_i = (1/n) Σ_{j=0}^{n-1} S_j α^{-ij};
    for (i = 0 to n - 1)
      Ĉ_i = R_i - E_i;
    print ''corrected codeword:  [Ĉ_0, Ĉ_1, ..., Ĉ_{n-1}]'';
  }
}
```

Figure 9.5. A Frequency Domain RS Decoding Algorithm

Example 9.8. Consider the (7,3) RS code over $GF(2^3)$ with $g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) = x^4 + \alpha^3 x^3 + x^2 + \alpha x + \alpha^3$ already considered in Examples 9.6 and 9.7. Suppose the received vector is $\mathbf{R} = (\alpha^3, \alpha, 1, \alpha^2, 0, \alpha^3, 1)$. The syndromes $S_j = \sum R_i \alpha^{ij}$ are $S_1 = \alpha^3, S_2 = \alpha^4, S_3 = \alpha^4, S_4 = 0$, so that $S(x) = \alpha^4 x^2 + \alpha^4 x + \alpha^3$. If we invoke the procedure `Euclid`$[x^4, \alpha^4 x^2 + \alpha^4 x + \alpha^3, 2, 1]$ we obtain the following table:

i	$v_i(x)$	$r_i(x)$	$q_i(x)$
-1	0	x^4	$--$
0	1	$\alpha^4 x^2 + \alpha^4 x + \alpha^3$	$--$
1	$\alpha^3 x^2 + \alpha^3 x + \alpha^5$	$x + \alpha$	$\alpha^3 x^2 + \alpha^3 x + \alpha^5$

Thus we conclude $\sigma(x) = \alpha^{-5}(\alpha^5 + \alpha^3 x + \alpha^3 x^2) = 1 + \alpha^5 x + \alpha^5 x^2$, and $\omega(x) = \alpha^{-5}(x + \alpha) = \alpha^2 x + \alpha^3$.

With the time domain approach, we find that $\sigma(\alpha^{-3}) = \sigma(\alpha^{-2}) = 0$, i.e., $\sigma(x) = (1 + \alpha^2 x)(1 + \alpha^3 x)$. Thus the error locations are $i = 2$ and $i = 3$. To evaluate these two errors, use the formula (9.72), together with the fact that $\sigma'(x) = \alpha^5$, so that $\omega(x)/\sigma'(x) = \alpha^4 x + \alpha^5$, and find

$$E_2 = \frac{\omega(\alpha^{-2})}{\sigma'(\alpha^{-2})} = \alpha^4 \cdot \alpha^{-2} + \alpha^5 = \alpha^3$$
$$E_3 = \frac{\omega(\alpha^{-3})}{\sigma'(\alpha^{-3})} = \alpha^4 \cdot \alpha^{-3} + \alpha^5 = \alpha^6.$$

Thus $\mathbf{E} = (0, 0, \alpha^3, \alpha^6, 0, 0, 0)$ and the decoder's output is $\widehat{C} = \mathbf{R} + \mathbf{E} = (\alpha^3, \alpha, \alpha, 1, 0, \alpha^3, 1)$.

With the frequency domain approach, we use the intial conditions $S_1 = \alpha^3, S_2 = \alpha^4, S_3 = \alpha^4, S_4 = 0$ and the recursion (based on the coefficients of $\sigma(x)$) $S_j = \alpha^5 S_{j-1} + \alpha^5 S_{j-2}$ to find

$$S_5 = \alpha^5 \cdot 0 + \alpha^5 \cdot \alpha^4 = \alpha^2$$
$$S_6 = \alpha^5 \cdot \alpha^2 + \alpha^5 \cdot 0 = 1$$
$$S_7 = S_0 = \alpha^5 \cdot 1 + \alpha^5 \cdot \alpha^2 = \alpha^4.$$

Thus $\mathbf{S} = (S_0, S_1, S_2, S_3, S_4, S_5, S_6) = (\alpha^4, \alpha^3, \alpha^4, \alpha^4, 0, \alpha^2, 1)$. To obtain $\mathbf{E}$, we take on inverse DFT of $\mathbf{S}$, using (9.12):

$$\mathbf{E} = \widehat{\mathbf{S}} = (0, 0, \alpha^3, \alpha^6, 0, 0, 0),$$

and now the decoding concludes as before. ∎

We conclude this section with a brief discussion of two important applications of RS codes: *burst-error correction* and *concatenated coding*.

We can illustrate the application to burst-error correction by returning to Example 9.8. There we saw the (7,3) RS code over $GF(8)$ in action, correcting two symbol errors. But instead of viewing each codeword as a 7-dimensional vector over $GF(8)$, we can expand each element of $GF(8)$ into a three-dimensional binary vector via Table 9.4 and thereby convert the codewords into 21-dimensional binary vectors. In other words, the $(7, 3)$ RS code over $GF(8)$ can be viewed as a $(21, 9)$ linear code over $GF(2)$. For example, the codeword

$$\mathbf{C} = (\alpha^3, \alpha, \alpha, 1, 0, \alpha^3, 1)$$

becomes the binary vector

$$\mathbf{C} = (011\ 010\ 010\ 001\ 000\ 011\ 001).$$

i	α^i
0	001
1	010
2	100
3	011
4	110
5	111
6	101

Table 9.4. The Field $GF(8)$ represented as powers of α, where $\alpha^3 = \alpha + 1$.

Now suppose this binary version of $\mathbf{C}$ was sent over a binary channel and suffered the following error burst of length 5:

$$\mathbf{E} = (000\ 000\ 0\overbrace{11\ 101}^{\text{error burst}}000\ 000\ 000).$$

Then the received vector would be

$$\mathbf{R} = (011\ 010\ 001\ 100\ 000\ 011\ 001),$$

which of course differs from $\mathbf{C}$ in four positions. Ordinarily it would be difficult or impossible to correct four errors in a $(21, 9)$ binary linear code (see Problem 9.33), but we can take advantage of the fact that this particular set of four errors has occured in a short burst by observing that when $\mathbf{E}$ is mapped into 7-dimensional vector from $GF(8)$,

$$\mathbf{E} = (0, 0, \alpha^3, \alpha^6, 0, 0, 0),$$

it only has weight 2! Thus if we convert $\mathbf{R}$ into a vector from $GF(8)$,

$$\mathbf{R} = (\alpha^3, \alpha, 1, \alpha^2, 0, \alpha^3, 1),$$

we can (and we already did in Example 9.8) find the error pattern and correct the errors, via the decoding algorithm of Figure 9.4 or 9.5. In this way the original RS code has become a (21,9) binary linear code which is capable of correcting many patterns of burst errors.

The generalization should be obvious: *the t-error correcting RS code of length n over $GF(q^m)$ can be implemented as a $(m(q^m - 1), m(q^m - 1 - 2t))$ linear code over $GF(q)$ which is capable of correcting any burst-error pattern that does not affect more than t of the symbols in the original $GF(q^m)$ version of the codeword.*

We come finally to the application of RS to *concatenated coding*, a subject already mentioned briefly in Chapter 6 (see p. 125). We illustrate with a numerical example.

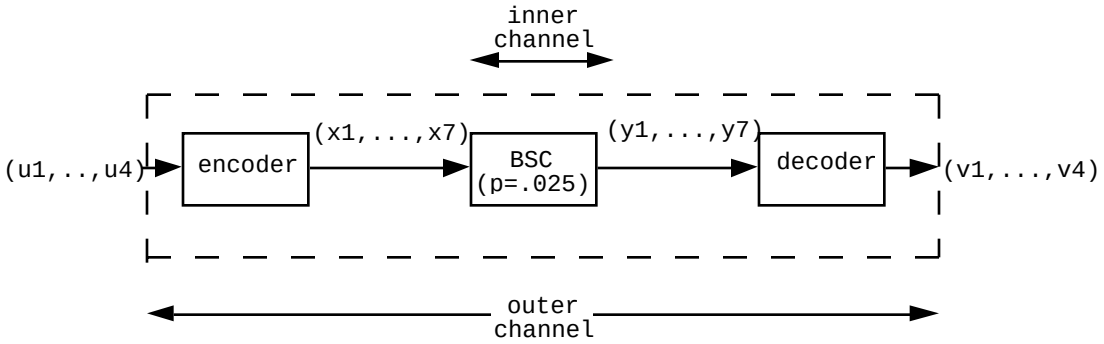

Figure 9.6. The (7,4) Hamming code on a BSC with $p = .025$.

Suppose the (7,4) binary Hamming code is being used on a BSC with raw error probability $p = .025$, as shown in Figure 9.6. In the notation of Figure 9.6, $P\{u \neq v\} = \sum_{k=2}^{7} \binom{7}{k} p^k (1-p)^{7-k} = .0121$. The idea of concatenation is to regard the "encoder-BSC-decoder" part of Figure 9.6 as one big noisy channel, called the *outer channel* (the BSC itself becomes the *inner channel*), and to design a code for it. In this example the outer channel is a DMC with 16 inputs and outputs; the results of this section suggest that we regard these inputs and outputs as elements from $GF(16)$ rather than as four-dimensional vectors over $GF(2)$. So let us now consider using the $(15, 11)$ RS code over $GF(16)$ to reduce the noise in the outer channel, as illustrated in Figure 9.7.

The RS encoder in Figure 9.7 takes 11 information symbols $\alpha = (\alpha_0, \ldots, \alpha_{10})$ from $GF(16)$ (which are really 44 bits from the original source) and produces a RS codeword $\mathbf{C} = (C_0, C_1, \ldots, C_{14})$. The outer channel then garbles $\mathbf{C}$, and it is received as $\mathbf{R} = (R_0, \ldots, R_{14})$. The RS decoder then produces an estimate $\beta = (\beta_0, \ldots, \beta_{10})$ of α, which will be equal to α if the outer channel has caused no more than two symbol

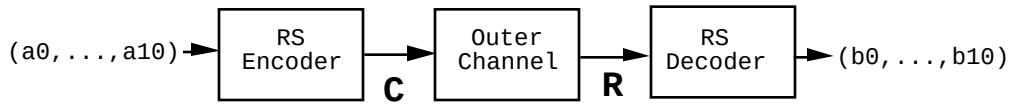

Figure 9.7. The $(15, 11)$ Reed-Solomon code being used on the outer channel of Figure 9.6.

errors. Thus if ϵ $(= 0.0121)$ denotes the probability of decoder error in Figure 9.6, the probability of decoder error in Figure 9.7 is not more than $\sum_{k=3}^{15} \binom{15}{k} \epsilon^k (1-\epsilon)^{15-k} = 0.0007$. The overall rate of the coding system depicted in Figure 9.7 is $11/15 \times 4/7 = 0.42$; indeed, the system is really just a (105,44) binary linear code which has been "factored" in a clever way. We might wish to compare this to an approximately comparable *unfactored* system, say the 11 error-correcting binary BCH code of length 127 which is a (127, 57) code. Its rate (0.45) is slightly higher and its decoder error probability (.0004) is slightly lower, but its decoding complexity is considerably larger—for the BCH code, the error-locator polynomial will typically be an 11th degree polynomial over $GF(128)$, whereas for the RS code it will be a quadratic polynomial over $GF(16)$.

The preceding example illustrates both the general idea of concatenation and the reason why RS codes are so useful in concatenated systems. *Any* coded communication system can be regarded as a noisy outer channel, as in Figure 9.8. However, for this point of view to be useful, we must be able to design an outer code capable of correcting most of the errors caused by the outer channel, which is likely to be a very complex beast, since its errors are caused by inner decoder failures. When the inner decoder fails, that is when $v \neq u$ in Figure 9.8, the symbols $v_1, \ldots, v_k$ usually bear practically no resemblance to $u_1, \ldots, u_k$. This means that errors in the outer channel tend to occur in bursts of length k. And we have already seen that RS codes are well suited to burst-error correction. This is the reason why RS codes are in widespread use as outer codes in concatenated systems.

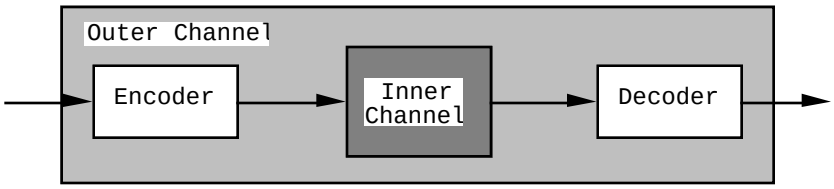

Figure 9.8. A general coded communication system, viewed as a noisy "outer" channel. (Compare to Fig. 9.6.)

9.7 Decoding When Erasures are Present

We have seen that BCH and RS codes can correct multiple errors. In this section we will see that they can also correct another class of channel flaws, called *erasures*. An erasure is simply a channel symbol which is received illegibly. For example, consider the English word *BLOCK*. If the third letter is changed from *O* to *A*, we get *BLACK*; this is an *error* in the third position. However, if the same word suffers an *erasure* in the third position, the result is *BL*CK*, where "*" is the *erasure symbol*. In practice, erasures are quite common. They can be expected to occur when the channel noise becomes unusually severe for a short time. For example, if you are trying to talk at the airport and a low-flying jet passes overhead, your conversation is *erased*. Your listeners will not mistake what you are trying to say; they will simply not be able to understand you.

In this section, we will learn something about erasure correction. We will see that in principle, an erasure is only half as hard to correct as an error (Theorem 9.11); and we will see how to modify the BCH and RS decoding algorithms in order to correct both erasures and errors.

To model a channel which can produce erasures as well as errors, we simply enlarge the underlying symbol set F to $\overline{F} = F \cup \{*\}$, where "*" is as above a special erasure symbol. The only allowed *transmitted* symbols are the elements of F, but any element in $\overline{F}$ can be *received*. The main theoretical result about simultaneous erasure and error correction follows. (Compare to Theorem 7.2.)

`Theorem 9.11.` *Let C be a code over the alphabet F with minimum distance d. Then C is capable of correcting any pattern of e_0 erasures and e_1 errors if $e_0 + 2e_1 \leq d - 1$.*

`Proof:` To prove the theorem, we first introduce the *extended Hamming distance* $\bar{d}_H(x, y)$ between symbols in $\overline{F}$:

$$\bar{d}_H(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \text{ and neither } x \text{ nor } y \text{ is "*"} \\ \frac{1}{2} & \text{if } x \neq y \text{ and one of } x \text{ and } y \text{ is "*"}. \end{cases}$$

Thus for example if $F = \{0, 1\}$ and $\overline{F} = \{0, 1, *\}$, then $\bar{d}_H(0, 1) = 1$, $\bar{d}_H(1, *) = 1/2$, $\bar{d}_H(1, 1) = 0$. We then extend the definition of $\bar{d}_H$ to vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ with components in $\overline{F}$ as follows:

$$\bar{d}_H(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \bar{d}_H(x_i, y_i).$$

With this definition, $\bar{d}_H$ becomes a *metric* on the set $\overline{F}^n$ of all n-dimensional vectors over $\overline{F}$. (See Problem 9.43), and indeed $\bar{d}_H(\mathbf{x}, \mathbf{y})$ is just the ordinary Hamming distance between $\mathbf{x}$ and $\mathbf{y}$, if no erasure symbols are involved in $\mathbf{x}$ or $\mathbf{y}$.

We next introduce a special decoding algorithm, called the *minimum distance decoding* (MDD) algorithm for the code C. When the MDD algorithm is given as input a received word $R \in \overline{F}^n$, it produces as its output a codeword C_i for which the extended Hamming distance $\bar{d}_H(C_i, R)$ is smallest. We will prove Theorem 9.11 by showing that the MDD algorithm will correct e_0 erasures and e_1 errors, if $e_0 + 2e_1 \leq d - 1$.

Thus suppose that C_i is the transmitted codeword, and that in transmission it suffers e_0 erasures and e_1 errors, with $e_0 + 2e_1 \leq d - 1$. If R is the corresponding garbled version of C_i, then $\bar{d}_H(C_i, R) = \frac{1}{2}e_0 + e_1 \leq \frac{1}{2}(d - 1)$. There can be no other codeword this close to R, since if e.g. $\bar{d}_H(C_j, R) \leq \frac{1}{2}(d - 1)$ where $j \neq i$, then by the triangle inequality

$$\begin{aligned} \bar{d}_H(C_i, C_j) &\leq \bar{d}_H(C_i, R) + \bar{d}_H(R, C_j) \\ &\leq \frac{1}{2}(d-1) + \frac{1}{2}(d-1) \quad , \\ &= d - 1 \end{aligned}$$

which contradicts the fact that the code's minimum distance is d. Therefore the distance $\bar{d}_H(C_i, R)$ is uniquely smallest for $j = i$, and the MDD algorithm will correctly indentify C_i, the actual transmitted codeword. ∎

Example 9.9. Let C be the (7,3) cyclic code from Example 8.2, with codewords

$$C_0 = 0000000$$
$$C_1 = 1011100$$
$$C_2 = 0101110$$
$$C_3 = 0010111$$
$$C_4 = 1001011$$
$$C_5 = 1100101$$
$$C_6 = 1110010$$
$$C_7 = 0111001$$

Since this code is linear, its minimum distance is the same as its minimum weight; thus $d = 4$. According to Theorem 9.11, then, this code is capable of correcting e_0 erasures and e_1 errors, provided $e_0 + 2e_1 \leq 3$. Here is a table of the allowed combinations of erasures and errors:

e_0	e_1
3	0
2	0
1	1
1	0
0	1
0	0

For example, suppose $R = [1\ 1\ 1\ 0\ *\ 0\ 1]$ is received. The MDD algorithm would make the following computations:

i	$\overline{d}_H(C_i, R)$	Erasure Positions	Error Positions
0	4.5	$\{4\}$	$\{0, 1, 2, 6\}$
1	3.5	$\{4\}$	$\{1, 3, 6\}$
2	5.5	$\{4\}$	$\{0, 2, 3, 5, 6\}$
3	3.5	$\{4\}$	$\{0, 1, 5\}$
4	4.5	$\{4\}$	$\{1, 2, 3, 5\}$
5	1.5	$\{4\}$	$\{2\}$
6	2.5	$\{4\}$	$\{5, 6\}$
7	2.5	$\{4\}$	$\{0, 3\}$

Therefore the MDD would output C_5 and conclude that R had suffered an erasure in position 4 and an error in position 2, i.e. $e_0 = 1$ and $e_1 = 1$. On the other hand if $\mathbf{y} = [*\ *\ *\ 1\ 0\ 1\ 0]$, the computation would run as follows:

i	$\overline{d}_H(C_i, R)$	Erasure Positions	Error Positions
0	3.5	$\{0, 1, 2\}$	$\{3, 5\}$
1	3.5	$\{0, 1, 2\}$	$\{4, 5\}$
2	2.5	$\{0, 1, 2\}$	$\{4\}$
3	4.5	$\{0, 1, 2\}$	$\{3, 4, 6\}$
4	2.5	$\{0, 1, 2\}$	$\{6\}$
5	5.5	$\{0, 1, 2\}$	$\{3, 4, 5, 6\}$
6	2.5	$\{0, 1, 2\}$	$\{3\}$
7	3.5	$\{0, 1, 2\}$	$\{5, 6\}$

Here the algorithm faces a three-way tie (between C_2, C_4, and C_6), but no matter which of these three it selects, it will conclude that the transmitted codeword has suffered 3 erasures and 1 error, which is beyond the code's guaranteed correction capabilities. ∎

Theorem 9.11 gives the *theoretical* erasure-and-error correction capability of a code in terms of its minimum distance, but from a *practical* standpoint the MDD algorithm used in the proof leaves much to be desired, since it is plainly impractical to compare the received word to each of the codewords unless the code is very small. Fortunately, for BCH and RS codes, there is a simple modification of the basic "errors only" decoding algorithms we have already presented in Section 9.6 (Figs. 9.4 and 9.5), which enables them to correct erasures as well as errors. In the remainder of this section, we will discuss this modification.

The erasures-and-errors decoding algorithms for BCH and RS codes, like their errors-only counterparts, are virtually identical, but for definititeness we'll consider in detail only RS codes. At the end of this section, we'll discuss the simple modifications required for BCH codes. By Theorem 9.7, the minimum distance of an (n, k) RS code is $r + 1$, where $r = n - k$, and so Theorem 9.11 implies the following.

`Theorem 9.12.` *Let C be an (n, k) RS code over a field F. Then C is capable of correcting any pattern of e_0 erasures and e_1 errors, if $e_0 + 2e_1 \le r$, where $r = n - k$.* ∎

Now let's begin our discussion of the erasures-and-errors decoding algorithm for RS codes. Suppose we are given a received vector $R = (R_0, R_1, \ldots, R_{n-1})$, which is a noisy version of an unknown codeword $C = (C_0, C_1, \ldots, C_{n-1})$, from an (n, k) RS code with generator polynomial $g(x) = (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^r)$, with $r = n - k$. We assume R has suffered e_0 erasures and e_1 errors, where $e_0 + 2e_1 \le r$. The first step in the decoding algorithm is to *store the locations of the erasures.* This is done by defining the *erasure set* I_0 as

$$I_0 = \{i : R_0 = *\}, \tag{9.73}$$

and then computing the *erasure location polynomial* $\sigma_0(x)$:

$$\sigma_0(x) = \prod_{i \in I_0} (1 - \alpha^i x). \tag{9.74}$$

(If there are no erasures, $\sigma_0(x)$ is defined to be 1.)

Once the erasure locations have been "stored" in $\sigma_0(x)$, the algorithm replaces the $*$'s in R with 0's, i.e., a new received vector $R' = (R'_0, R'_1, \ldots, R'_{n-1})$ is defined, as follows:

$$R'_i = \begin{cases} R_i & \text{if } R_i \ne * \\ 0 & \text{if } R_i = *. \end{cases} \tag{9.75}$$

The advantage of replacing the $*$'s with 0's is that unlike $*$, 0 is a legal element of the field F, and so arithmetic operations can be performed on any component of R'. The *disadvantage* of doing this is that when viewed as a garbled version of C, R' will have suffered $e_0 + e_1$ errors,* which may exceed the code's errors-only correction capability. However, as we shall see, by using the "side information" provided by the erasure locator polynomial $\sigma_0(x)$, the errors in R' can all be corrected.

With this preliminary "erasure management" completed, the decoding algorithm proceeds in a manner which is very similar to the errors-only algorithm. In particular, the next step is to compute the syndrome polynomial $S(x) = S_1 + S_2 x + \cdots + S_r x^{r-1}$, where

$$S_j = \sum_{i=0}^{n-1} R'_i \alpha^{ij}, \qquad \text{for } j = 1, 2, \ldots, r.$$

If now we define the *errors-and-erasures vector* $E' = (E'_0, E'_1, \ldots, E'_{n-1})$ as $E' = R' - C$, and the "twisted" errors-and erasures vector V by

$$V = (E'_0, E'_1\alpha, \ldots, E'_{n-1}\alpha^{n-1}), \tag{9.76}$$

then it follows by the results of Section 9.3 that $S(x) = \widehat{V}(x) \bmod x^r$, and the Key Equation (9.37) becomes

$$\sigma(x)S(x) \equiv \omega(x) \pmod{x^r}, \tag{9.77}$$

* Unless, of course, some of the erased components of $\mathbf{C}$ were actually 0, in which cases C and R' would differ in fewer than $e_0 + e_1$ positions.

where $\sigma(x)$ is the locator polynomial, and $\omega(x)$ is the evaluator polynomial, for the vector V. From now on, we'll call $\sigma(x)$ the *errors-and-erasures locator polynomial*, and $\omega(x)$ *errors-and-erasures evaluator polynomial*.

Let's focus for a moment on $\sigma(x)$, the errors-and erasures locator polynomial. We have

$$\sigma(x) = \prod_{i \in I}(1 - \alpha^i x), \tag{9.78}$$

where I is the errors-and erasures set, i.e.,

$$I = I_0 \cup I_1, \tag{9.79}$$

where I_0 is the erasure set defined in (9.73) and I_1 is the *error set* defined as follows:

$$I_1 = \{i : R_i \neq * \text{ and } R_i \neq C_i.\}.$$

It thus follows from (9.78) and (9.79) that

$$\sigma(x) = \sigma_0(x)\sigma_1(x), \tag{9.80}$$

where $\sigma_0(x)$ is as defined in (9.74) and

$$\sigma_1(x) = \prod_{i \in I_1}(1 - \alpha^i x). \tag{9.81}$$

Naturally we call $\sigma_1(x)$ the *error locator polynomial.*

Now we return to the key equation (9.77). In view of (9.80), we already know part of $\sigma(x)$, viz. $\sigma_0(x)$, and so the decoding algorithm's next step is to compute the *modified syndrome polynomial* $S_0(x)$, defined as follows:

$$S_0(x) = \sigma_0(x)S(x) \bmod x^r. \tag{9.82}$$

Combining (9.77), (9.80), and (9.82), the key equation becomes

$$\sigma_1(x)S_0(x) \equiv \omega(x) \pmod{x^r}. \tag{9.83}$$

At this point, the decoder will know $S_0(x)$, and wish to compute $\sigma_1(x)$ and $\omega(x)$, using Euclid's algorithm. Is this possible? Yes, because we have

$$\begin{aligned} \deg \sigma_1(x) &= e_1 \\ \deg \omega(x) &\leq e_0 + e_1 - 1 \end{aligned}$$

so that $\deg \sigma_1 + \deg \omega \leq e_0 + 2e_1 - 1 < r = \deg x^r$, since we have assumed $e_0 + 2e_1 \leq r$. Although it may no longer be true that $\gcd(\sigma(x), \omega(x)) = 1$, it will be true that $\gcd(\sigma_1(x), \omega(x)) = 1$ (see Prob. 9.45). It thus follows from Theorem 9.6 that the procedure `Euclid`$(x^r, S_0(x), \mu, \nu)$ will return $\sigma_1(x)$ and $\omega(x)$, if μ and ν are chosen properly. To chose μ and ν, we reason as follows. Since $e_0 + 2e_1 \leq r$, we have

$$\deg \sigma_1(x) = e_1 \leq \frac{r - e_0}{2},$$

so that $\deg \sigma_1(x) \leq \lfloor (r - e_0)/2 \rfloor$. Similarly,

$$\begin{aligned} \deg \omega(x) \leq e_0 + e_1 - 1 &\leq e_0 + \left\lfloor \frac{r - e_0}{2} \right\rfloor - 1 \\ &= \left\lfloor \frac{r + e_0}{2} \right\rfloor - 1 \\ &\leq \left\lceil \frac{r + e_0}{2} \right\rceil - 1 \end{aligned} \quad .$$

It is an easy exercise to prove that $\lfloor (r-e_0)/2 \rfloor + \lceil (r+e_0)/2 \rceil = r$ (See Prob. 9.46), and so it follows that if we define

$$\begin{aligned} \mu &= \left\lfloor \frac{r-e_0}{2} \right\rfloor \\ \nu &= \left\lceil \frac{r+e_0}{2} \right\rceil - 1, \end{aligned} \tag{9.84}$$

then the procedure $\texttt{Euclid}(x^r, S_0(x), \mu, \nu)$ is guaranteed return a pair of polynomials $(v(x), r(x))$ such that $\sigma_1(x) = \lambda v(x), \omega(x) = \lambda r(x)$, where λ is a nonzero scalar. To find λ we recall that $\sigma_1(0) = 1$ (see 9.81), and so we have

$$\begin{aligned} \sigma_1(x) &= v(x)/v(0) \\ \omega(x) &= r(x)/v(0). \end{aligned}$$

Now, having computed the erasure locator polynomial $\sigma_0(x)$ and the error locator polynomial $\sigma_1(x)$, the algorithm computes the erasure and error locator polynomial $\sigma(x)$ by polynomial multiplication — see (9.80).

At this stage, the algorithm has both the locator polynomial $\sigma(x)$ and evaluator polynomial $\omega(x)$ for the errors-and-erasures vector E', and the decoding can be completed by either the "time domain completion" or the "frequency domain completion" described in Section 9.6. The errors-and-erasures decoding algorithm is thus summarized in Figure 9.9.

```
/*RS Errors and Erasures Decoding Algorithm*/
{
  Input I_0;  e_0 = |I_0|;
  σ_0(x) = ∏_{i∈I_0}(1 − α^i x);
  for (i ∈ I_0);
     R_i = 0;
  for (j = 1, 2, ..., r)
     S_j = Σ_{i=0}^{n−1} R_i α^{ij};
  S(x) = S_1 + S_2 x + ··· + S_r x^{r−1};
  S_0(x) = σ_0(x)S(x) mod x^r;
  μ = ⌊(r − e_0)/2⌋; ν = ⌈(r + e_0)/2⌉ − 1;
  Euclid(x^r, S_0(x), μ, ν);
  σ_1(x) = v(x)/v(0);
  ω(x) = r(x)/v(0);
  σ(x) = σ_0(x)σ_1(x);
  ⋮
  (Time domain completion or frequency domain completion)
}
```

Figure 9.9. Decoding RS (or BCH) codes when erasures are present.

Example 9.10. We illustrate the erasures-and-errors RS decoding algorithm with the (7,2) RS code over the field $GF(8)$, which has generator polynomial $g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5) = x^5 + \alpha^2 x^4 + \alpha^3 x^3 + \alpha^6 x^2 + \alpha^4 x + \alpha$. (We are assuming that α, a primitive root in $GF(8)$, is a root of the $GF(2)$ - primitive polynomial $x^3 + x + 1$.) The code's redundancy is $r = 5$ and so by Theorem 9.11, it can correct any pattern of e_0 erasures and e_1 errors, provided $e_0 + 2e_1 \leq 5$. Let us take the garbled codeword

$$\mathbf{R} = [\alpha^4, \alpha^3, \alpha^6, *, \alpha^2, \alpha^4, \alpha^2],$$

and try to decode it, using the algorithm in Figure 9.9.

The first phase of the decoding algorithm is the "erasure management," which in this case amounts simply to observing that the erasure set is $I_0 = \{3\}$, so that $e_0 = 1$, the erasure locator polynomial is

$$\sigma_0(x) = 1 + \alpha^3 x,$$

and the modified received vector $\mathbf{R}'$ is

$$\mathbf{R}' = [\alpha^4, \alpha^3, \alpha^6, 0, \alpha^2, \alpha^4, \alpha^2].$$

The next step is to compute the syndrome values S_1, S_2, S_3, S_4, S_5, using $\mathbf{R}'$. We have

$$S_j = \alpha^4 + \alpha^{3+j} + \alpha^{6+2j} + \alpha^{2+4j} + \alpha^{4+5j} + \alpha^{2+6j},$$

so that a routine calculation gives

$$S_1 = 1, S_2 = 1, S_3 = \alpha^5, S_4 = \alpha^2, S_5 = \alpha^4.$$

Thus the modified syndrome polynomial $S_0(x)$ is

$$\begin{aligned} S_0(x) &= (1 + x + \alpha^5 x^2 + \alpha^2 x^3 + \alpha^4 x^4)(1 + \alpha^3 x) (\mathrm{mod}\, x^5) \\ &= 1 + \alpha x + +\alpha^2 x^2 + \alpha^4 x^3 + x^4 \end{aligned}$$

Since $e_0 = 1$, $r = 5$, the parameters μ and ν are

$$\mu = \left\lfloor \frac{5-1}{2} \right\rfloor = 2$$
$$\nu = \left\lceil \frac{5+1}{2} \right\rceil - 1 = 2$$

Thus we are required to invoke $\texttt{Euclid}(x^5, S_0(x), 2, 2)$. Here is a summary of the work:

i	v_i	r_i	q_i
-1	0	x^5	$--$
0	1	$x^4 + \alpha^4 x^3 + \alpha^2 x^2 + \alpha x + 1$	$--$
1	$x + \alpha^4$	$\alpha^4 x^3 + \alpha^5 x^2 + \alpha^4 x + \alpha^4$	$x + \alpha^4$
2	$\alpha^3 x^2 + \alpha^4 x + \alpha^6$	$\alpha^4 x^2 + \alpha^5 x + \alpha^6$	$\alpha^3 x + \alpha^5$

Thus $\texttt{Euclid}(x^5, S_0(x), 2, 2)$ returns $(v_2(x), r_2(x)) = (\alpha^3 x^2 + \alpha^4 x + \alpha^6, \alpha^4 x^2 + \alpha^5 x + \alpha^6)$, so that

$$\sigma_1(x) = \alpha v_2(x) = \alpha^4 x^2 + \alpha^5 x + 1$$
$$\omega(x) = \alpha r_2(x) = \alpha^5 x^2 + \alpha^6 x + 1$$

and finally

$$\sigma(x) = \sigma_0(x)\sigma_1(x) = x^3 + \alpha^2 x^2 + \alpha^2 x + 1.$$

This completes the "erasure specific" part of the decoding, i.e., the portion of the algorithm described in Figure 9.9. We will now finish the decoding, using both the time domain and frequency domain completions.

For the time domain completion, we note that $\sigma'(x) = x^2 + \alpha^2$, and compute the following table:

i	$\sigma(\alpha^{-i})$	$\sigma'(\alpha^{-i})$	$\omega(\alpha^{-i})$	$E_i = \omega(\alpha^{-i})/\sigma'(\alpha^{-i})$
0	0	α^6	α^3	α^4
1	α^5			
2	α^4			
3	0	α^4	α^5	α
4	0	1	α^3	α^3
5	α^5			
6	α^5			

Thus the errors-and-erasures vector is $\mathbf{E}' = [\alpha^4, 0, 0, \alpha, \alpha^3, 0, 0]$ (which means that there are two errors, in positions 0 and 4, in addition to the erasure in position 3), and so the decoded codeword is $\hat{\mathbf{C}} = \mathbf{R}' + \mathbf{E}'$, i.e.,

$$\hat{\mathbf{C}} = [0, \alpha^3, \alpha^6, \alpha, \alpha^5, \alpha^4, \alpha^2].$$

For the frequency-domain completion, having already computed S_1, S_2, S_3, S_4, S_5, we compute S_6 and S_7 $(= S_0)$ via the recursion

$$S_j = \alpha^2 S_{j-1} + \alpha^2 S_{j-2} + S_{j-3}$$

(since $\sigma(x) = 1 + \alpha^2 x + \alpha^2 x^2 + x^3$), and find that $S_6 = \alpha^2$, and $S_0 = \alpha^5$. Thus the complete syndrome vector S is

$$S = [\alpha^5, 1, 1, \alpha^5, \alpha^2, \alpha^4, \alpha^2].$$

we now compute the inverse DFT of S, i.e.,

$$\begin{aligned}\mathbf{E}'_i &= \alpha^5 + \alpha^{-i} + \alpha^{-2i} + \alpha^{5-3i} + \alpha^{2-4i} + \alpha^{4-5i} + \alpha^{2-6i} \\ &= \alpha^5 + \alpha^{6i} + \alpha^{5i} + \alpha^{5+4i} + \alpha^{2+3i} + \alpha^{4+2i} + \alpha^{2+i}\end{aligned}$$

This gives

$$\mathbf{E}' = [\alpha^4, 0, 0, \alpha, \alpha^3, 0, 0]$$

just as in the time-domain completion, and so

$$\hat{\mathbf{C}} = [0, \alpha^3, \alpha^6, \alpha, \alpha^5, \alpha^4, \alpha^2]$$

as before. ∎

Let's conclude this section with a brief discussion of how to decode **BCH** codes when erasures are present. The key difference between the (errors only) decoding algorithm for BCH codes and RS codes is that for BCH codes, once the errors have been *located*, there is no need to *evaluate* them, since the only possible error value is 1. What this means is that when erasures are present, the algorithm in Figure 9.9 still holds (with $2t$ replacing r); the only way in which the decoding of BCH codes is simpler is in the implementation of the time domain completion. (Compare Figures 9.4 and 9.5.)

9.8 The $(23, 12)$ Golay Code.

In this section we will discuss an extremely beautiful but alas! nongeneralizable code, the binary $(23, 12)$ Golay code. It is arguably the single most important error-correcting code. (There is also an $(11, 6)$ Golay code over $GF(3)$; see Probs. 9.62–9.65)

We begin with a tantalizing number-theoretic fact. In the 23-dimensional vector space over $GF(2)$, which we call V_{23}, a Hamming sphere of radius 3 contains

$$1 + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 2048 \text{ vectors.}$$

But $2048 = 2^{11}$ is an exact power of 2, and thus it is conceivable that we could pack V_{23} with $4096 = 2^{12}$ spheres of radius 3, exactly, with no overlap. If we could perform this combinatorial miracle, the centers of the spheres would constitute a code with 2^{12} codewords of length 23 (rate $= 12/23 = 0.52$) capable of correcting any error pattern of weight ≤ 3. In this section, not only will we prove that such a packing is possible; we will show that the centers of the spheres can be taken as the codewords in a $(23, 12)$ binary cyclic code!

In coding-theoretic terms, then, we need to construct a binary cyclic $(23, 12)$ triple error-correcting code, i.e., one with $d_{\min} \geq 7$. We base the construction on certain properties of the field $GF(2^{11})$. Since $2^{11} - 1 = 2047 = 23 \cdot 89$, $GF(2^{11})$ must contain a primitive 23rd root of unity, which we shall call β. The minimal polynomial of β over $GF(2)$ is $g(x) = \prod_{\gamma \in B}(x - \gamma)$, where $B = \{\beta^{2^i} : i = 0, 1, 2, \ldots\}$ is the set of conjugates of β. A simple computation shows that B contains only 11 elements; indeed,

$$g(x) = \prod_{\gamma \in B}(x - \gamma), \tag{9.85}$$

where

$$B = \{\beta^j : j = 1, 2, 4, 8, 16, 9, 18, 13, 3, 6, 12\}.$$

Similarly the minimal polynomial of $\beta^{-1} = \beta^{22}$ is

$$\widetilde{g}(x) = \prod_{\gamma \in \widetilde{B}}(x - \gamma) \tag{9.86}$$

where

$$\widetilde{B} = \{\beta^j : j = 22, 21, 19, 15, 7, 14, 5, 10, 20, 17, 11\}.$$

Since every 23rd root of unity except 1 is a zero of either $g(x)$ or $\widetilde{g}(x)$, it follows that the factorization of x^{23} into irreducible factors over $GF(2)$ is

$$x^{23} - 1 = (x - 1)g(x)\widetilde{g}(x) \tag{9.87}$$

In fact, it can be shown that

$$\begin{aligned} g(x) &= x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1 \\ \widetilde{g}(x) &= x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1, \end{aligned} \tag{9.88}$$

but we will not need this explicit factorization in the rest of this section. We can now define the Golay code.

`Definition.` The $(23, 13)$ Golay code is the binary cyclic code whose generator polynomial is $g(x)$, as defined in (9.85) or (9.88).

Now all (!) we have to do is show that the code's minimum weight is ≥ 7. The first step in this direction is rather easy.

Lemma 9.3. *Each nonzero Golay codeword has weight* ≥ 5.

Proof: In view of the structure of the set B of zeroes of $g(x)$ (cf. Eq. (9.85)), we see that for every codeword C,

$$C(\beta) = C(\beta^2) = C(\beta^3) = C(\beta^4) = 0. \tag{9.89}$$

It thus follows from the BCH argument (Theorem 9.3) that $d_{\min} \geq 5$. ∎

In view of Lemma 9.3, it remains to show that there can be no words of weight 5 or 6. The next lemma allows us to focus our attention on words of even weight.

Lemma 9.4. *If A_i denotes the number of Golay codewords of weight i, then, for $0 \leq i \leq 23$,*

$$A_i = A_{23-i}. \tag{9.90}$$

Proof: Note that $g(x)\widetilde{g}(x) = (x^{23} - 1)/(x - 1) = 1 + x + x^2 + \cdots + x^{22}$, and so the constant vector $K = (11111...111)$ is in the code. By adding K to a word of weight i, we get a word of weight $23 - i$, and conversely. Thus the correspondence $C \leftrightarrow C + K$ is one-to-one between words of weights i and $23 - i$. ∎

The next lemma eliminates words of weight 2, 6, 10, 14, 18, and 22; by Lemma 9.4 this also eliminates words of weight 1, 5, 9, 13, 17, and 21, and thus proves that the Golay code has minimum distance ≥ 7.

Lemma 9.5. *If C is a Golay codeword of even weight w, then $w \equiv 0 \pmod 4$.*

Proof: Let the generating function for C be denoted by $C(x)$, i.e.,

$$C(x) = x^{e_1} + x^{e_2} + \cdots + x^{e_w}, \tag{9.91}$$

where $0 \leq e_1 < e_2 < \cdots < e_w \leq 22$. Since C belongs to the Golay code, $C(\beta) = 0$, that is,

$$C(x) \equiv 0 \pmod{g(x)}. \tag{9.92}$$

Since C has even weight, $C(1) = 0$, that is

$$C(x) \equiv 0 \pmod{x - 1}. \tag{9.93}$$

Now if we define $\widetilde{C}(x)$ by

$$\widetilde{C}(x) = x^{-e_1} + x^{-e_2} + \cdots + x^{-e_w}, \tag{9.94}$$

with exponents taken modulo 23, it follows that $\widetilde{C}(\beta^{-1}) = C(\beta) = 0$, that is ,

$$\widetilde{C}(x) \equiv 0 \pmod{\widetilde{g}(x)}. \tag{9.95}$$

Combining (9.92), (9.93), (9.95) with (9.87), we have

$$C(x)\widetilde{C}(x) \equiv 0 \pmod{x^{23} - 1}. \tag{9.96}$$

Now let us actually compute $C(x)\tilde{C}(x) (\mathrm{mod}\, x^{23} - 1)$, using the defining equations (9.91) and (9.94):

$$\begin{aligned} C(x)\widetilde{C}(x) &\equiv \sum_{i,j=1}^{w} x^{e_j - e_j} \pmod{x^{23}-1} \\ &\equiv w + \sum_{\substack{i,j=1 \\ i\neq j}}^{w} x^{e_j - e_j} \pmod{x^{23}-1} \\ &\equiv \sum_{\substack{i,j=1 \\ i\neq j}}^{w} x^{e_i - e_j} \pmod{x^{23}-1} \end{aligned} \tag{9.97}$$

(the last equality because w is even and all computations take place in $GF(2)$). Thus

$$C(x)\widetilde{C}(x) \equiv \sum_{b=1}^{22} \mu_b x^b \pmod{x^{23}-1},$$

where μ_b is the number of ordered pairs (i,j) with $e_i - e_j \equiv b \pmod{23}$. By Eq. (9.96) each μ_b is even:

$$\mu_b \equiv 0 \pmod 2, \quad b = 1, 2, \ldots, 22. \tag{9.98}$$

Now, if $e_i - e_j \equiv b$, then also $e_j - e_i \equiv 23 - b \pmod{23}$. Thus

$$\mu_b = \mu_{23-b}, \quad b = 1, 2, \ldots, 11. \tag{9.99}$$

Finally, since there are $w(w-1)$ terms in the sum on the right side of (9.97),

$$\sum_{b=1}^{22} \mu_b = w(w-1). \tag{9.100}$$

Combining (9.98), (9.99), and (9.100), we have

$$\begin{aligned} w(w-1) &= \sum_{b=1}^{22} \mu_b \\ &= 2\sum_{b=1}^{11} \mu_b \\ &\equiv 0 \pmod 4, \end{aligned}$$

i.e., $w(w-1)$ is a multiple of 4. But since $w-1$ is *odd*, it follows that w itself is a multiple of 4, which completes the proof. ∎

Combining Lemmas 9.3, 9.4, and 9.5, we arrive at the following theorem.

`Theorem 9.13.` *The number of codewords of weight i in the $(23, 12)$ Golay code is 0 unless $i = 0$, 7, 8, 11, 12, 15, 16, or 23. Thus the spheres of radius 3 around the codewords do indeed pack V_{23} perfectly.*

There is a simple but useful variant of the $(23, 12)$ Golay code, that deserves mention here. If $C = (C_0, C_1, \ldots, C_{22})$ is a Golay codeword, let us extend C to length 24 by *appending an overall parity check*, i.e., by defining a 24th component C_{23} as follows:

$$C_{23} = C_0 + C_1 + \cdots + C_{22}.^*$$

If every Golay codeword is extended in this way, the resulting code is a binary linear (but no longer cyclic) $(24, 12)$ code, which is called the $(24, 12)$ *extended Golay code*. It is a simple matter then to prove the following theorem. (See Problem 9.60).

* See Problem 7.17b.

`Theorem 9.14.` *The number of codewords of weight i in the $(24, 12)$ extended Golay code is* 0 *unless* $i = 0$,8, 12, 16, *or* 24.

The $(24, 12)$ extended Golay code enjoys two small advantages over the original $(23, 12)$ Golay code, which are however enough to make the extended code preferable in most applications. First, since 24 is a multiple of eight, the $(24, 12)$ code is naturally suited to byte-oriented implementations. Second, since the minimum distance of the extended code is *eight*, if it is used to correct all patterns of three or fewer errors, all error patterns of weight 4, and many error patterns of higher weight, will still be *detectable*, whereas the original $(23, 12)$ has no such extra detection capability. (See Problems 9.59, 9.62).

We conclude with some remarks about the implementation of the Golay codes. Since the $(23, 12)$ code is cyclic, it is clear that we could design an 11-stage shift-register encoder for it (see Section 8.2, and do Problem 9.60.). The design of an algebraic decoding algorithm is not so easy; we could easily modify the BCH decoding algorithm in Figures 9.1 or 9.2 to correct every pattern of two or fewer errors, but the code is "accidentally" capable of correcting three! Fortunately, however, the code is small enough so that the syndrome "table lookup" algorithm discussed in Section 7.2 is usally practical. (See Prob. 9.61).

9.9. Problems for Chapter 9.

9.1. We saw in Section 9.1 that the function $\mathbf{f}(\mathbf{V}) = \mathbf{V}^3$ makes the matrix H_2 of Eq. (9.3) into the parity-check matrix of a double-error correcting code. Investigate whether or not the following candidate $\mathbf{f}$'s work:

(a) $\mathbf{f}(\mathbf{V}) = T\mathbf{V}$, where T is a linear transformation of V_m.

(b) $\mathbf{f}(\mathbf{V}) = a_0 + a_1\mathbf{V} + a_2\mathbf{V}^2$, where $\mathbf{V}$ is regarded as an element of $GF(2^m)$.

(c) $\mathbf{f}(\mathbf{V}) = \mathbf{V}^{-1}$, where $\mathbf{V} \in GF(2^m)$.

9.2. Suppose F is a finite field with q elements.

(a) If a is an arbitrary element of F, define the $q-1$st degree polynomial $f_a(x) = (x-a)^{q-1} - 1$. Find the value of $f_a(x)$ for each of the q elements of F.

(b) Using the results of part a, or otherwise, show that every function $f : F \to F$ can be represented as a polynomial of degree at most $q-1$.

9.3. (This problem gives a generalization of the Vandermonde determinant theorem, which is needed in the proof of Theorem 9.1.) Let $P_i(x)$ be a monic polynomial of degree i, for $i = 0, 1, \ldots, n-1$, and let $x_1, x_2, \ldots, x_n$ be distinct indeterminates. Show that

$$\det \begin{pmatrix} P_0(x_1) & \cdots & P_0(x_n) \\ P_1(x_1) & \cdots & P_1(x_n) \\ \vdots & & \vdots \\ P_{n-1}(x_1) & \cdots & P_{n-1}(x_n) \end{pmatrix} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} (x_j - x_i).$$

[Hint: If $x_i = x_j$, the left side vanishes.] The Vandermonde determinant theorem is the special case $P_i(x) = x^i$.

9.4. Here is a pseudocode listing for an algorithm for computing the dimension of the t error-correcting BCH code of length $2^m - 1$:

```
{
  S = {1, 3, ..., 2t − 1};
  k = 2^m − 1;
  while(S is not empty)
  {
    u_0 = least element in S;
    u = u_0;
    do
    {
      delete u from S;
```

```
        k = k − 1;
        u = 2u mod 2^m − 1;
      }
      while(u ≠ u_0)
    }
  }
```

(a) Show that when the algorithm terminates, the integer k is equal to the dimension of the t error-correcting BCH code of length $2^m - 1$.

(b) Use the algorithm to compute the dimension of the t error-correcting BCH code of length 63, for $1 \le t \le 31$.

9.5. (a) Prove that the dimension of the two-error-correcting BCH code of length $n = 2^m - 1$ is $n - 2m$, for all $m \ge 3$.

(b) More generally show that for any fixed $t \ge 1$, the dimension of the t-error-correcting BCH code of length $n = 2^m - 1$ is $n - mt$ for all sufficiently large m.

(c) What is the smallest value of m_0 such that the *three* error-correcting BCH code of length $n = 2^m - 1$ has dimension $n - 3m$ for all $m \ge m_0$?

9.6.

(a) For each t in the range $1 \le t \le 7$, compute the dimension of the t-error correcting BCH code of length 15.

(b) For each of the codes in part (a), calculate the generator polynomial, assuming a primitive root α in $GF(16)$ that satisfies the equation $\alpha^4 = \alpha + 1$. (Cf. Example 9.1.)

9.7. In Example 9.1 we computed the generator polynomial for the three-error-correcting BCH code of length 15, under the assumption that the primitive root α of $GF(16)$ satisfied $\alpha^4 = \alpha + 1$. If we had chosen a primitive root satisfying $\alpha^4 = \alpha^3 + 1$ instead, what would the generator polynomial have turned out to be?

9.8. Prove the inverse DFT formula, Eq. (9.12)

9.9. Consider the field $GF(7)$, as represented by the set of integers $\{0, 1, 2, 3, 4, 5, 6\}$, with all arithmetic done modulo 7.

(a) Show that 3 is a primitive 6th root of unity in $GF(7)$.

(b) Using 3 as the needed primitive 6th root of unity, find the DFT of the vectors $\mathbf{V}_1 = (1, 2, 3, 4, 5, 6)$ and $\mathbf{V}_2 = (1, 3, 2, 6, 4, 5)$.

(c) Why do you suppose the DFT of $\mathbf{V}_2$ is so much "simpler" than that of $\mathbf{V}_1$?

9.10. Prove that the DFT of the phase-shifted vector $\mathbf{V}_\mu$ in equation (9.17) is given by the formula (9.18).

9.11. (Generalized BCH codes). Let $g(x)$ be a polynomial with coefficients in $GF(q)$ which divides $x^n - 1$. Further assume that α is an nth root of unity in some extension field of $GF(q)$ and that

$$g(\alpha^i) = 0, \qquad \text{for } i = m_0, m_0 + 1, \ldots, m_0 + d - 2.$$

for integers m_0 and d. Let C be the cyclic code with generator polynomial $g(x)$. Show that the minimum distance of C is $\geq d$. [Hint: Use the BCH argument.]

9.12. Is the converse of the "BCH Argument" (Theorem 9.3) true? That is, if $\mathbf{V}$ is a vector of weight w, does it necessarily follow that the DFT $\widehat{\mathbf{V}}$ must have $w - 1$ or more consecutive zero components? If your answer is *yes*, give a proof. If your answer is *no*, give an explicit counterexample.

9.13. Prove that $\gcd(\widehat{V}(x), 1 - x^n) = \prod_{i \notin I}(1 - \alpha^i n)$, where $\widehat{V}(x)$ is as defined in (9.14), and I is as defined in (9.19).

9.14. Show that if any d consecutive components of $\widehat{\mathbf{V}}$ are known, the rest can be calculated, provided we also know $\sigma_{\mathbf{V}}(x)$ (cf. Corollary 9.2).

9.15. The field $GF(16)$, as represented in Table 9.1, contains a primitive 5th root of unity, namely α^3, which for the remainder of this problem we shall denote by β. Let $\mathbf{V} = (0, \alpha^4, \alpha^5, 0, \alpha^7)$, a vector of length 5 over $GF(16)$. Using the definitions in Section 9.3, compute $\widehat{\mathbf{V}}$, $\sigma_{\mathbf{V}}$, $\sigma_{\mathbf{V}}^{(i)}$ for $i = 1, 2, 4$, and $\omega_{\mathbf{V}}(x)$. (Cf. Example 9.2.)

9.16. In Example 9.2, the components of $\widehat{\mathbf{V}}$ satisfy the recursion $\widehat{V}_j = \alpha^6 \widehat{V}_{j-3}$. Explain "why" this should be so. [Hint: Examine (9.31) carefully.]

9.17. If S_j represents the jth syndrome value in the decoding of a BCH code (cf. Eq. (9.34)), show that, for all j, $S_{2j} = S_j^2$.

9.18. If $f(x) = f_0 + f_1 x + \cdots + f_n x^n$ is a polynomial over a field f, its *formal derivative* $f'(x)$ is defined as follows.

$$f'(x) = f_1 + 2f_2 x + \cdots + n f_n x^{n-1}.$$

From this definition, without the use of limits, deduce the following facts:

(a) $(f + g)' = f' + g'$.

(b) $(fg)' = fg' + f'g$.

(c) $(f^m)' = m f^{m-1} f'$.

(d) If $f(x) = \prod_{i=1}^{r}(x - \beta_i)$, then

$$f'(x) = \sum_{i=1}^{r} \prod_{\substack{j=1 \\ j \neq i}}^{r} (x - \beta_j).$$

(e) If $f(x)$ is as given in part (d) and the β_i are distinct, then

$$\sum_{i=1}^{r} \frac{1}{x - \beta_i} = \frac{f'(x)}{f(x)}.$$

9.19.

(a) Prove properties A-F of Euclid's algorithm given in Table 9.2.

(b) Prove that $r_n(x)$, the last nonzero remainder in Euclid's algorithm, is a greatest common divisor of $a(x)$ and $b(x)$, and that Eq. (9.43) holds.

9.20. Let $a(x) = x^8 - 1$, $b(x) = x^6 - 1$, polynomials over the field $GF(2)$.

(a) Apply Euclid's algorithm to the pair $(a(x), b(x))$, thus obtaining a table like that in Example 9.3.

(b) For each pair (μ, ν) with $\mu \geq 0$, $\nu \geq 0$, and $\mu + \nu = 7$, calculate $\texttt{Euclid}(a, b, \mu, \nu)$. (Cf. Example 9.4.)

9.21. (Padé approximants). Let $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots$ be a power series over a field F. If μ and ν are nonnegative integers, a (μ, ν) *Padé approximation* to $A(x)$ is a rational function $p(x)/q(x)$ such that

$$(i) \qquad q(x)A(x) \equiv p(x) \pmod{x^{\mu+\nu+1}}$$

$$(ii) \qquad \deg q(x) \leq \mu, \deg p(x) \leq \nu$$

Using Theorem 9.5, show that for each (μ, ν) there is (apart from a scalar factor) a unique pair $(p_0(x), q_0(x))$ such that if (i) and (ii) hold, then $p(x) = \lambda p_0(x)$ and $q(x) = \lambda q_0(x)$ for a nonzero scalar λ. The pair $(p_0(x), q_0(x))$ is called *the* (μ, ν) *Padé approximant* to $A(x)$. Referring to Table 9.3, compute the Padé approximants to $A(x) = 1 + x + x^2 + x^4 + x^6 + \ldots$ over $GF(2)$ with $\mu + \nu = 7$.

9.22. With the same setup as Example 9.5, decode the following noisy codeword from the $(15, 5)$ 3-error correcting BCH code:

$$R = [R_0, \ldots, R_{14}] = [110101010010010].$$

9.23. Consider the three-error-correcting BCH code of length 31, defined in terms of a primitive root $\alpha \in GF(32)$ satisfying $\alpha^5 + \alpha^2 + 1 = 0$.

(a) Compute the generator polynomial.

(b) Decode the following received vector: [0000000111101011111011100010000].

(c) Decode the following received vector: [1011001111101010011000100101001].

9.24. Let α be a primitive nth root of unity in the field F, and let $\mathcal{P}_k$ be the set of polynomials of degree $\leq k - 1$ over F, For each $P \in \mathcal{P}_k$, define the vector $\mathbf{C}(P) = (P(1), P(\alpha), \ldots, P(\alpha^{n-1}))$.

(a) Show that the code consisting of all vectors $\mathbf{C}(P)$, is an MDS code, and find the corresponding n, k, and d.

(b) Is the code cyclic? Explain.

(c) What relationship, if any, does this code bear to the RS code as defined in (9.67)?

9.25. Let F be any field which contains a primitive nth root of unity α. If r and i are fixed integers between 0 and n, the set of all vectors $\mathbf{C} = (C_0, C_1, \ldots, C_{n-1})$ with components in F such that

$$\sum_{i=0}^{n-1} C_i \alpha^{ij} = 0, \quad \text{for } j = i+1, i+2, \ldots, i+r$$

is called an *alternate* Reed-Solomon code.

(a) Show that the code so defined is an $(n, n-r)$ cyclic code. Find the generator polynomial and $d_{\min}$.

(b) Explicitly compute the generator polynomial $g(x)$ for the alternate RS code with $F = GF(8)$, $n = 7$, $r = 4$, and $i = 1$. (Cf. Example 9.6.)

(c) Show that there exist n fixed nonzero elements from F, say $\gamma_0, \ldots, \gamma_{n-1}$, such that the alternate code just described can be obtained from the original RS code (defined in (9.67)) by mapping each original RS codeword $(C_0, \ldots, C_{n-1})$ to the vector $(\gamma_0 C_0, \ldots, \gamma_{n-1} C_{n-1})$.

9.26. Let α be an nth root of unity in a finite field F, and let $\mathcal{C}$ be the linear code of length n defined as follows: $C = (C_0, C_1, \ldots, C_{n-1})$ is a codeword if and only if

$$\sum_{i=0}^{n-1} \frac{C_i}{x - \alpha^i} \equiv 0 \pmod{x^r},$$

where x is an indeterminate.

(a) Show that the code $\mathcal{C}$ is cyclic.

(b) Find the dimension of the code, in terms of n and r.

(c) Find the code's minimum distance.

(d) What is the relationship (if any) between $\mathcal{C}$ and the (n, k) RS code over F with generator polynomial $g(x) = (x - \alpha) \cdots (x - \alpha^r)$?

9.27. Give a complete discussion of *all* MDS codes over $GF(2)$.

9.28. Show that over any field F, the following two codes are MDS codes.

(a) The $(n, 1)$ repetition code.

(b) The $(n, n-1)$ parity-check code.

9.29. In Theorem 9.6, it is shown that in a (n, k) Reed-Solomon code over a field F, the minimum weight codeword has weight $n - k + 1$. Question: If the field F has q elements, exactly how many words of weight $n - k + 1$ are there in the code? (Hint: Use Theorem 9.9.)

9.30. Using Table 9.1 for guidance, compute the generator polynomial $g(x)$ for for a $(15, 7)$ RS code over $GF(16)$. (Cf. Example 9.6.)

9.31. Consider the $(7, 3)$ RS code of Example 9.6.

(a) Find the unique codeword $(C_0, \ldots, C_6)$ such that $C_0 = 1$ $C_1 = 0$, and $C_2 = 0$. (Cf. Example 9.7.)

(b) Is there a codeword with $C_0 = \alpha^3$, $C_1 = \alpha$, $C_2 = 1$, and $C_3 = 0$?

9.32. Decode the following garbled codeword from the $(7, 3)$ RS code from Example 9.8:

$$\mathbf{R} = [\alpha^3 \ 1 \ \alpha \ \alpha^2 \ \alpha^3 \ \alpha \ 1 \ ].$$

9.33. Do you think there is a $(21, 9)$ binary linear code which is capable of correcting any pattern of 4 or fewer errors?

Problems 9.34–9.39 are all related. The present an alternative approach to Reed-Solomon codes. More important, however, they culminate with the construction of the famous Justesen codes (see Ref. [15]). These codes are important because they (together with certain variants of them) are the only known explicity constructable family of linear codes which contain sequences of codes of lengths n_i, dimensions k_i, and minimum distances d_i such that:

$$\lim_{i\to\infty} n_i = \infty,$$
$$\lim_{i\to\infty} k_i/n_i > 0,$$
$$\lim_{i\to\infty} d_i/n_i > 0.$$

(See conclusions of Prob. 9.39. Also cf. Prob. 7.21, the Gilbert bound, which shows that such sequences of codes must exist, but does not explain how to construct them.)

9.34. (Cf. Theorem 9.10.) Denote by P_r the set of polynomials of degree $\leq r$ over the finite field $GF(q^m)$ and let $(\alpha_0, \alpha_1, \ldots, a_{n-1})$ be a list of $n > r$ distinct elements from $GF(q^m)$. Corresponding to each $f(x) \in P_r$, let the vector $(C_0, C_1, \ldots, C_{n-1}) \in GF(q^m)^n$ be defined by $C_i = f(\alpha_i)$. Show that the set of vectors obtained from P_r in this way is a linear code over F_{q^m} with length n, dimension $r+1$, and minimum distance $n-r$.

9.35. The setup being the same as in Prob. 9.34, corresponding to each $f \in P_r$ let

$$\mathbf{C} = (C_0, C_0', C_1, C_1', \ldots, C_{n-1}, C_{n-1}') \in GF(q^m)^{2n}$$

be defined by $C_i = f(\alpha_i)$, $C_i' = \alpha_i f(\alpha_i)$. Show that the set of vectors thus obtained is a linear code over $GF(q^m)$ with length $2n$ and dimension $r+1$, such that within each nonzero codeword there are at least $n-r$ distinct pairs (C_i, C_i').

9.36. Let $\phi : GF(q^m) \to GF(q)^m$ be a one-to-one linear mapping from $GF(q^m)$ onto $GF(q)^m$. Take the code over $GF(q^m)$ defined in Prob. 9.35, and make it into a code over $GF(q)$ by mapping the codeword $\mathbf{C}$ onto $(\phi(C_0), \phi(C_0'), \ldots, \phi(C_{n-1}), \phi(C_{n-1}'))$. Show that the resulting $GF(q)$ linear code has length $2mn$ and dimension $m(r+1)$, and that within each nonzero codeword, among the n subvectors $(\phi(C_i), \phi(C_i'))$, there are at least $n-r$ distinct ones.

9.37. (This problem is not out of place, despite appearances.) Let $\{x_1, \ldots, x_M\}$ be a set of M distinct vectors with components from $GF(2)$, and let $w_i = w_H(x_i)$ denote the Hamming weight of x_i. Let $p = (w_1 + \cdots + w_M)/nM$. Prove that

$$\log M \leq nH_2(p),$$

where H_2 is the binary entropy function. [Hint: Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ be a random vector which is equally likely to assume any of the values x_i, and let p_j denote the fraction of the M vectors that have a 1 in their jth coordinate. Now verify the following string of equalities and inequalities:

$$\log M = H(\mathbf{X}) \leq \sum_{j=1}^{n} H(X_j) = \sum_{j=1}^{n} H(p_j) \leq nH(p).\Big]$$

(This result due to James Massey [38].)

9.38. Returning to the codes defined in Prob. 9.35, specialize to $q = 2, n = 2^m$, and $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$ any ordering of the elements of $GF(2^m)$. Let $r/2^m = \rho$, and show that the resulting codes have

(i) length $= m2^{m+1}$,

(ii) rate $= \frac{1}{2}(\rho + \frac{1}{2^m})$,

(iii) $\frac{d_{\min}}{n} \geq (1-\rho)H_2^{-1}\left[\frac{1}{2} + \frac{\log_2(1-\rho)}{2m}\right]$.

These codes are the *Justesen codes* mentioned above. [Hint: To prove (iii), use the results of Prob. 9.37].

9.39. Finally, show that for any $0 \leq R \leq \frac{1}{2}$, there is an infinite sequence of Justesen codes over $GF(2)$ with lengths n_i, dimensions k_i, and minimum distances d_i such that:

$$\begin{aligned} \lim_{i\to\infty} n_i &= \infty, \\ \lim_{i\to\infty} k_i/n_i &= R, \\ \lim_{i\to\infty} \sup\ d_i/n_i &= H_2^{-1}(1/2)\cdot(1-2R) \\ &= 0.110028(1-2R). \end{aligned}$$

9.40. Show that $\bar{d}_H$, as defined in the proof of Theorem 9.11, is a bona fide metric (Cf. Problem 7.4).

9.41. For the $(7,3)$ code of Example 9.9, find a vector $y \in \{0,1,*\}^7$ for which $\min_i \bar{d}_H$ is as large as possible.

9.42. For a given value of d, how many pairs of nonnegative integers (e_0, e_1) are there such that $e_0 + 2e_1 \leq d-1$?

9.43. If m and n are positive integers such that $m+n$ is even, show that

$$\lfloor \frac{m}{2} \rfloor + \lceil \frac{n}{2} \rceil = \frac{m+n}{2}.$$

(See the remarks immediately preceeding Eq. (9.84).

9.44. Consider the $(7,4)$ Hamming code of Example 7.3. The code has $d_{\min} = 3$ and so by Theorem 9.11, it is capable of correcting any pattern of e_0 erasures and e_1 errors, provided $e_0 + 2e_1 \leq 2$. Bearing this in mind, decode (if possible) each of the words in parts (a), (b), and (c).

(a) [1 1 1 0 0 $*$ 0]

(b) [0 $*$ 1 1 1 0 1]

(c) [0 1 $*$ 1 0 $*$ 1]

(d) If $\mathbf{R}$ is a randomly chosen vector of length 7 containing exactly one erasure, what is the probability that it will be uniquely decoded by the MDD decoding algorithm introduced in the proof of Theorem 9.11?

9.45. Show that if $\sigma_1(x)$ is the error locator polynomial, and $\omega(x)$ is the errors and erasures evaluator polynomial for RS errors–and–erasures decoding (see equations (9.77) and (9.81)), then $\gcd(\sigma_1, \omega) = 1$.

9.46. Investigate the probability of decoder error for a Reed-Solomon decoder under the following circumstances. (Note that in both parts (a) and (b), $e_0 + 2e_1 > r$, so that the hypotheses of Theorem 9.11 are violated.)

(a) r erasures and 1 error.

(b) $r-1$ erasures and 1 error.

(c) $r+1$ erasures, no errors.

9.47. Consider the $(15,7)$ RS code over $GF(16)$ (generated by a primitive root satisfying $\alpha^4 = \alpha + 1$), and decode the following received word.

$$\mathbf{R} = [\alpha^{13}\ 1\ *\ \alpha^{10}\ \alpha^{12}\ \alpha^6\ *\ \alpha^5\ \alpha^{13}\ *\ \alpha^1\ \alpha^8\ \alpha^7\ \alpha^2\ \alpha^9\ ].$$

9.48. Using the suggestions at the end of Section 9.7, decode the following noisy vector from the $(15,5)$ BCH code with generator polynomial $g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x^2 + 1$ (Cf. Example 9.1).

$$R = [1\ 1\ *\ 0\ 0\ 0\ *\ 0\ 0\ 0\ 1\ *\ 1\ 0\ 1\ ].$$

(a) Use the time domain completion.

(b) Use the frequency-domain completion.

9.49. Consider an (n,k) linear code with $d_{\min} = d$. If there are no errors, Theorem 9.11 guarantees that the code is capable of correcting any pattern of up to $d-1$ erasures. Show that this result cannot be improved, by showing that there is at least one set of d erasures that the code isn't capable of correcting.

9.50. In Section 8.4, we considered codes capable of correcting single bursts of errors. It turns out that it is very much easier to correct single bursts of *erasures*. After doing this problem, you will agree that this is so.

(a) If C is an (n,k) linear code which is capable of correcting any erasure burst of length b or less, show that $n-k \geq b$. (Cf. The Reiger bound, Corollary to Theorem 8.10).

(b) Show that any (n,k) cyclic code is capable of correcting any erasure burst of length $n-k$ or less.

(c) Consider the $(7,3)$ cyclic code of Example 8.2. Correct the following codewords, which have suffered erasure bursts of length 4: $(10****0)$, $(****101)$, $(*101**)$.

9.51. When a linear code has suffered erasures *but no errors*, there is a very simple general technique for correcting the erasures, which we shall develop in this problem. The idea is to replace each of the erased symbols with a distinct indeterminate, and then to solve for the indeterminates, using the parity-check matrix. For example, consider the $(7,4)$ binary Hamming code of Example 7.3. The code has $d_{\min} = 3$ and so by Theorem 9.11, it is capable of correcting any pattern of two or fewer erasures, provided there are no errors. If say the received word is $R = (1\ *\ 1\ *\ 1\ 0\ 1\ )$, we replace the two erased symbols with indeterminates x and y, thus obtaining $R = (1\ x\ 1\ y\ 1\ 0\ 1\ )$.

(a) Use the fact that every codeword in the $(7,4)$ Hamming code satisfies the equation $HC^T = 0$, where H is the parity-check matrix given in Section 7.4, to obtain three simultaneous linear equations in the indeterminates x and y, and solve these equations, thus correcting the erasures.

(b) If there were *three* erased positions instead of only two, we could use the same technique to obtain three equations in the three indeterminates representing the erasures. We could then solve these equations for the indeterminates, thus correcting three erasures. Yet Theorem 9.11 only guarantees that the code is capable of correcting two erasures. What's wrong here?

9.52. In this problem, we consider an alternative approach to correcting erasures and errors, which involves the idea of "guessing" the values of the erasures.

(a) Assume first that the code is binary, i.e., the field F in Theorem 9.11 is $GF(2)$. Suppose we have a code C with mimimum distance d, and that we have received a noisy codeword containing e_0 erasures and e_1 errors, with $e_0 + 2e_1 \leq d - 1$. Suppose that we change all the erased positions to 0, and then try to decode the word, using an errors-only decoding algorithm capable of correcting any pattern of up to $(d-1)/2$ errors. If the decoder succeeds, we stop. Otherwise, we try again, this time assuming that all the erased positions are 1. Show that this procedure, i.e., guessing that the erasures are all 0s, and then guessing that they are all 1s, will always succeed in correcting the errors and erasures.

(b) Illustrate the decoding technique suggested in part (a) by decoding the word $[1110 * 01]$ from the $(7, 3)$ binary cyclic code. (Cf. Example 9.9).

(c) Does this "guessing" procedure work for nonbinary fields? In particular, how could you modify it to work over the ternary field $GF(3)$?

9.53. Consider the $(7, 3)$ binary cyclic code with generator polynomial $g(x) = x^4 + x^3 + x^2 + 1$, as described, e.g., in Example 9.9. It has $d_{\min} = 4$, and so by Theorem 9.10, it can correct any pattern of up to 3 erasures (if no attempt is made to correct errors as well). It can also, however, correct some, thought not all, patterns of *four* erasures. In this problem, you are asked to investigate the $\binom{7}{4} = 35$ possible erasure patterns of size four, and to determine which of them are correctable. In particular, please find the *number* of weight four erasure patterns that are correctable.

9.54. This problem concerns the probability that a randomly selected vector from $GF(q)^n$ will be decoded by a decoder for a Reed-Solomon.

(a) Derive the following formula for the fraction of the total "volume" of $GF(q)^n$ occupied by nonoverlapping Hamming spheres of radius t around the codewords in an (n, k) code over $GF(q)$:

$$\frac{q^k \sum_{i=0}^{t} \binom{n}{i} (q-1)^i}{q^n}.$$

(b) Use the formula from part a to compute the limit, for a fixed value of t, as $q \to \infty$, of the probability that a randomly selected word of length $q - 1$ will fall within Hamming distance t or less of some codeword in a t error correcting RS code of length $q - 1$ over $GF(q)$. (Assume that the code's redundancy is $r = 2t$.)

9.55. Let C be an (n, k) binary cyclic code, with generator polynomial $g(x)$ and parity-check polynomial $h(x)$.

(a) Show that if $h(1) \neq 0$, then every codeword has even weight.

(b) Show that if there is no pair (θ_1, θ_2) of roots of $h(x)$ such that $\theta_1 \theta_2 = 1$, then every codeword's weight is divisible by four. (Hint: This is a generalization of the result in Lemma 9.5.)

9.56. In the text we proved that the $(23, 12)$ Golay code had $d_{\min} \geq 7$. Show that in fact, $d_{\min} = 7$ for this code. Do this in two ways:

(a) By examining the generator polynomial $g(x)$.

(b) By showing that *any* $(23, 12)$ binary linear code must have $d_{\min} \leq 7$.

9.57. Show that there is no $(90, 78)$ binary linear code with $d_{\min} = 5$, i.e., a perfect double-error-correcting code of length 90, despite the fact that $1 + \binom{90}{1} + \binom{90}{2} = 2^{12}$. (Hint: Let r denote the number of 12-bit syndromes of odd weight corresponding to one-bit errors . Show that the number of odd weight syndromes corresponding to two-bit errors is $r(90 - r)$, and attempt to determine r.)

9.58. Show that if C is a $(23, 12)$ binary linear code, that its minimum distance must be ≤ 7. (Note: This result, combined with Theorem 9.6, shows that in fact $d_{\min} = 7$ for the Golay code, although since the generator polynomial $g(x)$ as defined in Eq. (9.83) has weight 7, there is an easier proof!)

9.59. The (23,12) binary Golay code defined in Section 9.8, when combined with syndrome table lookup decoding, has the property that every error pattern of weight ≤ 3 will be corrected.

(a) Describe in detail what the decoder will do if the error pattern has weight 4.

(b) What if the error pattern has weight 5?

(c) Generalize the results of parts (a) and (b). For each integer t in the range $4 \leq t \leq 23$, what will the decoder do, if the error pattern has weight t?

9.60. Prove Theorem 9.14.

9.61. This problem concerns the number of codewords of weight 8 in the $(24, 12)$ extended Golay code.

(a) Show that the number of codewords of weight 8 is not zero. Do this in two ways: (1) By examining the generator polynomial $g(x)$, for the original $(23, 12)$ Golay code; (2) By showing that *any* $(24, 12)$ binary linear code must have $d_{\min} \leq 8$.

(b) Given [12] that the code contains exactly 759 words of weight 8, show that for any subset $\{i_1, \ldots, i_5\}$ of five elements from $\{0, 1, \ldots, 23\}$ there is exactly one codeword of weight 8 which is 1 at these five coordinate positions.

9.62. This problem concerns the error *detecting* capabilities of the extended $(24, 12)$ Golay code. (In parts (a) and (b), we assume that the code is being used to correct all error patterns of weight three or less.)

(a) Show that the code can detect all error patterns of weight 4.

(b) Given that the weight enumerator of the code is

$$1 + 759x^8 + 2576x^{12} + 759x^{16} + x^{24},$$

for each e in the range $4 \leq e \leq 24$, compute the number of error patterns of weight e that the code will detect.

(c) Now assume the decoder only attempts to correct error patterns of weight *two* or less, and repeat part (b).

(d) Now assume the decoder only attempts to correct error patterns of weight *one* or less, and repeat part (b).

(e) Finally, assume the decoder is used in a *detect only mode*, i.e, if the syndrome is zero, the received word is accepted as correct, but otherwise it is rejected. Repeat part (b).

9.63. In this problem, we will briefly consider encoders for the $(23, 12)$ and $(24, 12)$ Golay codes.

(a) Design a shift register encoder for the $(23, 12)$ Golay code.

(b) By modifying your design in part (a), or otherwise, come up with a design for an encoder for the $(24, 12)$ extended Golay code.

9.64. Discuss the size and complexity of a syndrome table lookup decoder for the $(24, 12)$ Golay code.

In Probs. 9.65–9.68 we will investigate the ternary Golay code. Observe that in the vector space $GF(3)^{11}$ *a Hamming sphere of radius 2 contains*

$$1 + 2\binom{11}{1} + 4\binom{11}{2} = 243 = 3^5$$

vectors. This suggests that it might be possible to perfectly pack $GF(3)^{11}$ *with* $729 = 3^6$ *spheres of radius 2. The ternary Golay code does this. It is an (11,6) linear code over* $GF(3)$ *whose codewords, when taken as sphere centers, produce such a packing. The code is defined as follows. Since* $3^5 - 1 = 11 \cdot 22$*, it follows that* $GF(3^5)$ *contains a primitive 11th root of unity, which we shall call* β*. Since over* $GF(3)$ *the factorization of* $x^{11} - 1$ *is* $x^{11} - 1 = (x-1)g(x)\tilde{g}(x)$*, where* $g(x) = x^5 + x^4 - x^3 + x^2 - 1$ *and* $\tilde{g}(x) = x^5 - x^3 + x^2 - x - 1^{11}$*, we may assume that* β *is a zero of* $g(x)$*. The (11,6) ternary Golay code is then defined to be the cyclic code with generator polynomial* $g(x)$*. To show that the spheres of radius 2 around the 729 codewords are disjoint, we must show that the minimum Hamming distance between codewords is* ≥ 5*, that is, every nonzero codeword has weight* ≥ 5*. The following problems contain a proof of this fact.*[12]

9.65. Show that the code's minimum weight is ≥ 4. [Hint: Use the BCH argument, Theorem 9.3.].

9.66. Show that if $C_0 + C_1 + \cdots + C_{10} = 0$, the codeword $\mathbf{C} = (C_0, C_1, \ldots, C_{10})$ has Hamming weight divisible by 3. [Hint: See Lemma 3, Section 9.8].

9.67. If, on the other hand, $C_0 + C_1 + \cdots + C_{10} = \alpha \neq 0$, show that $(C_0 + \alpha, C_1 + \alpha, \ldots, C_{10} + \alpha)$ is a codeword and has weight divisible by 3.

9.68. Use the preceding results to show that the code contains no codewords of weight 4, 7, or 10. [Hint: If the weight is 4, by appropriate scalar multiplication the nonzero components can be transformed to either (1,1,1,1) or (1,1,1,-1).]