EMS Textbooks in Mathematics

Jørn Justesen
Tom Høholdt

A Course In Error-Correcting Codes

Second edition

European Mathematical Society

EMS Textbooks in Mathematics

EMS Textbooks in Mathematics is a series of books aimed at students or professional mathematicians seeking an introduction into a particular field. The individual volumes are intended not only to provide relevant techniques, results, and applications, but also to afford insight into the motivations and ideas behind the theory. Suitably designed exercises help to master the subject and prepare the reader for the study of more advanced and specialized literature.

Jørn Justesen and Tom Høholdt, *A Course In Error-Correcting Codes*
Markus Stroppel, *Locally Compact Groups*
Peter Kunkel and Volker Mehrmann, *Differential-Algebraic Equations*
Dorothee D. Haroske and Hans Triebel, *Distributions, Sobolev Spaces, Elliptic Equations*
Thomas Timmermann, *An Invitation to Quantum Groups and Duality*
Oleg Bogopolski, *Introduction to Group Theory*
Marek Jarnicki and Peter Pflug, *First Steps in Several Complex Variables: Reinhardt Domains*
Tammo tom Dieck, *Algebraic Topology*
Mauro C. Beltrametti et al., *Lectures on Curves, Surfaces and Projective Varieties*
Wolfgang Woess, *Denumerable Markov Chains*
Eduard Zehnder, *Lectures on Dynamical Systems. Hamiltonian Vector Fields and Symplectic Capacities*
Andrzej Skowroński and Kunio Yamagata, *Frobenius Algebras I. Basic Representation Theory*
Piotr W. Nowak and Guoliang Yu, *Large Scale Geometry*
Joaquim Bruna and Juliá Cufí, *Complex Analysis*
Eduardo Casas-Alvero, *Analytic Projective Geometry*
Fabrice Baudoin, *Diffusion Processes and Stochastic Calculus*
Olivier Lablée, *Spectral Theory in Riemannian Geometry*
Dietmar A. Salamon, *Measure and Integration*
Andrzej Skowroński and Kunio Yamagata, *Frobenius Algebras II. Tilted and Hochschild Extension Algebras*

Jørn Justesen
Tom Høholdt

A Course In Error-Correcting Codes

Second edition

European Mathematical Society

Authors:

Jørn Justesen
Department of Photonics Engineering
Technical University of Denmark
2800 Kgs. Lyngby
Denmark

E-mail: jorn@justesen.info

Tom Høholdt
Department of Applied Mathematics and Computer Science
Technical University of Denmark
2800 Kgs. Lyngby
Denmark

E-mail: tomh@dtu.dk

2010 Mathematics Subject Classification: 94-01;12-01

Key words: Error-correcting codes, Reed–Solomon codes, convolutional codes, product codes, graph codes, algebraic geometry codes

First edition published in 2004 by the European Mathematical Society

ISBN 978-3-03719-179-8

The Swiss National Library lists this publication in The Swiss Book, the Swiss national bibliography, and the detailed bibliographic data are available on the Internet at http://www.helveticat.ch.

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in other ways, and storage in data banks. For any kind of use permission of the copyright owner must be obtained.

© 2017 European Mathematical Society

Contact address:

European Mathematical Society Publishing House
Seminar for Applied Mathematics
ETH-Zentrum SEW A21
CH-8092 Zürich
Switzerland

Phone: +41 (0)44 632 34 36
Email: info@ems-ph.org
Homepage: www.ems-ph.org

Typeset using the authors' TEX files: le-tex publishing services GmbH, Leipzig, Germany
Printing and binding: Beltz Bad Langensalza GmbH, Bad Langensalza, Germany
∞ Printed on acid free paper
9 8 7 6 5 4 3 2 1

Preface to the second edition

The new edition of the book reflects the fact that both the theory and the applications have evolved over the last twelve years. Some topics have been removed, others expanded, and new codes were added.

We want to thank Shorijo Sakata, Knud J. Larsen, and Johan S. Rosenkilde Nielsen for their help in improving the text.

We have not included references to scientific literature, industry standards, or internet resources. As both theory and applications have diversified, we can no longer point to a few useful starting points for further studies, but the interested reader should have no difficulty in searching for additional information about specific subjects.

Washington D.C., Virum, December 2016 Jørn Justesen, Tom Høholdt

Contents

Chapter 1

Block Codes for Error Correction

This chapter introduces the fundamental concepts of *block codes* and error correction. Codes are used for several purposes in communication and storage of information. In this book we discuss only error-correcting codes, i.e., in the received message, some symbols are changed, and it is the objective of the coding to enable these errors to be corrected. Most practical codes are described as vector spaces, and familiarity with some concepts from linear algebra is assumed. In particular this includes bases of vector spaces, matrices, and systems of linear equations. We describe the first approach to error correction, the Hamming codes for correcting single errors (1950). Initially we let all codes be binary, i.e., the symbols are only 0 and 1, since this is both the simplest and the most important case, but later on we will consider other symbol alphabets as well.

1.1 Linear codes and vector spaces

Definition 1.1.1. *A* block code *C is a set of M* codewords

$$\begin{aligned} C &= \{c_1, c_2, \ldots, c_M\} \\ c_i &= (c_{i0}, c_{i1}, \ldots, c_{in-1}) \end{aligned}$$

where the codewords are n-tuples and we refer to n as the length *of the code.*

The elements c_{ij} belong to a finite alphabet of q symbols. For the time being we consider only *binary* codes, i.e., the alphabet is $\{0, 1\}$, but later we shall consider larger alphabets. The alphabet will be given the structure of a field, which allows us to do computations on the codewords. The theory of finite fields is presented in Chapter 2.

Example 1.1.1. The *binary field* $\mathbb{F}_2$.
The elements are denoted 0 and 1 and we do addition and multiplication according to the following rules: $0 + 0 = 0, 1 + 0 = 0 + 1 = 1, 1 + 1 = 0$ and $0 \cdot 0 = 1 \cdot 0 = 0 \cdot 1 = 0, 1 \cdot 1 = 1$. One may say that addition is performed modulo 2, and in some contexts the logical operation + is referred to as exclusive or (xor).

With few exceptions we shall consider only linear codes, which are described as vector spaces.

If $\mathbb{F}$ is a field and n a natural number, then the elements of $\mathbb{F}^n$ can be seen as forming a vector space $V = (\mathbb{F}^n, +, \mathbb{F})$ where

$$\begin{aligned} x, y &\in \mathbb{F}^n, \\ x &= (x_0, x_1, \ldots, x_{n-1}), \\ y &= (y_0, y_1, \ldots, y_{n-1}), \\ x + y &= (x_0 + y_0, x_1 + y_1, \ldots, x_{n-1} + y_{n-1}), \\ fx &= (fx_0, fx_1, \ldots, fx_{n-1}), \text{ where } f \in \mathbb{F}. \end{aligned}$$

This may be a familiar concept when $\mathbb{F}$ is the field of real numbers, but we shall use other fields, in particular $\mathbb{F}_2$, in the following examples. Note that in a vector space one has the notion of a *basis*, i.e., a maximal set of linearly independent vectors, and that any vector is a linear combination of elements from the basis. The *dimension* of a vector space is the number of elements in a basis.

In V we have an inner product defined by

$$x \cdot y = x_0 y_0 + x_1 y_1 + \cdots + x_{n-1} y_{n-1}$$

So the value of the inner product is an element of the field $\mathbb{F}$. If two vectors x and y satisfy $x \cdot y = 0$, they are said to be *orthogonal*. Note that we can have $x \cdot x = 0$ with $x \neq 0$ if $\mathbb{F} = \mathbb{F}_2$.

Definition 1.1.2. *A* linear (n, k) block code C *is a k-dimensional subspace of the vector space V.*

The code is called linear since if C is a subspace, we have

$$\begin{aligned} c_i \in C \wedge c_j \in C &\Longrightarrow c_i + c_j \in C, \\ c_i \in C \wedge f \in \mathbb{F} &\Longrightarrow fc_i \in C. \end{aligned}$$

In particular the zero vector is always a codeword. The number of codewords is $M = q^k$, where q is the number of elements in the field $\mathbb{F}$.

Example 1.1.2. An $(n, k) = (7, 4)$ binary block code.
Consider the code consisting of the following 16 vectors:

0000000	1111111
1000110	0111001
0100011	1011100
0010101	1101010
0001111	1110000
1100101	0011010
1010011	0101100
1001001	0110110

It may readily be verified that the sum of any two codewords is again a codeword.

When a code is used for communication or storage, the *information* may be assumed to be a long sequence of binary digits. The sequence is segmented into blocks of length k. We may think of such a block of information as a binary vector u of length k. We shall therefore need an encoding function that maps k-vectors onto codewords. In a *systematic encoding*, we simply let the first k coordinates be equal to the information symbols. The remaining $n-k$ coordinates are sometimes referred to as *parity check symbols* and we shall justify this name below.

Instead of listing all the codewords, a code may be specified by a *basis* of k linearly independent codewords.

Definition 1.1.3. *A* generator matrix *G of an (n,k) code C is a $k \times n$ matrix whose rows are linearly independent codewords.*

If the information is the vector u of length k, we can state the *encoding rule* as

$$c = uG \tag{1.1}$$

Example 1.1.3. A basis for the (7,4) code.
We may select four independent vectors from the list above to give the generator matrix

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

The same code, in the sense of a set of words or a vector space, may be described by different generator matrices or bases of the vector space. We usually just take one that is convenient for our purpose. However, G may also be interpreted as specifying a particular encoding of the information. Thus row operations on the matrix do not change the code, but the modified G represents a different encoding mapping. Since G has rank k, we can obtain a convenient form of G by row operations in such a way that k columns form the $k \times k$ identity matrix I. We often assume that this matrix can be chosen as the first k columns and write the generator matrix as

$$G = (I, A).$$

This form of the generator matrix gives a systematic encoding of the information.

We may now define a *parity check* as a vector, h, of length n which satisfies

$$Gh^T = 0,$$

where h^T denotes the transpose of h.

The parity check vectors form again a subspace of V of dimension $n-k$.

Definition 1.1.4. *A* parity check matrix *H for an (n,k) code C is an $(n-k) \times n$ matrix whose rows are linearly independent parity checks.*

So if G is a generator matrix for the code and H is a parity check matrix, we have

$$GH^T = 0,$$

where 0 is a $k \times (n-k)$ matrix of zeroes.

From the systematic form of the generator matrix we can find H as

$$H = (-A^T, I) \tag{1.2}$$

where I now is an $(n-k) \times (n-k)$ identity matrix. If such a parity check matrix is used, the last $n-k$ elements of the codeword are given as linear combinations of the first k elements. This justifies calling the last symbols *parity check symbols*.

Definition 1.1.5. *Let H be a parity check matrix for an (n,k) code C and let $r \in \mathbb{F}^n$, then the* syndrome $s = \mathrm{syn}(r)$ *is given by*

$$s = Hr^T. \tag{1.3}$$

We note that if the received word is $r = c + e$, where c is a codeword and e is the *error pattern* (also called the *error vector*), then

$$s = H(c+e)^T = He^T. \tag{1.4}$$

The term syndrome refers to the fact that s reflects the error in the received word. The codeword itself does not contribute to the syndrome, and for an error-free codeword $s = 0$.

It follows from the above definition that the rows of H are orthogonal to the codewords of C. The code spanned by the rows of H is what is called the *dual code* $C^\perp$, defined by

$$C^\perp = \{x \in \mathbb{F}^n \mid x \cdot c = 0 \; \forall c \in C\}$$

It is often convenient to talk about the *rate* of a code, $R = \frac{k}{n}$. Thus the dual code has rate $1 - R$.

Example 1.1.4. A parity check matrix for the (7,4) code.
We may write the parity check matrix as

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

1.2 Minimum distance and minimum weight

In order to determine the error correcting capability of a code, we will introduce the following useful concept

Definition 1.2.1. *The* Hamming weight *of a vector x, denoted $w_{\mathrm{H}}(x)$, is equal to the number of nonzero coordinates.*

Note that the Hamming weight is often simply called weight.

For a received vector $r = c_j + e$, the number of errors is the Hamming weight of e.

We would like to be able to correct all error patterns of weight $\leq t$ for some t, and use the following definition to explain what we mean by that.

Definition 1.2.2. *A code is t-error correcting if for any two codewords $c_i \neq c_j$, and for any error patterns e_1 and e_2 of weight $\leq t$, we have $c_i + e_1 \neq c_j + e_2$.*

This means that it is not possible to get a certain received word from making at most t errors in two different codewords.

A more convenient way of expressing this property uses the notion of *Hamming distance*

Definition 1.2.3. *The* Hamming distance *between two vectors x and y, denoted $d_{\mathrm{H}}(x, y)$, is the number of coordinates where they differ.*

It is not hard to see that

$$\begin{aligned} d_{\mathrm{H}}(x, y) &= 0 \iff x = y, \\ d_{\mathrm{H}}(x, y) &= d_{\mathrm{H}}(y, x), \\ d_{\mathrm{H}}(x, y) &\leq d_{\mathrm{H}}(x, z) + d_{\mathrm{H}}(z, y). \end{aligned}$$

So d_{H} satisfies the usual properties of a distance, in particular the third property, i.e., the *triangle inequality*. With this distance V becomes a *metric space*.

As with the weight, the Hamming distance is often simply called distance, hence we will in the following often skip the subscript H on the weight and the distance.

Definition 1.2.4. *The* minimum distance *of a code, d, is the minimum Hamming distance between any pair of different codewords.*

So the minimum distance can be calculated by comparing all pairs of codewords, but for linear codes this is not necessary, since we have

Lemma 1.2.1. *In an (n, k) code the minimum distance is equal to the minimum weight of a nonzero codeword.*

Proof. It follows from the definitions that $w(x) = d(0, x)$ and that $d(x, y) = w(x - y)$. Let c be a codeword of minimum weight. Then $w(c) = d(0, c)$ and since 0 is a codeword, we have $d_{\min} \leq w_{\min}$. On the other hand, if c_1 and c_2 are codewords at minimum distance, we have $d(c_1, c_2) = w(c_1 - c_2)$ and since $c_1 - c_2$ is again a codeword, we get $w_{\min} \leq d_{\min}$. Combining the two inequalities, we get the result. $\square$

Based on this observation we can now prove

Theorem 1.2.1. *An (n,k) code is t-error correcting if and only if $t < \frac{d}{2}$.*

Proof. Suppose $t < \frac{d}{2}$ and we have two codewords c_i and c_j and two error patterns e_1 and e_2 of weight $\leq t$, such that $c_i + e_1 = c_j + e_2$. Then $c_i - c_j = e_2 - e_1$, but $w(e_2 - e_1) = w(c_i - c_j) \leq 2t < d$, contradicting the fact that the minimum weight is d. On the other hand, suppose that $t \geq \frac{d}{2}$ and let c have weight d. Change $\lceil \frac{d}{2} \rceil$ of the nonzero positions to zeroes to obtain y. Then $d(0, y) \leq d - \lceil \frac{d}{2} \rceil \leq t$ and $d(c, y) = \lceil \frac{d}{2} \rceil \leq \lceil \frac{2t}{2} \rceil = t$, but $0 + y = c + (y - c)$, so the code is not t-error correcting. $\square$

Example 1.2.1. The minimum distance of the (7, 4) code from Example 1.1.2 is 3 and $t = 1$.

Finding the minimum distance of a code is difficult in general, but the parity check matrix can sometimes be used. This is based on the following simple observation.

Lemma 1.2.2. *Let C be an (n,k) code and H a parity check matrix for C. Then, if j columns are linearly dependent, C contains a codeword with nonzero elements in some of the corresponding positions, and if C contains a word of weight j, then there exist j linearly dependent columns of H.*

This follows directly from the definition of matrix multiplication, since for a codeword c we have $Hc^T = 0$.

Lemma 1.2.3. *Let C be an (n,k) code with parity check matrix H. The minimum distance of C equals the minimum number of linearly dependent columns of H.*

This follows immediately from Lemma 1.2.2 and Lemma 1.2.1.

For a *binary* code this means that $d \geq 3 \Leftrightarrow$ *the columns of H are distinct and nonzero.*

One of our goals is, for given n and k, to construct codes with large minimum distance. The following important theorem ensures the existence of certain codes, which in turn can serve as a reference for other codes.

Theorem 1.2.2. *The Varshamov–Gilbert bound.*
There exists a binary linear code of length n, with at most m linearly independent parity checks and minimum distance at least d, if

$$1 + \binom{n-1}{1} + \cdots + \binom{n-1}{d-2} < 2^m.$$

Proof. We shall construct an $m \times n$ matrix such that no $d-1$ columns are linearly dependent. The first column can be any nonzero m-tuple. Now suppose we have chosen i columns so that no $d-1$ are linearly dependent. There are

$$\binom{i}{1} + \cdots + \binom{i}{d-2}$$

linear combinations of these columns taken $d-2$ or fewer at a time. If this number is smaller than $2^m - 1$ we can add an extra column such that still $d-1$ columns are linearly independent. See Fig. 1.1. □

For large n, the Gilbert–Varshamov bound indicates that good codes exist, but there is no known practical way of constructing such codes. It is also not known if long binary codes can have distances greater than indicated by the Gilbert–Varshamov bound. Short codes can have better minimum distances. The following examples are particularly important.

Definition 1.2.5. *A* binary Hamming code *is a code whose parity check matrix has all nonzero binary m-vectors as columns.*

So the length of a binary Hamming code is $2^m - 1$ and the dimension is $2^m - 1 - m$, since it is clear that the parity check matrix has rank m. The minimum distance is at least 3 by the above, and it it easy to see that it is exactly 3. The columns can be ordered in different ways: a convenient method is to represent the natural numbers from 1 to $2^m - 1$ as binary m-tuples.

Example 1.2.2. The $(7, 4)$ code is a binary Hamming code with $m = 3$.

Definition 1.2.6. *An* extended binary Hamming *code is obtained by adding a zero column to the parity check matrix of a Hamming code and then adding a row of all* 1*s.*

This means that the length of the extended code is 2^m. The extra parity check is $c_0 + c_1 + \cdots + c_{n-1} = 0$, and therefore all codewords in the extended code have even weight and the minimum distance is 4. Thus the parameters (n, k, d) of the binary extended Hamming code are $(2^m, 2^m - m - 1, 4)$.

Definition 1.2.7. *A* biorthogonal code *is a dual of a binary extended Hamming code.*

The biorthogonal code has length $n = 2^m$ and dimension $k = m + 1$. It can be seen that the code consists of the all 0s vector, the all 1s vector, and $2n - 2$ vectors of weight $\frac{n}{2}$ (See problem 1.5.8).

Example 1.2.3. The $(16, 11, 4)$ binary extended Hamming code and its dual. Based on the definition above we can get a parity check matrix for the $(16, 11, 4)$ code as

$$\begin{bmatrix} 1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0 \\ 0&1&1&0&0&1&1&0&0&1&1&0&0&1&1&0 \\ 0&0&0&1&1&1&1&0&0&0&0&1&1&1&1&0 \\ 0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&0 \\ 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \end{bmatrix}$$

The same matrix is a generator matrix for the $(16, 5, 8)$ biorthogonal code.

1.3 Syndrome decoding and the Hamming bound

When a code is used for correcting errors, one of the important problems is the design of a *decoder*. One can consider this as a mapping from $\mathbb{F}_q^n$ into the code C, as an algorithm, or sometimes even as a physical device. We will usually see a decoder as a mapping or as an algorithm. One way of stating the objective of the decoder is: for a received vector r, select as the transmitted codeword c a codeword that minimizes $d(r, c)$. This is called *maximum likelihood decoding* for reasons we will explain in Chapter 3. It is clear that if the code is t-error correcting and $r = c + e$ with $w(e) \leq t$, then the output of such a decoder is c.

It is often difficult to design a maximum likelihood decoder, but if we only want to correct t errors, where $t < \frac{d}{2}$, it is sometimes easier to get a good algorithm.

Definition 1.3.1. *A* minimum distance decoder *is a decoder that, given a received word r, selects the codeword c that satisfies $d(r, c) < \frac{d}{2}$ if such a word exists, and otherwise declares failure.*

It is obvious that there can be at most one codeword within distance $\frac{d}{2}$ from a received word.

Using the notion of syndrome from the previous section we can think of a decoder as a mapping from syndromes to error patterns. Thus for each syndrome we should choose an error pattern with the smallest number of errors, and if there are several error patterns of equal weight with the same syndrome we may choose one of these arbitrarily. Such a *syndrome decoder* is not only a useful concept, but may also be a reasonable implementation if $n - k$ is not too large.

Definition 1.3.2. *Let C be an (n, k) code and $a \in \mathbb{F}^n$. The* coset containing *a is the set $a + C = \{a + c \mid c \in C\}$.*

If two words x and y are in the same coset and H is a parity check matrix of the code, we have $Hx^T = H(a+c_1)^T = Ha^T = H(a+c_2)^T = Hy^T$, so the two words have the same syndrome. On the other hand, if two words x and y have the same syndrome, then $Hx^T = Hy^T$ and therefore $H(x-y)^T = 0$, so this is the case if and only if $x-y$ is a codeword and therefore x and y are in the same coset. We therefore have

Lemma 1.3.1. *Two words are in the same coset if and only if they have the same syndrome.*

The cosets form a partition of the space $\mathbb{F}^n$ into q^{n-k} classes, each containing q^k elements.

This gives an alternative way of describing a syndrome decoder: Let r be a received word. Find a vector f of smallest weight in the coset containing r (i.e., $\text{syn}(f) = \text{syn}(r)$) and decode into $r-f$. A word of smallest weight in a coset can be found once and for all; such a word is called a *coset leader*. With a list of syndromes and corresponding coset leaders, syndrome decoding can be performed as follows: Decode into $r-f$, where f is the coset leader of the coset corresponding to $\text{syn}(r)$. In this way we actually do maximum likelihood decoding and can correct q^{n-k} error patterns. If we only list the cosets where the coset leader is unique and have the corresponding syndromes, we do minimum distance decoding.

We illustrate this in

Example 1.3.1. Let C be the binary $(6,3)$ code with parity check matrix

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

From the parity check matrix we see that we have the following syndromes and coset leaders:

syndrome	coset leader
$(000)^T$	000000
$(111)^T$	100000
$(101)^T$	010000
$(110)^T$	001000
$(100)^T$	000100
$(010)^T$	000010
$(001)^T$	000001
$(011)^T$	000011

In the last coset there is more than one word with weight 2, so we have chosen one of these, but for the first seven cosets the coset leader is unique.

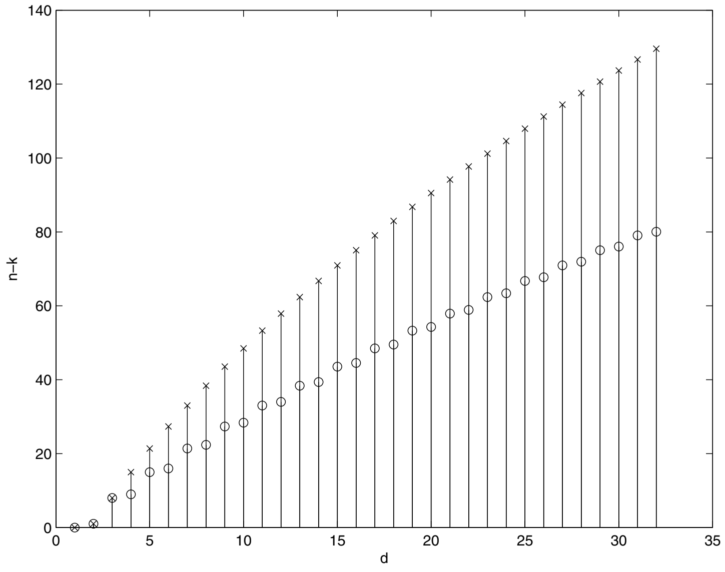

Figure 1.1. Gilbert (x) and Hamming (o) bounds on the number of parity symbols for binary codes of length 255.

This illustrates

Lemma 1.3.2. *Let C be an (n, k) code. C is t-error correcting if and only if all words of weight $\leq t$ are coset leaders.*

Proof. Suppose C is t-error correcting. Then $d = w_{\min} > 2t$. If two words of weight $\leq t$ were in the same coset their difference would be in C and have weight at most $2t$, a contradiction.

To prove the converse, suppose all words of weight $\leq t$ are coset leaders, but that c_1 and c_2 were codewords at distance $\leq 2t$. Then we can find e_1 and e_2 such that $c_1 + e_1 = c_2 + e_2$ with $w(e_1) \leq t$ and $w(e_2) \leq t$ and therefore $\text{syn}(e_1) = \text{syn}(e_2)$, contradicting the fact that e_1 and e_2 are in different cosets. □

We note that the total number of words of weight $\leq t$ is

$$1 + (q-1)\binom{n}{1} + (q-1)^2\binom{n}{2} + \cdots + (q-1)^t\binom{n}{t},$$

and that the total number of syndromes is q^{n-k}, so we have

Theorem 1.3.1. *The Hamming bound.*

If C is an (n,k) code over a field $\mathbb{F}$ with q elements that corrects t errors, then

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

The bound can be seen as an upper bound on the minimum distance of a code with given n and k, as an upper bound on k if n and d are given, or as a lower bound on n if k and d are given.

In the binary case the bound specializes to

$$\sum_{j=0}^{t}\binom{n}{j} \le 2^{n-k}.$$

For the binary Hamming codes where $t = 1$ we have $1 + n = 2^{n-k}$. For this reason these codes are called *perfect*. See Fig. 1.1.

1.4 Weight distributions

If we need more information about the error-correcting capability of a code than what is indicated by the minimum distance, we can use the *weight distribution*.

Definition 1.4.1. *The* weight distribution *of a code is the vector $A = (A_0, A_1, \ldots, A_n)$ where A_w is the number of codewords of weight w. The* weight enumerator *is the polynomial*

$$A(z) = \sum_{w=0}^{n} A_w z^w.$$

We note that for a linear code the number A_w is also the number of codewords at distance w from a given codeword. In this sense, the geometry of the code as seen from a codeword is the same for all these.

We note the following important result on the weight distribution of dual codes.

Theorem 1.4.1. *MacWilliams relation.*
If $A(z)$ is the weight enumerator of a binary (n,k) code C, then the weight enumerator $B(z)$ of the dual code $C^{\perp}$ is given by

$$B(z) = 2^{-k}(1+z)^n A\left(\frac{1-z}{1+z}\right). \tag{1.5}$$

Because of the importance of this theorem, we give a proof even though it is quite lengthy (and may be skipped at first reading).

Proof. Let H be a parity check matrix of C. Let H_{ext} be the matrix that consists of all linear combinations of the rows of H, so H_{ext} has as rows all the codewords of $C^{\perp}$. Let $y \in \mathbb{F}_2^n$ and define the extended syndrome by

$$s_{\text{ext}} = H_{\text{ext}} y^T$$

It is obvious that

$$c \in C \iff Hc^T = 0 \iff H_{\text{ext}} c^T = s_{\text{ext}} = 0. \tag{1.6}$$

Let

$$s_{\text{ext}} = (s_1, s_2, \ldots s_{2^{n-k}})$$

and define

$$E_j = \left\{ x \in \mathbb{F}_2^n \, | s_j = 0 \right\}, \quad j = 1, 2, \ldots 2^{n-k}.$$

From (1.6) we have that

$$\begin{aligned} C &= E_1 \cap E_2 \cap \cdots \cap E_{2^{n-k}} \\ &= \mathbb{F}_2^n \setminus \left(\overline{E}_1 \cup \overline{E}_2 \cup \cdots \cup \overline{E}_{2^{n-k}} \right), \end{aligned}$$

where

$$\overline{E}_j = \mathbb{F}_2^n \setminus E_j = \left\{ x \in \mathbb{F}_2^n \, | s_j = 1 \right\}.$$

If we let

$$\overline{E} = \overline{E}_1 \cup \cdots \cup \overline{E}_{2^{n-k}},$$

we have

$$C = \mathbb{F}_2^n \setminus \overline{E}. \tag{1.7}$$

The weight enumerator of $\mathbb{F}_2^n$ is

$$\sum_{i=0}^{n} \binom{n}{i} z^i = (1+z)^n,$$

so if we let $E(z)$ denote the weight enumerator of $\overline{E}$, we have

$$A(z) = (1+z)^n - E(z).$$

Let us determine $E(z)$.

We first note that by linearity if $s_{\text{ext}} \neq 0$, then s_{ext} consists of 2^{n-k-1} 0s and 2^{n-k-1} 1s. That means that a word from $\overline{E}$ is in exactly 2^{n-k-1} of the $\overline{E}_j$s, and so

$$2^{n-k-1} E(z) = \sum_{j=1}^{2^{n-k}} E_j(z)$$

where $E_j(z)$ is the weight enumerator of $\overline{E}_j$.

Let w_j denote the Hamming weight of the j-th row of H_{ext}, then

$$\begin{aligned}
E_j(z) &= (1+z)^{n-w_j} \sum_{k=1,\text{odd}}^{w_j} \binom{w_j}{k} z^k \\
&= (1+z)^{n-w_j} \frac{1}{2}\left[(1+z)^{w_j} - (1-z)^{w_j}\right] \\
&= \frac{1}{2}(1+z)^n - \frac{1}{2}(1+z)^{n-w_j}(1-z)^{w_j}.
\end{aligned}$$

Consequently,

$$\begin{aligned}
E(z) &= (1+z)^n - \frac{1}{2^{n-k}}(1+z)^n \sum_{j=1}^{2^{n-k}} \left(\frac{1-z}{1+z}\right)^{w_j} \\
&= (1+z)^n - \frac{1}{2^{n-k}}(1+z)^n B\left(\frac{1-z}{1+z}\right), \qquad (1.8)
\end{aligned}$$

and therefore

$$A(z) = \frac{1}{2^{n-k}}(1+z)^n B\left(\frac{1-z}{1+z}\right).$$

By interchanging the roles of C and $C^{\perp}$ we get the result. □

From (1.5) we get

$$\begin{aligned}
2^k \sum_{i=0}^{n} B_i z^i &= \sum_{w=0}^{n} A_w \left(\frac{1-z}{1+z}\right)^w (1+z)^n, \\
2^k \sum_{i=0}^{n} B_i z^i &= \sum_{w=0}^{n} A_w (1-z)^w (1+z)^{n-w}, \\
2^k \sum_{i=0}^{n} B_i z^i &= \sum_{w=0}^{n} A_w \sum_{m=0}^{w} \binom{w}{m}(-z)^m \sum_{l=0}^{n-w} \binom{n-w}{l} z^l. \qquad (1.9)
\end{aligned}$$

From this we can find B if A is known and vice versa. The actual calculation can be tedious, but can be accomplished by many symbolic mathematical programs.

Example 1.4.1. (Example 1.2.3 continued) The biorthogonal $(16, 5, 8)$ code has weight enumerator

$$A(z) = 1 + 30z^8 + z^{16}.$$

The weight enumerator for the $(16, 11, 4)$ extended Hamming code can then be found using (1.9):

$$\begin{aligned} B(z) &= 2^{-5}\left((1+z)^{16} + 30(1-z)^8(1+z)^8 + (1-z)^{16}\right) \\ &= 2^{-5}\left(\sum_{j=0}^{16}\binom{16}{j}\left[z^j + (-z)^j\right] + 30\sum_{m=0}^{8}\binom{8}{m}(-z)^m\sum_{l=0}^{8}\binom{8}{l}z^l\right) \\ &= 1 + 140z^4 + 448z^6 + 870z^8 + 448z^{10} + 140z^{12} + z^{16}. \end{aligned}$$

1.5 Problems

Problem 1.5.1. Consider the binary code

$$\begin{array}{c} (0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0) \\ (0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1) \\ (1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1) \\ (1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0) \\ (1\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0) \end{array}$$

(1) Is this a linear code?

(2) Add more words such that the resulting code is linear.

(3) Determine a basis of this code.

Problem 1.5.2. Let C be the binary linear code of length 6 with generator matrix

$$G = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

(1) Determine a generator matrix for the code of the form $(I\,A)$.

(2) Determine a parity check matrix for the code $C^{\perp}$.

(3) Is (1,1,1,1,1,1) a parity check for the code?

Problem 1.5.3. Consider the linear code with the generator matrix

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

(1) Determine the dimension and the minimum distance of the code and its dual.

(2) How many errors do the two codes correct?

Problem 1.5.4. Let the columns of the parity check matrix of a code be h_1, h_2, h_3, h_4, h_5, so

$$H = \begin{bmatrix} | & | & | & | & | \\ h_1 & h_2 & h_3 & h_4 & h_5 \\ | & | & | & | & | \end{bmatrix}.$$

(1) What syndrome corresponds to an error at position j?

(2) Express $H(10011)^T$ using h_1, h_2, h_3, h_4, h_5.

(3) Show that if $(1, 1, 1, 1, 1)$ is a codeword, then $h_1 + h_2 + h_3 + h_4 + h_5 = 0$.

Problem 1.5.5. Use the Gilbert–Varshamov bound to determine k such that there exists a binary $(15, k)$ code with minimum distance 5.

Problem 1.5.6.

(1) Determine the parameters of the binary Hamming codes with $m = 3, 4, 5,$ and 8.

(2) What are the parameters of the extended codes?

Problem 1.5.7. In Example 1.1.4 we gave a parity check matrix for a Hamming code.

(1) What is the dimension and the minimum distance of the dual code?

(2) Show that all pairs of codewords in this code have the same distance.

(3) Find the minimum distance of the dual to a general binary Hamming code.

These codes are called *equidistant* because of the property noted in (2).

Problem 1.5.8. Consider the biorthogonal code $B(m)$ of length 2^m and dimension $m + 1$.

(1) Show that $B(2)$ contains (0, 0, 0, 0), (1, 1, 1, 1), and six words of weight 2.

(2) Show that $B(m)$ contains the all 0s vector, the all 1s vector, and $2n - 2$ vectors of weight $\frac{n}{2}$.

(3) Replace $\{0, 1\}$ with $\{1, -1\}$ in all the codewords and show that these are orthogonal as real vectors.

Problem 1.5.9. Let G be a generator matrix of an (n, k, d) code C where $d \geq 2$. Let G^* be the matrix obtained by deleting a column of G and C^* be the code with generator matrix G^*.

(1) What can you say about n^*, k^* and d^* ?

This process of obtaining a shorter code is called *puncturing*.

Another way of getting a shorter code from an (n, k, d) code C is to force an information symbol, x say, to be zero and then delete that position.

This process is called *shortening*.

(2) What are the parameters of the shortened code?

Problem 1.5.10. Let H be a parity check matrix for an (n, k) code C. We construct a new code C_{ext} of length $n + 1$ by defining $c_n = c_0 + c_1 + \cdots + c_{n-1}$. What can be said about the dimension, the minimum distance, and the parity check matrix of C_{ext}?

Problem 1.5.11. Let C be a binary (n, k) code.

(1) Show that the number of codewords that have 0 at position j is either 2^k or 2^{k-1}.

(2) Show that $\sum_{c\in C} w(c) \le n \cdot 2^{k-1}$ (Hint: Write all the the codewords as rows of a $2^k \times n$ matrix.)

(3) Prove that

$$d_{\min} \le \frac{n \cdot 2^{k-1}}{2^k - 1}.$$

This is the so-called *Plotkin bound*.

Problem 1.5.12. Let C be the code with generator matrix

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

(1) Determine a parity check matrix for C.

(2) Determine the minimum distance of C.

(3) Determine the cosets that contain (111111), (110010), and (100000), respectively, and find for each of these the coset leader.

(4) Decode the received word (111111).

Problem 1.5.13. Let C be the code of length 9 with parity check matrix

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

(1) What is the dimension of C?

(2) Determine the minimum distance of C.

(3) Determine coset leaders and the corresponding syndromes for at least 11 cosets.

(4) Is 000110011 a codeword?

(5) Decode the received words 110101101 and 111111111.

Problem 1.5.14. Let C be an (n, k) code with minimum distance 5 and parity check matrix H.
Is it true that $H(1110\ldots0)^T = H(001110\ldots0)^T$?

Problem 1.5.15. Determine a parity check matrix for a code that has the following coset leaders: 000000, 100000, 010000, 001000, 000100, 000010, 000001, 110000.

Problem 1.5.16. Find an upper bound on k for which there exists a $(15, k)$ code with minimum distance 5. Compare the result with Problem 1.5.5.

Problem 1.5.17. An (8, 1) code consists of the all 0s word and the all 1s word. What is the weight distribution of the dual code?

Problem 1.5.18. In this problem we investigate the existence of a binary $(31, 22, 5)$ code C.

(1) Show that the Hamming bound does not rule out the existence of such a code.

(2) Determine the number of cosets of C and determine the number of coset leaders of weight 0, 1, and 2, respectively.

(3) Determine for each of the cosets, where the coset leaders have weights 0, 1 or 2, the maximal number of words of weight 3 in such a coset.

(4) Determine for each of the cosets, where the coset leader has weight at least 3, the maximal number of words of weight 3 in such a coset.

(5) Show that this leads to a contradiction and therefore such a code does not exist!

Problem 1.5.19. Let C be the binary $(32, 16)$ code with generator matrix $G = (I, A)$ where I is a 16×16 identity matrix and

$$A = \begin{bmatrix} J & \hat{I} & \hat{I} & \hat{I} \\ \hat{I} & J & \hat{I} & \hat{I} \\ \hat{I} & \hat{I} & J & \hat{I} \\ \hat{I} & \hat{I} & \hat{I} & J \end{bmatrix},$$

where $\hat{I}$ is a 4×4 identity matrix and

$$J = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

(1) Prove that $C = C^{\perp}$, i.e., the code is self-dual. (Why is it enough to show that each row in G is orthogonal to any other row?)

(2) Determine a parity check matrix H.

(3) Find another generator matrix of the form $G' = (A', I)$.

(4) Prove that for a self-dual code the following statement is true: If two codewords have weights that are multiples of 4, that is also true for their sum.

(5) Prove that the minimum distance of the code is a multiple of 4.

(6) Prove that the minimum distance is 8.

(7) How many errors can the code correct?

(8) How many cosets does the code have?

(9) How many error patterns are there of weight $0, 1, 2$, and 3?

(10) How many cosets have coset leaders of weight at least 4?

Problem 1.5.20. *Project.*
Write a program for decoding the $(32, 16)$ code above using syndrome decoding. If more than three errors occur, the program should indicate a decoding failure and leave the received word unchanged.
We suggest that you generate a table of error patterns using the syndrome as the address. Initially the table is filled with the all 1s vector (or some other symbol indicating decoding failure). Then the syndromes for zero to three errors are calculated, and the error patterns are stored in the corresponding locations.
Assume that the bit error probability is 0.01.

(1) What is the probability of a decoding failure?

(2) Give an estimate of the probability of decoding to a wrong codeword.

Problem 1.5.21. *Project.*
Write a program that enables you to determine the weight enumerator of the binary $(32, 16)$ code above.
One way of getting all the codewords is to let the information vector run through the 2^{16} possibilities by converting the integers from 0 to $2^{16} - 1$ to 16 bit vectors and multiplying these on the generator matrix.

Problem 1.5.22. *Project.*
A way to construct binary codes of prescribed length n and minimum distance d is the following: List all the binary n vectors in some order and choose the codewords one at the time such that each of them has distance at least d from the previously chosen words. This construction is greedy in the sense that it selects the first vector on a list that satisfies the distance test. Clearly different ways of listing all length n binary vectors will produce different codes. One such list can be obtained by converting the integers from 0 to $2^n - 1$ to binary n vectors.
Write a program for this greedy method.

(1) Try distance 3, 5, and 6. Do you get good codes?

(2) Are the codes linear? If so, why?

You might try listing the binary vectors in a different order.

Chapter 2

Finite Fields

We extend the description from binary codes to codes where the alphabet is a finite field. In this chapter we define finite fields and investigate some of the most important properties. We have chosen to cover in the text only what we think is necessary for coding theory. Some additional material is treated in the problems. The chapter also contains a section on geometries over finite fields; these will be used in code constructions in Chapters 5 and 8.

2.1 Fundamental properties of finite fields

We begin with the *definition of a field*.

Definition 2.1.1. *A* field F *is a nonempty set,* S*, with two binary operations,* $+$ *and* $\cdot$*, and two different elements of* S*,* 0 *and* 1*, such that the following axioms are satisfied:*

1. $\forall x, y \quad x + y = y + x;$
2. $\forall x, y, z \quad (x + y) + z = x + (y + z);$
3. $\forall x \quad x + 0 = x;$
4. $\forall x \; \exists(-x) \quad x + (-x) = 0;$
5. $\forall x, y \quad x \cdot y = y \cdot x;$
6. $\forall x, y, z \quad x \cdot (y \cdot z) = (x \cdot y) \cdot z;$
7. $\forall x \quad x \cdot 1 = x;$
8. $\forall x \neq 0 \; \exists x^{-1} \quad x \cdot x^{-1} = 1;$
9. $\forall x, y, z \quad x \cdot (y + z) = x \cdot y + x \cdot z.$

Classical examples of fields are the rational numbers $\mathbb{Q}$, the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$.

If the number of elements $|S|$ in S is finite, we have a *finite field*. It is an interesting fact that finite fields can be completely determined in the sense that they exist if and only if the number of elements is a power of a prime, and they are essentially unique. In the following we will mostly consider the cases where $|S| = p$, p is a prime, and $|S| = 2^m$.

Theorem 2.1.1. *Let p be a prime and let $S = \{0, 1, \dots, p-1\}$. Let $+$ and $\cdot$ denote addition and multiplication modulo p, respectively. Then $(S, +, \cdot, 0, 1)$ is a finite field with p elements which we denote $\mathbb{F}_p$.*

Proof. It follows from elementary number theory that the axioms 1.–7. and 9. are satisfied. That the nonzero elements of S have a multiplicative inverse (axiom 8.) can be seen in the following way:

Let x be a nonzero element of S and consider the set $\hat{S} = \{1 \cdot x, 2 \cdot x, \dots, (p-1) \cdot x\}$. We will prove that the elements in $\hat{S}$ are distinct and nonzero, so $\hat{S} = S \backslash \{0\}$ and in particular there exists an i such that $i \cdot x = 1$.

It is clear that $0 \notin \hat{S}$, since $0 = i \cdot x, 1 \le i \le p-1$ implies $p|ix$, and since p is a prime one has $p|i$ or $p|x$, a contradiction.

To see that the elements in $\hat{S}$ are distinct suppose that $i \cdot x = j \cdot x$, where $1 \le i, j \le p-1$. Then $p|(ix - jx) \Leftrightarrow p|(i-j)x$ and again since p is a prime we get $p|(i-j)$ or $p|x$. But $x < p$ and $i - j \in \{-(p-2), \dots, (p-2)\}$, so $i = j$. □

Example 2.1.1. The finite field $\mathbb{F}_2$.
If we let $p = 2$, we recover the field $\mathbb{F}_2$.

Example 2.1.2. The finite field $\mathbb{F}_3$.
If we let $p = 3$ we get *the ternary field* with elements $0, 1, 2$.
Here we have

$$0+0=0, \quad 0+1=1+0=1, \quad 0+2=2+0=2, \quad 1+1=2,$$
$$1+2=2+1=0, \quad 2+2=1, \quad 0\cdot 0=0\cdot 1=1\cdot 0=2\cdot 0=0\cdot 2=0,$$
$$1\cdot 2=2\cdot 1=2, \quad 2\cdot 2=1 \quad (\text{so } 2^{-1}=2\ !).$$

We will now prove that multiplication in any finite field F with q elements can essentially be done as addition of integers modulo $q-1$. This results from the fact that F contains an element α, a so-called *primitive element*, such that $F\backslash\{0\} = \{\alpha^i \mid i = 0, 1, \dots, q-2\}$ and $\alpha^{q-1} = 1$. Therefore $\alpha^i \cdot \alpha^j = \alpha^{(i+j) \bmod (q-1)}$.

Definition 2.1.2. *Let F be a finite field with q elements, and let $a \in F\backslash\{0\}$. The* order *of a,* ord (a)*, is the smallest positive integer s such that $a^s = 1$.*

The set $\{a^i \mid i = 1, 2, \dots\}$ is a subset of F and must therefore be finite, so there exists i_1 and i_2 such that $a^{i_1} = a^{i_2}$ and hence $a^{i_1 - i_2} = 1$. This means that there exists an i such that $a^i = 1$ and therefore also a smallest such i, so the order is well defined.

Lemma 2.1.1. *Let F be a finite field with q elements, and let $a, b \in F\backslash\{0\}$; then*

1. ord $(a) = s \Rightarrow a, a^2, \dots, a^s$ *are all different.*
2. $a^j = 1 \Leftrightarrow$ ord $(a) \,|\, j$.

3. $\operatorname{ord}\left(a^j\right) = \frac{\operatorname{ord}(a)}{\gcd(\operatorname{ord}(a), j)}$.
4. $\operatorname{ord}(a) = s$, $\operatorname{ord}(b) = j$, $\gcd(s, j) = 1 \Rightarrow \operatorname{ord}(ab) = sj$.

Proof.

1. If $a^i = a^j$, $0 < i < j \leq s$, then $a^{j-i} = 1$ with $0 < j - i < s$, contradicting the definition of the order.
2. If $\operatorname{ord}(a) = s$ and $j = sh$, then $a^j = a^{sh} = (a^s)^h = 1$. If $a^j = 1$ we let $j = sh + r$ with $0 \leq r < s$ and get $1 = a^j = a^{sh+r} = (a^s)^h a^r = a^r$ and therefore $r = 0$ by the definition of the order, so $s|j$.
3. Let $\operatorname{ord}(a) = s$ and $\operatorname{ord}\left(a^j\right) = l$; then $1 = (a^j)^l = a^{jl}$ so by item 2., we get $s|jl$, and therefore $\frac{s}{\gcd(s,j)} \Big| \frac{jl}{\gcd(s,j)}$, hence $\frac{s}{\gcd(s,j)} \Big| l$.

   On the other hand $(a^j)^{\frac{s}{\gcd(j,s)}} = (a^s)^{\frac{j}{\gcd(j,s)}} = 1$, so again by item 2. we have $l \Big| \frac{s}{\gcd(j,s)}$, and therefore $l = \frac{s}{\gcd(j,s)}$.
4. $(ab)^{sj} = (a^s)^j (b^j)^s = 1 \cdot 1 = 1$, so by item 2. $\operatorname{ord}(ab)\,|sj$, and hence $\operatorname{ord}(ab) = l_1 \cdot l_2$ where $l_1|s$ and $l_2|j$.

   So $1 = (ab)^{l_1 l_2}$ and therefore $1 = [(ab)^{l_1 l_2}]^{\frac{s}{l_1}} = a^{sl_2} b^{sl_2} = b^{sl_2}$, so by item 2., $j|l_2 s$, and since $\gcd(j, s) = 1$, we have $j|l_2$ and hence $j = l_2$.

   In the same way we get $s = l_1$ and the claim is proved. □

The lemma enables us to prove

Theorem 2.1.2. *Let F be a finite field with q elements. Then F has an element of order $q - 1$.*

Proof. Since $|F \backslash \{0\}| = q - 1$, it follows from 1. of Lemma 2.1.1 that the order of any element is at most $q - 1$.

Let α be an element of maximal order in $F \backslash \{0\}$. Let $\operatorname{ord}(\alpha) = r$ and $\operatorname{ord}(\beta) = s$. We will first show that $s|r$.

Suppose not; then there exist a prime p and natural numbers i and j such that $r = p^i \cdot a$, $s = p^j \cdot b$ with $j > i$ and $\gcd(a, p) = \gcd(b, p) = 1$.

We have from Lemma 2.1.1 3. that

$$\operatorname{ord}\left(\alpha^{p^i}\right) = \frac{r}{\gcd\left(r, p^i\right)} = a$$

and

$$\operatorname{ord}\left(\beta^b\right) = \frac{s}{\gcd(s, b)} = p^j,$$

and since $\gcd(a, p) = 1$ we get from 2.1.1 4.

$$\operatorname{ord}\left(\alpha^{p^i} \cdot \beta^b\right) = a \cdot p^j > a \cdot p^i = r$$

contradicting the assumption that r is the maximal order.

So we have $\mathrm{ord}\,(\beta)\,|\mathrm{ord}\,(\alpha)$.

This implies that every element of $F\backslash\{0\}$ is a zero of the polynomial $z^{\mathrm{ord}(\alpha)}-1$ and since a polynomial of degree n can have at most n zeroes (see Theorem 2.2.2), we conclude that $\mathrm{ord}\,(\alpha) \geq q-1$, and hence that $\mathrm{ord}\,(\alpha) = q-1$, and the theorem is proved. □

Corollary 2.1.1. *The order of any nonzero element in a finite field with q elements divides $q-1$.*

Corollary 2.1.2. *Any element γ of a finite field with q elements satisfies $\gamma^q - \gamma = 0$.*

The theorem does not give a method to find a primitive element; usually one has to use trial and error.

Example 2.1.3. 3 is a primitive element of $\mathbb{F}_{17}$.
Since the possible orders of a nonzero element of $\mathbb{F}_{17}$ are $1, 2, 4, 8$, and 16, but $3^2 = 9$, $3^4 = 13$, and $3^8 = 16$, we see that 3 must have order 16.

Example 2.1.4. Hamming codes over $\mathbb{F}_p$.
Let p be a prime and m a positive integer. Let H be a matrix with m rows and with the maximum number of columns such that any two are linearly independent. It is easy to see that this number n equals $\frac{p^m-1}{p-1}$. The code that has H as a parity check matrix is an

$$\left(\frac{p^m-1}{p-1}, \frac{p^m-1}{p-1} - m, 3\right)$$

code, the *p-ary Hamming code*.
Note that $1 + (p-1)n = p^m$, so the Hamming bound is satisfied with equality. For this reason it is called a *perfect code*.

2.2 Polynomials over finite fields

In the following we will consider some of the properties of polynomials with coefficients in a field.

Recall that if F is a field, then $F[x]$ denotes the set of polynomials with coefficients from F, i.e., expressions of the form

$$a_n x^n + \cdots + a_1 x + a_0,$$

where $a_i \in F$. We have the notion of the *degree* of a polynomial (denoted deg) and can do addition and multiplication of polynomials.

Theorem 2.2.1. *Let $a(x), b(x) \in F[x], b(x) \neq 0$; then there exist unique polynomials $q(x)$ and $r(x)$ with* $\deg(r(x)) < \deg(b(x))$ *such that*

$$a(x) = q(x)b(x) + r(x).$$

Proof. To prove the uniqueness, suppose $a(x) = q_1(x)b(x) + r_1(x)$ and $a(x) = q_2(x)b(x) + r_2(x)$, where $\deg(r_1(x)) < \deg(b(x))$ and $\deg(r_2(x)) < \deg(b(x))$. We then get $r_2(x) - r_1(x) = b(x)(q_1(x) - q_2(x))$, but since the degree of the polynomial on the left side is smaller than the degree of $b(x)$, this is only possible if $(q_1(x) - q_2(x)) = 0$ and $r_2(x) - r_1(x) = 0$.

To prove the existence, we first note that if $\deg(b(x)) > \deg(a(x))$ we have $a(x) = 0 \cdot b(x) + a(x)$ so the claim is obvious here. If $\deg(b(x)) \leq \deg(a(x))$ let $a(x) = a_n x^n + \cdots + a_1 x + a_0$ and $b(x) = b_m x^m + \cdots + b_1 x + b_0$ and look at the polynomial $b_m^{-1} a_n x^{n-m} b(x) - a(x)$. This has degree $< n$, so we can use induction on the degree of $a(x)$ to get the result. □

Theorem 2.2.2. *A polynomial of degree n has at most n zeroes.*

Proof. Let a be a zero of the polynomial $f(x) \in F[x]$. By the above, we have that $f(x) = q(x)(x-a) + r(x)$ with $\deg(r(x)) < 1$, so $r(x)$ must be a constant. Since a is a zero of $f(x)$, we have $0 = f(a) = q(x) \cdot 0 + r(x)$, and therefore $r(x) = 0$. Hence, $f(x) = q(x)(x-a)$ and $q(x)$ has degree $n-1$. If b, $b \neq a$ also is a zero of $f(x)$, it must be a zero of $q(x)$ so we can repeat the argument and eventually get the result. □

We note that from the above we also get

Corollary 2.2.1. *If $f(x) \in F[x]$, then $f(a) = 0 \Leftrightarrow (x-a) | f(x)$.*

A polynomial $f(x) \in F[x]$ is called *irreducible* if $f(x) = a(x)b(x)$ implies that $\deg(a(x)) = 0$ or $\deg(b(x)) = 0$ (i.e., either $a(x)$, or $b(x)$ is a constant). Irreducible polynomials play the same role in $F[x]$ as prime numbers for the integers, since it can be shown that any polynomial can be written (uniquely) as a product of irreducible polynomials. From this it follows that if $f(x)$ is irreducible and $f(x)|a(x)b(x)$, then $f(x)|a(x)$ or $f(x)|b(x)$. Actually the proofs are fairly easy modifications of the corresponding proofs for integers.

Example 2.2.1. Quadratic polynomials over $\mathbb{F}_p$, $p > 2$.
Let $a \in \mathbb{F}_p \setminus \{0\}$ and let α be a primitive element of $\mathbb{F}_p$. Then either $a = \alpha^{2i}$, or $a = \alpha^{2i+1}$, where $0 \leq i < \frac{p-1}{2}$. In the first case we get

$$x^2 - a = (x - \alpha^i)(x + \alpha^i),$$

while in the second case $x^2 - a$ is irreducible.

For a general quadratic polynomial $f(x) = Ax^2 + Bx + C$ we get

$$f(x) = Ax^2 + Bx + C = A(x - \gamma_1)(x - \gamma_2),$$

where

$$\gamma_1 = (2A)^{-1}(-B + \sqrt{B^2 - 4AC}), \quad \gamma_2 = (2A)^{-1}(-B - \sqrt{B^2 - 4AC})$$

if $B^2 - 4AC$ is a square in $\mathbb{F}_p$, and if not the polynomial $f(x)$ is irreducible.

Example 2.2.2. For the polynomial $3x^2 + 5x + 2 \in \mathbb{F}_{11}$ we get $\gamma_1 = 6^{-1}(-5 + \sqrt{25 - 24}) = 6^{-1}(-4) = 2 \times 7 = 3$ and $\gamma_2 = 10$, so $3x^2 + 5x + 2 = 3(x - 3) \cdot (x - 10)$. The polynomial $3x^2 + 5x + 1 \in \mathbb{F}_{11}$ is irreducible since 2 is not a square in $\mathbb{F}_{11}$.

2.2.1 Reed–Solomon codes over $\mathbb{F}_p$

In Chapter 5 we will treat the Reed–Solomon codes in general; here we just use the theory of polynomials to treat the codes over $\mathbb{F}_p$. For p a prime, let $x_1, x_2, \ldots, x_n$ be different elements of $\mathbb{F}_p$ (so $n \le p$). Let $0 \le k \le n$ and $T = \{f(x) \in \mathbb{F}_p[x] \mid \deg(f(x)) < k\}$. Then the *Reed–Solomon code* is

$$\{(f(x_1), f(x_2), \ldots, f(x_n)) \mid f(x) \in T\}$$

As we shall see later, this is an $(n, k, n - k + 1)$ code.

Example 2.2.3. Reed–Solomon codes over $\mathbb{F}_{11}$.
Since 2 is a primitive element of $\mathbb{F}_{11}$, we can take $x_i = 2^{i-1} \bmod 11$, $i = 1, 2, \ldots, 10$ with $k = 4$ and $f(x) = f_3x^3 + \cdots + f_1x + f_0$; we get as the corresponding codeword

$$(f(1), f(2), f(4), f(8), f(5), f(10), f(9), f(7), f(3), f(6)).$$

So,

the polynomial 1	gives the codeword	$(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$,
the polynomial x	gives the codeword	$(1, 2, 4, 8, 5, 10, 9, 7, 3, 6)$,
the polynomial x^2	gives the codeword	$(1, 4, 5, 9, 3, 1, 4, 5, 9, 3)$,
the polynomial x^3	gives the codeword	$(1, 8, 9, 6, 4, 10, 3, 2, 5, 7)$,

and these four codewords can be used as the rows of a generator matrix of the code, which is a $(10, 4, 7)$ code over $\mathbb{F}_{11}$.

2.3 The finite field $\mathbb{F}_{2^m}$

In this section we will *construct the field* $\mathbb{F}_{2^m}$.

As elements of $\mathbb{F}_{2^m}$ we take the 2^m m-tuples of elements from $\mathbb{F}_2$ and we define addition coordinatewise. This implies that $a + a = 0$. It is easy to see that the first four axioms for a field are satisfied.

Multiplication is somewhat more complicated. Let $f(x) \in \mathbb{F}_2[x]$ be an irreducible polynomial of degree m. Let $a = (a_{m-1}, \dots, a_1, a_0)$ and $b = (b_{m-1}, \dots, b_1, b_0)$ and therefore $a(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$ and $b(x) = b_{m-1}x^{m-1} + \cdots + b_1 x + b_0$. We define the multiplication as $a(x)b(x)$ modulo $f(x)$, i.e., if $a(x)b(x) = q(x)f(x) + r(x)$, where $\deg(r(x)) < m$, we set $r = (r_0, r_1, \dots, r_{m-1})$. If we let $1 = (1, 0, \dots, 0)$ we see that $1 \cdot a = a$.

With the addition and multiplication we have just defined we have constructed a field. This is fairly easy to see, again the most difficult part being to prove the existence of a multiplicative inverse for the nonzero elements.

To this end we copy the idea from $\mathbb{F}_p$. Let $a \in \mathbb{F}_{2^m}\backslash\{0\}$. The set $A = \{a \cdot h \mid h \in \mathbb{F}_{2^m}\backslash\{0\}\}$ does not contain 0 since this would, for the corresponding polynomials, imply that $f(x)|a(x)$ or $f(x)|h(x)$. Moreover, the elements of A are different, since if $a \cdot h_1 = a \cdot h_2$ we get for the corresponding polynomials that $f(x)|a(x)(h_1(x) - h_2(x))$, and since $f(x)$ is irreducible, this implies $f(x)|a(x)$ or $f(x)|h_1(x) - h_2(x)$. But this is only possible if $h_1(x) = h_2(x)$, since $f(x)$ has degree m but both $a(x)$ and $h_1(x) - h_2(x)$ have degree $< m$. This gives that $A = \mathbb{F}_{2^m}\backslash\{0\}$ and in particular that $1 \in A$.

It can be proven that for any positive integer m there exists an irreducible polynomial of degree m with coefficients in $\mathbb{F}_2$ and therefore the above construction gives for any positive integer m a finite field with $q = 2^m$ elements; we denote this field by $\mathbb{F}_q$. It can also be proven that $\mathbb{F}_q$ is essentially unique. We also note that if $f(x) \in \mathbb{F}_{2^m}[x]$ is irreducible with degree m_1, then a construction similar to the one above can be used to produce the finite field $\mathbb{F}_{2^{mm_1}}$.

Example 2.3.1. Irreducible polynomials from $\mathbb{F}_2[x]$ of degree at most 4.
There are two polynomials of degree 1: x, $x + 1$. Irreducible polynomials of higher degree must have constant term 1, since otherwise x would be a factor and an odd number of terms, since otherwise else 1 would be a zero and therefore $x - 1$ would be a factor, so for degrees 2 and 3 we get: $x^2 + x + 1$, $x^3 + x + 1$ and $x^3 + x^2 + 1$. For degree 4 we get: $x^4 + x + 1$, $x^4 + x^3 + 1$, $x^4 + x^3 + x^2 + x + 1$. The polynomial $x^4 + x^2 + 1$, which also has an odd number of terms and constant term 1, is not irreducible since $x^4 + x^2 + 1 = (x^2 + x + 1)^2$.

Example 2.3.2. The finite field $\mathbb{F}_4$.
Let the elements be 00, 10, 01 and 11; then we get the following table for addition:

$+$	00	10	01	11
00	00	10	01	11
10	10	00	11	01
01	01	11	00	10
11	11	01	10	00

Using the irreducible polynomial $x^2 + x + 1$ we get the following table for multiplication:

$\cdot$	00	10	01	11
00	00	00	00	00
10	00	10	01	11
01	00	01	11	10
11	00	11	10	01

From the multiplication table it is seen that the element 01 corresponding to the polynomial x is primitive, i.e., has order 3.

We have seen that multiplication in a finite field is easy once we have a primitive element and from the construction above addition is easy when we have the elements as binary m-tuples. Therefore, we should have a table listing all the binary m-tuples and the corresponding powers of a primitive element in order to do calculations in $\mathbb{F}_{2^m}$. We illustrate this in

Example 2.3.3. The finite field $\mathbb{F}_{16}$.
The polynomial $x^4 + x + 1 \in \mathbb{F}_2[x]$ is irreducible and can therefore be used to construct $\mathbb{F}_{16}$. The elements are the binary 4-tuples: $(0, 0, 0, 0), \dots, (1, 1, 1, 1)$. These can also be considered as all binary polynomials of degree at most 3. If we calculate the powers of $(0, 0, 1, 0)$ which corresponds to the polynomial x, we get:

$$\begin{aligned}
&x^0 = 1, \quad x^1 = x, \quad x^2 = x^2, \quad x^3 = x^3, \quad x^4 = x + 1, \quad x^5 = x^2 + x, \\
&x^6 = x^3 + x^2, \quad x^7 = x^3 + x + 1, \quad x^8 = x^2 + 1, \quad x^9 = x^3 + x, \\
&x^{10} = x^2 + x + 1, \quad x^{11} = x^3 + x^2 + x, \quad x^{12} = x^3 + x^2 + x + 1, \\
&x^{13} = x^3 + x^2 + 1, \quad x^{14} = x^3 + 1, \quad x^{15} = 1.
\end{aligned}$$

This means we can use $(0, 0, 1, 0)$ as a primitive element which we call α, and list all the binary 4-tuples and the corresponding powers of α.

binary 4-tuple	power of α	polynomial
0000		0
0001	α^0	1
0010	α	x
0100	α^2	x^2
1000	α^3	x^3
0011	α^4	$x + 1$
0110	α^5	$x^2 + x$
1100	α^6	$x^3 + x^2$
1011	α^7	$x^3 + x + 1$
0101	α^8	$x^2 + 1$
1010	α^9	$x^3 + x$
0111	α^{10}	$x^2 + x + 1$
1110	α^{11}	$x^3 + x^2 + x$
1111	α^{12}	$x^3 + x^2 + x + 1$
1101	α^{13}	$x^3 + x^2 + 1$
1001	α^{14}	$x^3 + 1$

If $f(x) \in \mathbb{F}_2[x]$ is irreducible and has degree m, then it can be used to construct $\mathbb{F}_{2^m}$ as we have seen. If we do this, *then there exists an element* $\beta \in \mathbb{F}_{2^m}$ *such that* $f(\beta) = 0$, namely the element corresponding to the polynomial x. This follows directly from the way $\mathbb{F}_{2^m}$ is constructed.

Actually, as we shall see, the polynomial $f(x)$ can be written as a product of polynomials from $\mathbb{F}_{2^m}[x]$ of degree 1.

2.3.1 Quadratic polynomials over $\mathbb{F}_{2^m}$

We first note that if $\alpha \in \mathbb{F}_{2^m}$ is a primitive element, then $\alpha^{2i} = (\alpha^i)^2$ and $\alpha^{2i+1} = \alpha^{2i+2^m} = (\alpha^{i+2^{m-1}})^2$, so every element in $\mathbb{F}_{2^m}$ is a square.

Let now $a(x) = Ax^2 + Bx + C$, where $A, B, C \in \mathbb{F}_{2^m}$ and $A \neq 0$.

If $C = 0$, we have $a(x) = x(Ax + B)$ with zeroes $x = 0$ and $x = A^{-1}B$.

If $C \neq 0$ and $B = 0$, we get $a(x) = Ax^2 + C = A(x^2 + A^{-1}C) = A(x + \sqrt{A^{-1}C})^2$.

If $C \neq 0$ and $B \neq 0$, we get $a(x) = A(x^2 + A^{-1}Bx + A^{-1}C)$. With the substitution $y = B^{-1}Ax$ this becomes $a(y) = A(A^{-2}B^2y^2 + A^{-2}B^2y + A^{-1}C) = A^{-1}B^2(y^2 + y + AB^{-2}C)$, so we only have to consider polynomials of the form $b(y) = y^2 + y + u$ with $u \in \mathbb{F}_{2^m} \setminus \{0\}$. In order to analyse this we introduce

Definition 2.3.1. *The map* $\mathrm{Tr} : \mathbb{F}_{2^m} \to \mathbb{F}_2$ *defined by*

$$\mathrm{Tr}(x) = x + x^2 + x^4 + \cdots + x^{2^{m-1}}$$

is called the (*absolute*) *trace.*

For properties of the trace map, see Problem 2.6.11.
We can now prove

Theorem 2.3.1. *The equation*

$$y^2 + y + u = 0$$

has a solution $y \in \mathbb{F}_{2^m}$ *if and only if* $\mathrm{Tr}(u) = 0$

Proof. Suppose $y \in \mathbb{F}_{2^m}$ satisfy $y^2+y = u$. This implies $\mathrm{Tr}(u) = \mathrm{Tr}(y^2+y) = \mathrm{Tr}(y^2)+\mathrm{Tr}(y) = y^2+y^4+\cdots+y^{2^m}+y+y^2+\cdots+y^{2^{m-1}} = y^{2^m}+y = 0$, since $y \in \mathbb{F}_{2^m}$.

On the other hand, if $y^2 + y + u = 0$ has no solution in $\mathbb{F}_{2^m}$, then the polynomial $y^2 + y + u$ is irreducible over $\mathbb{F}_{2^m}$ and can be used to construct the field $\mathbb{F}_{2^{2m}}$ and in this field there exists an element β such that $\beta^2 + \beta + u = 0$. Hence $\mathrm{Tr}(u) = \mathrm{Tr}(\beta^2+\beta) = \mathrm{Tr}(\beta^2)+\mathrm{Tr}(\beta) = \beta^{2^m}+\beta \neq 0$ since $\beta \notin \mathbb{F}_{2^m}$. $\square$

If $\mathrm{Tr}(u) = 0$ we can find the solutions in the following way: Let α be a primitive element in $\mathbb{F}_{2^m}$. Now solve the binary system of m equations (i.e., $y_i \in \mathbb{F}_2$)

$$\sum_{i=0}^{m-1} (\alpha^{2i} + \alpha^i) y_i = u.$$

It is easy to see that if there is a solution, then the equation $y^2 + y + u = 0$ has the solutions $y = \sum_{i=0}^{m-1} y_i \alpha^i$ and $y + 1$.

2.4 Minimal polynomials and factorization of $x^n - 1$

Our main objective in this section is to present an algorithm for the factorization of $x^n - 1$ into irreducible polynomials from $\mathbb{F}_2[x]$. We do this at the end of the section. Along the way we shall define the so-called *minimal polynomials* and prove some of their properties.

Lemma 2.4.1. *If* $a, b \in \mathbb{F}_{2^m}$, *then* $(a + b)^2 = a^2 + b^2$.

Theorem 2.4.1. *Let* $f(x) \in \mathbb{F}_{2^m}[x]$. *Then* $f(x) \in \mathbb{F}_2[x] \Leftrightarrow f(x^2) = f(x)^2$.

Proof. Let $f(x) = f_k x^k + \cdots + f_1 x + f_0$. Then $f(x)^2 = (f_k x^k + \cdots + f_1 x + f_0)^2 = f_k^2 x^{2k} + \cdots + f_1^2 x^2 + f_0^2$ and $f(x^2) = f_k x^{2k} + \cdots + f_1 x^2 + f_0$. So $f(x)^2 = f(x^2) \Leftrightarrow f_i^2 = f_i \Leftrightarrow f_i \in \mathbb{F}_2$. □

Corollary 2.4.1. *Let $f(x) \in \mathbb{F}_2[x]$. If $\gamma \in \mathbb{F}_{2^m}$ is a zero of $f(x)$, then so is γ^2.*

Theorem 2.4.2. $(x^m - 1)|(x^n - 1) \Leftrightarrow m|n$.

Proof. Follows from the identity:

$$x^n - 1 = (x^m - 1)(x^{n-m} + x^{n-2m} + \cdots + x^{n-km}) + x^{n-km} - 1. \qquad \square$$

Theorem 2.4.3. *$\mathbb{F}_{2^m}$ is a subfield of $\mathbb{F}_{2^n}$ $\Leftrightarrow m|n$.*

Proof. If $\mathbb{F}_{2^m}$ is a subfield of $\mathbb{F}_{2^n}$, then $\mathbb{F}_{2^n}$ contains an element of order $2^m - 1$, and therefore $(2^m - 1)|(2^n - 1)$ and so $m|n$. If $m|n$, we have $(2^m - 1)|(2^n - 1)$ and therefore $(x^{2^m-1} - 1)|(x^{2^n-1} - 1)$, so $(x^{2^m} - x)|(x^{2^n} - x)$ and hence $\mathbb{F}_{2^m} = \{x \mid x^{2^m} = x\} \subset \mathbb{F}_{2^n} = \{x \mid x^{2^n} = x\}$. □

Example 2.4.1. $(x^{n_1 n_2} - 1)$, with n_1, n_2 odd. If $\beta \in \mathbb{F}_{2^d}$ has order n_1, then

$$x^{n_1} - 1 = (x - 1)(x - \beta) \cdots (x - \beta)^{n_1 - 1}$$

So we get

$$x^{n_1 n_2} - 1 = (x^{n_2} - 1)(x^{n_2} - \beta)(x^{n_2} - \beta^2) \ldots (x^{n_2} - \beta^{n_1})$$

Definition 2.4.1. *Let γ be an element of $\mathbb{F}_{2^m}$. The* minimal polynomial *of γ, $m_\gamma(x)$, is the monic polynomial in $\mathbb{F}_2[x]$ of lowest degree that has γ as a zero.*

Since γ is a zero of $x^{2^m} - x$, there indeed exists a binary polynomial with γ as a zero. The minimal polynomial is unique, since if there were two of the same degree, their difference would have lower degree, but still have γ as a zero.

Theorem 2.4.4. *Let γ be an element of $\mathbb{F}_{2^m}$ and let $m_\gamma(x) \in \mathbb{F}_2[x]$ be the minimal polynomial of γ. Then:*

1. *$m_\gamma(x)$ is irreducible.*
2. *If $f(x) \in \mathbb{F}_2[x]$ satisfies $f(\gamma) = 0$, then $m_\gamma(x)|f(x)$.*
3. *$x^{2^m} - x$ is the product of the distinct minimal polynomials of the elements of $\mathbb{F}_{2^m}$.*
4. deg$(m_\gamma(x)) \leq m$, *with equality if γ is a primitive element (in this case the minimal polynomial is called* primitive).

Proof.

1. If $m_\gamma(x) = a(x)b(x)$ we would have $a(\gamma) = 0$ or $b(\gamma) = 0$, contradicting the minimality of the degree of $m_\gamma(x)$.
2. Let $f(x) = m_\gamma(x)q(x) + r(x)$ with $\deg(r(x)) < \deg(m_\gamma(x))$. This gives $r(\gamma) = 0$, and therefore $r(x) = 0$.
3. Any element of $\mathbb{F}_{2^m}$ is a zero of $x^{2^m} - x$, so by item 2. we get the result.
4. Since $\mathbb{F}_{2^m}$ can be seen as a vector space over $\mathbb{F}_2$ of dimension m, the $m+1$ elements $1, \gamma, \dots, \gamma^m$ are linearly dependent, so there exists $(a_0, a_1, \dots, a_m) \neq (0, 0, \dots, 0)$ $a_i \in \mathbb{F}_2$ such that $a_m\gamma^m + \cdots + a_1\gamma + a_0 = 0$. If γ is a primitive element, $1, \gamma, \dots, \gamma^{m-1}$ must be linearly independent, since otherwise, the powers of γ would give fewer than $2^m - 1$ distinct elements. □

Theorem 2.4.5. $x^{2^m} - x$ = *the product of all binary irreducible polynomials whose degrees divide* m.

Proof. Let $f(x)$ be an irreducible polynomial of degree d, where $d|m$. We want to prove that $f(x)|x^{2^m} - x$. This is trivial if $f(x) = x$, so we assume $f(x) \neq x$. Since $f(x)$ is irreducible, it can be used to construct $\mathbb{F}_{2^d}$ and $f(x)$ is the minimal polynomial of some element in $\mathbb{F}_{2^d}$; so by item 2. above, we have $f(x)|x^{2^d-1} - 1$. Since $d|m$, this implies that $2^d - 1|2^m - 1$ and therefore $x^{2^d-1} - 1|x^{2^m-1} - 1$. Conversely if $f(x)$ is irreducible, divides $x^{2^m} - x$ and has degree d, we shall prove that $d|m$. Again we can assume $f(x) \neq x$ and hence $f(x)|x^{2^m-1} - 1$. We can use $f(x)$ to construct $\mathbb{F}_{2^d}$. Let $\beta \in \mathbb{F}_{2^d}$ be a zero of $f(x)$ and let α be a primitive element of $\mathbb{F}_{2^d}$, say

$$\alpha = a_{d-1}\beta^{d-1} + \cdots + a_1\beta + a_0. \tag{2.1}$$

Since $f(\beta) = 0$, we have $\beta^{2^m} = \beta$ and by Equation (2.1) and Lemma 2.4.1 we get that $\alpha^{2^m} = \alpha$, and hence $\alpha^{2^m-1} = 1$. Then the order $2^d - 1$ of α must divide $2^m - 1$, so by Theorem 2.4.2 $d|m$ and we are finished. □

Definition 2.4.2. *Let n be an* odd *number and let j be an integer* $0 \le j < n$. *The* cyclotomic coset *containing j is defined as*

$$\{j, 2j \mod n, \dots, 2^i j \mod n, \dots, 2^{s-1} j \mod n\},$$

where s is the smallest positive integer such that $2^s j \bmod n = j$.

If we look at the numbers $2^i j \mod n$, $i = 0, 1, \dots,$ they cannot all be different, so the s in the definition above does indeed exist.

If $n = 15$ we get the following cyclotomic cosets:

$$\{0\}$$
$$\{1, 2, 4, 8\}$$
$$\{3, 6, 12, 9\}$$
$$\{5, 10\}$$
$$\{7, 14, 13, 11\}$$

We will use the following notation: if j is the smallest number in a cyclotomic coset, then that coset is called C_j. The subscripts j are called *coset representatives*. So the above cosets are C_0, C_1, C_3, C_5, and C_7. Of course, we always have $C_0 = \{0\}$.

It can be seen from the definition that if $n = 2^m - 1$ and we represent a number i as a binary m-vector, then the cyclotomic coset containing i consists of that m-vector and all its cyclic shifts.

The algorithm given below is based on

Theorem 2.4.6. *Suppose n is an odd number and the cyclotomic coset C_1 has m elements. Let α be a primitive element of $\mathbb{F}_{2^m}$ and $\beta = \alpha^{\frac{2^m-1}{n}}$. Then $m_{\beta^j} = \prod_{i \in C_j}(x - \beta^i)$.*

Proof. Let $f_j = \prod_{i \in C_j}(x - \beta^i)$; then $f_j(x^2) = (f_j(x))^2$ by Definition 2.4.2 and the fact that β is of order n. It then follows from Theorem 2.4.1 that $f_j(x) \in \mathbb{F}_2[x]$.

We also have that $f_j(\beta^j) = 0$, so $m_{\beta^j} \mid f_j(x)$ by Theorem 2.4.4. It follows from Corollary 2.4.1 that the elements $(\beta)^i$ with $i \in C_j$ also are zeroes of $m_{\beta^j}(x)$ and therefore $m_{\beta^j} = f_j(x)$. □

Corollary 2.4.2. *With the above* notation *we have*

$$x^n - 1 = \prod_j f_j(x)$$

We can then give the algorithm. Since this is the first time we present an algorithm, we emphasize that we use the concept of an algorithm in the common informal sense of a computational procedure that can be effectively executed by a person or a suitably programmed computer. There is abundant evidence that computable functions do not depend on the choice of a specific finite set of basic instructions or on the programming language. An algorithm takes a finite input, and terminates after a finite number of steps producing a finite output. Here, algorithms are presented in standard algebraic notation, and it should be clear from the context that the descriptions can be converted to programs in a specific language.

Algorithm 2.4.1. *Factorization of $x^n - 1$.*
Input: *An odd number n*

1. *Find the cyclotomic cosets modulo n.*
2. *Find the number m of elements in C_1.*
3. *Construct the finite field $\mathbb{F}_{2^m}$, select a primitive element α, and put $\beta = \alpha^{\frac{2^m-1}{n}}$.*
4. *Calculate*
$$f_j(x) = \prod_{i \in C_j} (x - \beta^i), \quad j = 0, 1, \dots.$$

Output: *The factors of $x^n - 1$, $f_0(x)$, $f_1(x), \dots$*

One can argue that this is not really an algorithm because in step 3 one needs an irreducible polynomial in $\mathbb{F}_2[x]$ of degree m.

The table in Appendix C not only gives such polynomials up to degree 16, these are also chosen so that they are minimal polynomials of a primitive element.

For $m \leq 8$ we have moreover given the minimal polynomials for α^j, where j is a coset representative.

Example 2.4.2. Factorization of $x^{51} - 1$.
The cyclotomic cosets modulo 51 are

$$\begin{aligned}
C_0 &= \{0\}, \\
C_1 &= \{1, 2, 4, 8, 16, 32, 13, 26\}, \\
C_3 &= \{3, 6, 12, 24, 48, 45, 39, 27\}, \\
C_5 &= \{5, 10, 20, 40, 29, 7, 14, 28\}, \\
C_9 &= \{9, 18, 36, 21, 42, 33, 15, 30\}, \\
C_{11} &= \{11, 22, 44, 37, 23, 46, 41, 31\}, \\
C_{17} &= \{17, 34\}, \\
C_{19} &= \{19, 38, 25, 50, 49, 47, 43, 35\}.
\end{aligned}$$

We see that $|C_1| = 8$. Since $\frac{2^8-1}{51} = 5$, we have that $\beta = \alpha^5$, where α is a primitive element of $\mathbb{F}_{2^8}$.
Using the table from Appendix C (the section with $m = 8$) we get the following list of minimal polynomials:

$$\begin{aligned}
m_1(x) &= 1 + x, \\
m_\beta(x) &= m_{\alpha^5}(x) = x^8 + x^7 + x^6 + x^5 + x^4 + x + 1, \\
m_{\beta^3}(x) &= m_{\alpha^{15}}(x) = x^8 + x^7 + x^6 + x^4 + x^2 + x + 1, \\
m_{\beta^5}(x) &= m_{\alpha^{25}}(x) = x^8 + x^4 + x^3 + x + 1,
\end{aligned}$$

$$\begin{aligned}
m_{\beta^9}(x) &= m_{\alpha^{45}}(x) = x^8 + x^5 + x^4 + x^3 + 1,\\
m_{\beta^{11}}(x) &= m_{\alpha^{55}}(x) = x^8 + x^7 + x^5 + x^4 + 1,\\
m_{\beta^{17}}(x) &= m_{\alpha^{85}}(x) = x^2 + x + 1,\\
m_{\beta^{19}}(x) &= m_{\alpha^{95}}(x) = x^8 + x^7 + x^4 + x^3 + x^2 + x + 1.
\end{aligned}$$

2.5 Geometries over finite fields

In this section we will introduce the affine and projective planes over finite fields. As we shall see later, these can be used in various code constructions.

2.5.1 Affine planes

Let $\mathbb{F}_q$ be a finite field. The *affine plane over* $\mathbb{F}_q$ consists of *points* (x, y) with $x, y \in \mathbb{F}_q$ and *lines* which are sets of points $\{(x, y) \mid y = ax + b\}$ or $\{(x, y) \mid x = a\}$ with $a, b \in \mathbb{F}_q$.

It is easy to see that

1. the number of points is q^2;
2. the number of lines is $q^2 + q$;
3. the number of points on a line is q;
4. the number of lines through a point is $q + 1$.

The affine plane over $\mathbb{F}_q$ is denoted A(2,q).

If (x_1, y_1) and (x_2, y_2) are two different points, then there is a unique line containing both, namely

$$\begin{aligned}
&\{(x, y) \mid x = x_1\} \quad \text{if} \quad x_1 = x_2, \quad \text{or}\\
&\left\{(x, y) \mid y = \frac{y_2 - y_1}{x_2 - x_1}x + \frac{y_1x_2 - y_2x_1}{x_2 - x_1}\right\} \quad \text{if } x_1 \neq x_2.
\end{aligned}$$

Two different lines can either have one common point or none, i.e. they are parallel.

Example 2.5.1. A(2,3)
The affine plane over $\mathbb{F}_3$ has nine points $(0, 0)$, $(1, 0)$, $(2, 0)$, $(0, 1)$, $(1, 1)$, $(2, 1)$, $(0, 2)$, $(1, 2)$, $(2, 2)$ and twelve lines with equations

$$\begin{gathered}
x = 0, \quad x = 1, \quad x = 2, \quad y = 0, \quad y = 1, \quad y = 2, \quad y = x,\\
y = x + 1, \quad y = x + 2, \quad y = 2x, \quad y = 2x + 1, \quad y = 2x + 2.
\end{gathered}$$

The lines are divided into four classes with three parallel lines in each.

2.5.2 Projective planes

Let $\mathbb{F}_q$ be a finite field. In ${\mathbb{F}_q}^3 \setminus (0,0,0)$ let $\sim$ be the relation given by $(x_1, x_2, x_3) \sim (y_1, y_2, y_3)$ if there exists an element $t \in \mathbb{F}_q \setminus 0$ such that $(x_1, x_2, x_3) = t(y_1, y_2, y_3)$. It is easy to see that $\sim$ is an equivalence relation.

Let $\mathscr{P}$ be the set of equivalence classes. For $(x_1, x_2, x_3) \in {\mathbb{F}_q}^3 \setminus (0,0,0)$ we denote the class containing (x_1, x_2, x_3) by $(x_1 : x_2 : x_3)$.

The *projective plane* has $\mathscr{P}$ as its set of *points* and the *lines* are subsets of $\mathscr{P}$ of the form

$$\{(x_1 : x_2 : x_3) \,|\, ax_1 + bx_2 + cx_3 = 0\}, \quad \text{where} \quad (a,b,c) \neq (0,0,0).$$

We note that this makes sense since any element in the class $(x_1 : x_2 : x_3)$ has the form $t(x_1, x_2, x_3)$ with $t \in \mathbb{F}_q \setminus 0$ and therefore also satisfies the equation. We also note that if $(a_1, b_1, c_1) \sim (a_2, b_2, c_2)$ the corresponding sets are equal. It can be seen that

1. the number of points is $\frac{q^3-1}{q-1} = q^2 + q + 1$;
2. the number of lines is $\frac{q^3-1}{q-1} = q^2 + q + 1$;
3. the number of points on a line is $q + 1$;
4. the number lines through a point is $q + 1$.

The projective plane over $\mathbb{F}_q$ is denoted PG(2,q).

It also holds that two different points determine a unique line, and that two different lines have a unique common point.

Example 2.5.2. PG(2,2).
The points are $P_0 = (0 : 0 : 1)$, $P_1 = (0 : 1 : 0)$, $P_2 = ((1 : 0 : 0)$, $P_3 = (0 : 1 : 1)$, $P_4 = (1 : 1 : 0)$, $P_5 = (1 : 1 : 1) : 1)$, $P_6 = (1 : 0 : 1)$,
and the lines

$$\begin{aligned}
l_0 : x_1 &= 0 \text{ containing } P_0, P_1, P_3,\\
l_1 : x_3 &= 0 \text{ containing } P_1, P_2, P_4,\\
l_2 : x_2 + x_3 &= 0 \text{ containing } P_2, P_3, P_5,\\
l_3 : x_1 + x_2 + x_3 &= 0 \text{ containing } P_3, P_4, P_6,\\
l_4 : x_1 + x_2 &= 0 \text{ containing } P_4, P_5, P_0,\\
l_5 : x_1 + x_3 &= 0 \text{ containing } P_5, P_6, P_1,\\
l_6 : x_2 &= 0 \text{ containing } P_6, P_0, P_2.
\end{aligned}$$

A useful way to describe the plane is to use the *incidence matrix*. This is in the present case a 7×7 matrix $A = \{a_{ij}\}$, where

$$a_{ij} = \begin{cases} 1, & \text{if} \quad P_i \in l_j, \\ 0, & \text{otherwise.} \end{cases}$$

With the numbering used above we get

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

We note that this matrix is cyclic in the sense that the rows are cyclic shifts of the first row. This is no coincidence, as we shall see later.

2.6 Problems

Problem 2.6.1. Consider $\mathbb{F}_{17}$.

(1) What is the sum of all the elements?

(2) What is the product of the nonzero elements?

(3) What is the order of 2?

(4) What are the possible orders of the elements?

(5) Determine for all the possible orders an element of that order.

(6) How many primitive elements are there?

(7) Try to solve the equation $x^2 + x + 1 = 0$.

(8) Try to solve the equation $x^2 + x - 6 = 0$.

Problem 2.6.2. Let F be a field.

(1) Show that $a \cdot b = 0 \Rightarrow a = 0$ or $b = 0$.

(2) Show that $\{0, 1, 2, 3\}$ with addition and multiplication modulo 4 is *not* a field.

Problem 2.6.3. Let $a \in \mathbb{F}_q$. Determine

$$\sum_{j=0}^{q-2} (a^i)^j$$

Problem 2.6.4. Determine all binary irreducible polynomials of degree 3.

Problem 2.6.5. Construct $\mathbb{F}_8$ using $f(x) = x^3 + x + 1$, that is, explain what the elements are and how to add and multiply them.
Construct a multiplication table for the elements of $\mathbb{F}_8 \setminus \{0, 1\}$.

Problem 2.6.6. Which of the following polynomials can be used to construct $\mathbb{F}_{16}$?

$$x^4 + x^2 + x, \quad x^4 + x^3 + x^2 + 1, \quad x^4 + x + 1, \quad x^4 + x^2 + 1.$$

Problem 2.6.7. We consider $\mathbb{F}_{16}$ as constructed in Example 2.3.3

(1) Determine the sum of all the elements of $\mathbb{F}_{16}$.

(2) Determine the product of all the nonzero elements of $\mathbb{F}_{16}$.

(3) Determine all the primitive elements of $\mathbb{F}_{16}$.

Problem 2.6.8.

(1) The polynomial $z^4 + z^3 + 1$ has zeroes in $\mathbb{F}_{16}$. How many?

(2) The polynomial $z^4 + z^2 + z$ has zeroes in $\mathbb{F}_{16}$. How many?

Problem 2.6.9. Let $f(x) = (x^2 + x + 1)(x^3 + x + 1)$.
Determine the smallest number m such that $f(x)$ has five zeroes in $\mathbb{F}_{2^m}$.

Problem 2.6.10. What are the possible orders of the elements in $\mathbb{F}_{2^5}$? and in $\mathbb{F}_{2^6}$?

Problem 2.6.11. The mapping $\mathrm{Tr} : \mathbb{F}_{2^m} \to \mathbb{F}_2$ was defined by

$$\mathrm{Tr}(x) = x + x^2 + x^4 + \cdots + x^{2^{m-1}}$$

(1) Show that $\mathrm{Tr}(x) \in \mathbb{F}_2$

(2) Show that $\mathrm{Tr}(x + y) = \mathrm{Tr}(x) + \mathrm{Tr}(y)$

(3) Show that there exists at least one element $\beta \in \mathbb{F}_{2^m}$ such that $\mathrm{Tr}(\beta) = 1$

(4) Show that

$$|\{x \in \mathbb{F}_{2^m} : \mathrm{Tr}(x) = 0\}| = |\{x \in \mathbb{F}_{2^m} : \mathrm{Tr}(x) = 1\}|$$

(and hence $= 2^{m-1}$).

Problem 2.6.12. Consider the finite fields $\mathbb{F}_{q^m}$ and $\mathbb{F}_q$ and define the mapping $\mathrm{Tr} : \mathbb{F}_{q^m} \to \mathbb{F}_q$ by

$$\mathrm{Tr}(x) = x + x^q + x^{q^2} + \cdots + x^{q^{m-1}}$$

(1) Show that $\mathrm{Tr}(x) \in \mathbb{F}_q$

(2) Show that $\mathrm{Tr}(x + y) = \mathrm{Tr}(x) + \mathrm{Tr}(y)$

(3) Show that $\mathrm{Tr}(\gamma x) = \gamma \mathrm{Tr}(x)$ for $\gamma \in \mathbb{F}_q$

(4) Show that $\mathrm{Tr}(x)$ takes on each value of $\mathbb{F}_q$ equally often (i.e., q^{m-1} times). Hint: for $\beta \in \mathbb{F}_q$ consider the polynomial

$$x^{q^{m-1}} + \cdots + x^q + x - \beta.$$

Problem 2.6.13. Factorize $x^9 - 1$ over $\mathbb{F}_2$.

Problem 2.6.14. Factorize $x^{73} - 1$ over $\mathbb{F}_2$.

Problem 2.6.15. Factorize $x^{85} - 1$ over $\mathbb{F}_2$.

Problem 2.6.16. Factorize $x^{18} - 1$ over $\mathbb{F}_2$.

Problem 2.6.17. Is $x^8 + x^7 + x^6 + x^5 + x^4 + x + 1$ an irreducible polynomial over $\mathbb{F}_2$?

Problem 2.6.18.

(1) Show that $f(x) = x^4 + x^3 + x^2 + x + 1$ is irreducible in $\mathbb{F}_2[x]$.

(2) Construct $\mathbb{F}_{16}$ using $f(x)$, that is, explain what the elements are and how to add and multiply them.

(3) Find a primitive element.

(4) Show that the polynomial $z^4 + z^3 + z^2 + z + 1$ has four roots in $\mathbb{F}_{16}$.

Problem 2.6.19. Let $f(x) \in \mathbb{F}_2[x]$ be an irreducible polynomial of degree m. Then $f(x)$ has a zero γ in $\mathbb{F}_{2^m}$.
By solving (1)–(7) below you shall prove the following: $f(x)$ has m different zeroes in $\mathbb{F}_{2^m}$ and these have the same order.

(1) Show that $f(\gamma^2) = f(\gamma^4) = \cdots = f\left(\gamma^{2^{m-1}}\right) = 0$.

(2) Show that $\gamma, \gamma^2, \ldots, \gamma^{2^{m-1}}$ have the same order.

(3) Show that $\gamma^{2^i} = \gamma^{2^j}, j > i \Rightarrow \gamma^{2^{j-i}} = \gamma$. (You can use the fact that $a^2 = b^2 \Rightarrow a = b$)

Let s be the smallest positive number such that $\gamma^{2^s} = \gamma$.

(4) Show that $\gamma, \gamma^2, \ldots, \gamma^{2^{s-1}}$ are different.

(5) Show that $g(x) = (x - \gamma)(x - \gamma^2) \cdots \left(x - \gamma^{2^{s-1}}\right)$ divides $f(x)$.

(6) Show that $g(x) \in \mathbb{F}_2[x]$.

(7) Show that $g(x) = f(x)$ and hence $s = m$.

Problem 2.6.20.

(1) Determine the number of primitive elements of $\mathbb{F}_{32}$.

(2) Show that the polynomial $x^5 + x^2 + 1$ is irreducible over $\mathbb{F}_2$.

(3) Are there elements γ in $\mathbb{F}_{32}$ that have order 15?

(4) Is $\mathbb{F}_{16}$ a subfield of $\mathbb{F}_{32}$?

We construct $\mathbb{F}_{32}$ using the polynomial $x^5 + x^2 + 1$. Let α be an element of $\mathbb{F}_{32}$ that satisfies $\alpha^5 + \alpha^2 + 1 = 0$.

(5) Calculate $(x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8)(x - \alpha^{16})$.

(6) $\alpha^4 + \alpha^3 + \alpha = \alpha^i$. What is i?

$\mathbb{F}_{32}$ can be seen as a vector space over $\mathbb{F}_2$.

(7) Show that the dimension of this vector space is 5.

Let γ be an element of $\mathbb{F}_{32}$, $\gamma \neq 0, 1$.

(8) Show that γ is not a root of a binary polynomial of degree less than 5.

(9) Show that $1, \gamma, \gamma^2, \gamma^3, \gamma^4$ is a basis for the vector space.

(10) What are the coordinates of α^8 with respect to the basis $1, \alpha, \alpha^2, \alpha^3, \alpha^4$?

Problem 2.6.21. Let C be a linear (n, k, d) code over $\mathbb{F}_q$.

(1) Show that d equals the minimal number of linearly dependent columns of a parity matrix H.

(2) What is the maximal length of an $(n, k, 3)$ code over $\mathbb{F}_q$?

(3) Construct a maximal length $(n, k, 3)$ code over $\mathbb{F}_q$ and show that it is perfect, i.e., $1 + (q - 1)n = q^{n-k}$.

Chapter 3

Communication Channels and Error Probability

In this chapter we discuss the performance of error-correcting codes in terms of error probabilities. We derive methods for calculating the error probabilities, or at least we provide upper bounds on these probabilities. Some basic knowledge of probability theory will be assumed, but the presentation is largely self-contained.

The concept or error-correcting codes first appeared in Shannon's paper "A Mathematical Theory of Communication" (1948). In information theory a *information channel* is a model of a communication link or a related system where the input is a message and the output is an imperfect reproduction of it.

It is often useful to simulate decoding in software. In that case the required probability distributions are obtained from suitable random number generators. Some suggestions for programming projects are included in the problem section.

3.1 Probability and entropy

3.1.1 Probability distributions

Assume that the symbols from the binary alphabet $\{0, 1\}$ have probabilities $P(1) = p$, $P(0) = 1 - p$. If the symbols in a string are mutually independent, we have

$$P(x_1, x_2, \ldots, x_n) = \prod_i P(x_i) \tag{3.1}$$

Lemma 3.1.1. *The probability that a string of length n consists of j 1s and $n - j$ 0s is given by the* binomial distribution:

$$P(n, j) = \binom{n}{j} p^j (1 - p)^{n-j}. \tag{3.2}$$

Proof. Follows from (3.1). $\square$

The expected value of $P(n, j)$ is $\mu = np$, and the variance is $\sigma^2 = np(1 - p)$.

When n is large, it may be convenient to approximate the binomial distribution by the *Poisson distribution*

$$P(j) = e^{-\mu}\frac{\mu^j}{j!}, \tag{3.3}$$

where again μ is the expected value and the variance is $\sigma^2 = \mu$. The formula (3.3) may be obtained from (3.2) by letting n go to infinity and p to zero while keeping $\mu = np$ fixed.

3.1.2 Discrete messages and entropy

To discuss how information is transmitted through a channel that introduces errors we need at least a simple model of the messages we want to transmit. We assume that the sender has an unlimited amount of data that he or she wants to send to a receiver. The data are divided into messages, which we usually assume to be strings of independent symbols.

For a *discrete memoryless source* the output is a sequence of independent random variables with the same properties. Each output, X, takes values in a finite alphabet $\{x_1, x_2, \ldots, x_q\}$. The probability of x_j is $P(x_j) = p_j$. It is often convenient to refer to the probability distribution of X as the vector $Q(X) = (p_1, p_2, \ldots, p_q)$. As a measure of the amount of information represented by X we define

Definition 3.1.1. *The entropy of a discrete source X is*

$$H(X) = E\left[-\log P(X)\right] = -\sum_j p_j \log p_j, \tag{3.4}$$

where E indicates the expected value. Usually the logarithms are taken in base 2 in this context, and the (dimensionless) unit of information is a bit.

We note that for an alphabet of q symbols, the maximal value of H is $\log q$, and this value is reached when all symbols have probability $\frac{1}{q}$. For a vector of n independent symbols, the entropy of the vector is n times the entropy of the individual symbols.

Example 3.1.1. The entropy of a discrete source.
Consider a source that has an alphabet of four symbols with probability distribution $\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right)$. If the symbols all had probability $\frac{1}{4}$, the entropy would be $\log_2(4) = 2$ bits. Now we find

$$H = \frac{1}{2}\log 2 + \frac{1}{4}\log 4 + \frac{1}{8}\log 8 + \frac{1}{8}\log 8 = \frac{1}{2} + \frac{1}{2} + \frac{3}{4} = \frac{7}{4}.$$

The source symbols may be represented by on the average $\frac{7}{4}$ binary symbols if we use the source code $a : 0, b : 10, c : 110, d : 111$.

The importance of the information theoretic quantities is justified in particular by the coding theorems, which indicate that sequences of symbols can be mapped into a standard representation with the same information content, usually binary symbols. Thus an encoder maps (a fixed or variable) number of source symbols to a (fixed or variable) number of binary encoded symbols such that (on the average) N source symbols are mapped to approximately $NH(X)$ encoded symbols. The code in Example 3.1.1 is a simple example of a variable-length source code. Most interesting sources are more complicated, and in particular they have memory. Coding for such sources is often referred to as data compression.

3.2 Mutual information and capacity of discrete channels

An information channel is a model of a communication link, a storage medium, or a related system where the input is a string of symbols from the input alphabet, and the output is an imperfect reproduction of it, possibly using a different alphabet. The channel may describe a number of restrictions on the message and various transmission impairments.

In this section we introduce the concept of mutual information as a measure of the amount of information that flows through the channel, and we define the capacity of a channel.

3.2.1 Discrete memoryless channels

In a *discrete memoryless channel* the input and output are sequences of symbols, and the current output depends only on the current input. The channel connects a pair of random variables, (X, Y), with values from finite alphabets $\{x_1, x_2, \ldots, x_r\}$ and $\{y_1, y_2, \ldots, y_s\}$. The channel is described by the conditional probability of y_i given x_j, $P(y_i|x_j) = p_{ji}$. It is often convenient to represent the channel by the transition matrix $Q(Y|X) = [p_{ji}]$. The probability distribution of the output variable, Y, is obtained by multiplying the input distribution by the transition matrix

$$Q(Y) = Q(X)Q(Y|X).$$

As a measure of the amount of information about X represented by Y we define

Definition 3.2.1. *The* mutual information *of the pair* (X, Y) *is*

$$I(X;Y) = E\left[\log \frac{P(y|x)}{P(y)}\right] = \sum_j P(x_j) \sum_i p_{ji} \left[\log p_{ji} - \log P(y_i)\right]. \quad (3.5)$$

We note that for a given X, the mutual information $I(X;Y)$ cannot exceed $H(X)$. This value is reached when Q is the unit matrix, i.e., each value of X corresponds to a unique value of Y, since in that case $P(X,Y) = P(Y)$.

Note also that $I(Y;X) = I(X;Y)$. This symmetry follows from rewriting $P(Y|X)$ as $\frac{P(Y,X)}{P(X)}$ in the definition of I. This is an unexpected property since we tend to think of the flow of information as going from X to Y (carried by a flow of energy). But actually we may interpret I as the amount of information about Y provided by$X$.

Lemma 3.2.1. $I(X;Y) = H(Y) - H(Y|X) = H(X) - H(X|Y)$.

This follows immediately from the definition and is a convenient way of calculating I. Usually it is easier to calculate I from the first form, as indicated on the right side of (3.5). The term $H(Y|X)$ can be found from the transition probabilities of the channel, and $H(Y)$ is found from the output distribution. To use the second form we need to calculate $H(X|Y)$ from the reverse transition probabilities.

Definition 3.2.2. *The* capacity of a discrete channel, $C(Y|X)$, *is the maximum of I with respect to $P(X)$.*

The definition of capacity may appear straightforward, but calculating the capacity analytically is often difficult. In many cases of interest, the symmetry of the transition probabilities suggests that the maximum is obtained for symmetric input probabilities. If necessary, the maximum over a few parameters can be computed numerically.

The single most important channel model is the *binary symmetric channel* (BSC). This channel models a situation where random errors occur with probability p in binary data. The transition matrix is

$$Q = \begin{pmatrix} 1-p & p \\ p & 1-p \end{pmatrix}.$$

For equally distributed inputs, the output has the same distribution, and the mutual information may be found from Lemma 3.2.1 as (in bits/symbol)

$$C = 1 - H(p), \tag{3.6}$$

where H is the binary entropy function

$$H(p) = -p \log p - (1-p)\log(1-p). \tag{3.7}$$

The entropy function is symmetric with respect to $p = \frac{1}{2}$, and reaches its maximal value, 1, there. Thus the capacity of the BSC drops to 0, which is expected since the output is independent of the input. For small p the capacity decreases quickly with increasing p, and for $p = 0.11$, we get $C = \frac{1}{2}$.

For a memoryless channel, we can reach the maximal value of I by using independent inputs. However, it is not clear how such symbols could be used to communicate a message efficiently. If we want to transmit k information bits reliably, we must use the channel at least $n = \frac{k}{C}$ times in order to get enough information. A block code is precisely a rule for selecting a vector of n symbols in such a way that with high probability we can recover the message from the received word.

Example 3.2.1. The *binary erasure channel.* We say that the channel *erases* symbols if it outputs a special character ? in stead of the one sent. If actual errors do not occur, the transition matrix becomes

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

The capacity of this channel, the binary erasure channel (BEC) is easily found from the last version of Lemma 3.2.1 as $C = 1 - p$. Thus the information is simply reduced by the fraction of the transmitted symbols that are erased. If a long (n, k) code is used on the BEC, the syndrome equations provide a system of linear equations that may be solved to give the erased symbols. If close to np symbols are erased and the rate of the code is less than $1 - p$, there are more equations than variables. Since we know that at least the transmitted word is a solution, it is usually unique. However, for any set of j columns of the parity check matrix to be linearly independent, we must have $j < d$, and for a binary code d is much smaller than $n - k$. Thus it is not always possible to correct the erasures.

Similar erasure channels with large alphabets are useful for describing loss of packets in communication networks and segment failures in storage media.

Example 3.2.2. Let the transition matrix of a discrete channel be

$$Q = \begin{pmatrix} \frac{1}{2} & \frac{3}{8} & \frac{1}{8} & 0 \\ 0 & \frac{1}{8} & \frac{3}{8} & \frac{1}{2} \end{pmatrix}.$$

Because of the symmetry, the input symbols may be chosen to have probability $\frac{1}{2}$. The mutual information is then readily calculated from Lemma 3.2.1 as $I = 3\log(3)/4 = 0.594$. We can interpret the channel as one that with probability $1/2$ gives an error free symbol, while with probability $1/2$ we have a BSC with cross over probability $1/4$ and capacity 0.189. If we do not distinguish the reliable outputs (1 and 4) from the unreliable outputs (2 and 3), we get a BSC with cross over probability $1/8$ and capacity 0.456.

Other discrete channels may be used as models of signal processing in modulators/demodulators (modems). Many channel models of practical interest are constructed by assuming that the input symbols are real values, and that the channel adds independent Gaussian noise of zero mean and some variance to the input. In Appendix B we discuss coding for such channels and give the capacity of a Gaussian noise channel.

For codes over larger symbol alphabets, there may be several types of errors with different probabilities. However, if p is the probability that the received symbol is different from the one transmitted, and the errors are mutually independent, then (3.2) and (3.3) can still be used.

3.2.2 Approximations for long sequences

For a long binary (n, k) block code, we expect the number of errors to be close to np. For the code to correct this number of errors it is necessary that the number of syndromes, 2^{n-k}, is at least equal to the number of error patterns (this is the Hamming bound). We relate this result to the capacity of the channel through the following useful approximation to the binomial coefficients:

Lemma 3.2.2.

$$\sum_{j=0}^{m} \binom{n}{j} < 2^{nH(\frac{m}{n})}. \tag{3.8}$$

Proof. For $\frac{m}{n} < \frac{1}{2}$ we can use the binomial expansion to get

$$\begin{aligned}
1 &= \left(\frac{m}{n} + \left(1 - \frac{m}{n}\right)\right)^n \geq \sum_{j=0}^{m} \binom{n}{j} \left(\frac{m}{n}\right)^j \left(1 - \frac{m}{n}\right)^{n-j} \\
&= \sum_{j=0}^{m} \binom{n}{j} \left(\frac{\frac{m}{n}}{1 - \frac{m}{n}}\right)^j (1 - \frac{m}{n})^n \geq \sum_{j=0}^{m} \binom{n}{j} \left(\frac{\frac{m}{n}}{1 - \frac{m}{n}}\right)^m \left(1 - \frac{m}{n}\right)^n \\
&= 2^{-nH(\frac{m}{n})} \sum_{j=0}^{m} \binom{n}{j}
\end{aligned}$$

and the inequality follows. □

Thus for long codes the Hamming bound becomes $H(t/n) < 1 - \frac{k}{n}$, and np errors can be corrected only if $H(p) < 1 - \frac{k}{n}$, i.e. $R = \frac{k}{n} < C$.

The main difficulty in proving that one can get a small probability of decoding error for rates close to capacity is related to the fact that it is not possible to construct long codes that attain the Hamming bound. At the end of this section we show that while it is not possible to correct all errors of weight close to np, such errors can be corrected with high probability.

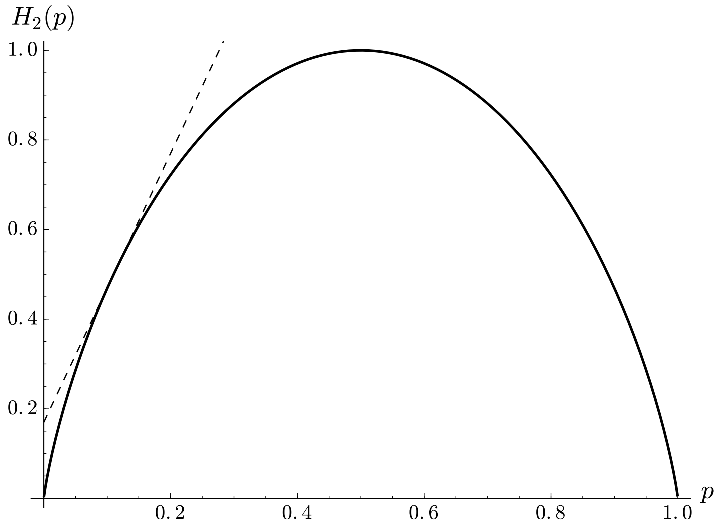

Figure 3.1. The binary entropy function, $H(p)$, and the tangent $T_p(\epsilon)$ (dotted line) for $p = 0.11$.

In applying the binomial distribution to long sequences it is convenient to use the relation

$$p^j(1-p)^{n-j} = 2^{-n(-j/n\log(p)-(1-j/n)\log(1-p))} = 2^{-nT_p(j/n)}. \tag{3.9}$$

Here the function T_p (as a function of j/n) is the tangent to the entropy function in the point $j/n = p$. The slope of the entropy function is readily found to be $\log((1-p)/p)$. Both functions are plotted in Figure 3.1.

3.3 Error probabilities for specific codes

3.3.1 The probability of failure and error for bounded distance decoding

If a linear binary (n, k) code is used for communication, we usually assume that the codewords are used with equal probability, i.e., each codeword c_i has probability

$$P(c_i) = 2^{-k}.$$

If errors occur with probability p, and are mutually independent and independent of the transmitted symbol, we have a *binary symmetric channel*.

In this section we consider *bounded distance decoding*, i.e., all patterns of at most t errors and no other error patterns are corrected. In the particular case where $t = \left\lfloor \frac{d-1}{2} \right\rfloor$ such a decoding procedure is, as in Chapter 1, called *minimum distance decoding*.

If more that t errors occur in bounded distance decoding, the word is either not decoded, or it is decoded to a wrong codeword. We use the term *decoding error* to indicate that the decoder produces a word different from the codeword that was transmitted. We use the term *decoding failure* to indicate that the correct word is not recovered. Thus decoding failure includes decoding error. We have chosen to define decoding failure in this way because the probability of decoding error, P_{err}, is typically much smaller than the probability of decoding failure, P_{fail}. If there is a need for the exact value of the probability that no decoded word is produced, it may be calculated as $P_{\text{fail}} - P_{\text{err}}$.

From (3.2) we get

Theorem 3.3.1. *The probability of decoding failure for bounded distance decoding is*

$$P_{\text{fail}} = 1 - \sum_{j=0}^{t} \binom{n}{j} p^j (1-p)^{n-j} = \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j}. \tag{3.10}$$

The latter sum may be easier to evaluate for small p. In that case one, or a few, terms often give a sufficiently accurate result. Clearly for minimum distance decoding, the error probability depends only on the minimum distance of the code.

If more than t errors occur, a wrong word may be produced by the decoder. This probability may be found exactly for bounded distance decoding, but we consider only the binary case.

Lemma 3.3.1. *Let the zero word be transmitted and the weight of the decoded word be w. If the error pattern has weight j and the distance from the received vector to the decoded word is l, we have*

$$j + l - w = 2i \geq 0.$$

Proof. Since j is the distance from the transmitted word to the received vector, w the distance to the decoded word, and l the distance from the received vector to the decoded word, it follows from the triangle inequality that $i \geq 0$. The error sequence consists of $j - i$ errors among the w nonzero positions of the decoded word and i errors in other coordinates. Thus $l = i + (w - j + i)$. □

We can now find the number of such vectors.

Lemma 3.3.2. *Let the weight of the decoded word be w. Then the number of vectors at distance j from the transmitted word and at distance s from the decoded word is*

$$T(j,l,w) = \binom{w}{j-i}\binom{n-w}{i}. \tag{3.11}$$

Here $i = \frac{j+l-w}{2}$, for $(j+l-w)$ even, and $w-l \le j \le w+l$. Otherwise T is 0.

Proof. From Lemma 3.3.1. □

Once a program for evaluating the function defined in Lemma 3.3.2 is available, or it has been computed for a limited range of parameters, some of the expressions given below ((3.12) and (3.17)) may be readily evaluated.

The weight enumerator for a linear code was defined in Section 1.4 as

$$A(z) = \sum_{w=0}^{n} A_w z^w,$$

where A_w is the number of codewords of weight w. The probability of decoding error is now found by summing (3.11) over all codewords, over j, and $l \le t$.

Theorem 3.3.2. *The error probability for bounded distance decoding on a* BSC *with probability p is*

$$P_{\text{err}} = \sum_{w>0} \sum_{j=w-t}^{w+t} \sum_{l=0}^{t} A_w T(j,l,w) p^j (1-p)^{n-j}. \tag{3.12}$$

Proof. Since the code is linear, we may assume that the zero word is transmitted. The received vector can be within distance t of at most one nonzero codeword, so the probability is found exactly by the summation. □

Theorem 3.3.2 indicates that the weight enumerator contains the information required to find the error probability. However, actually finding the weight enumerator usually requires extensive computations. It is often possible to get a sufficiently good approximation by expanding (3.12) in powers of p and retaining only a few terms.

In some cases it is useful to approximate P_{err} by P_{fail} multiplied by the fraction of syndromes that are decoded. We give this approximation for an alphabet of size q:

$$P_{\text{err}} \simeq P_{\text{fail}}\, q^{-n+k} \sum_{j=0}^{t} \binom{n}{j} (q-1)^j \tag{3.13}$$

Example 3.3.1. Error probability of the $(16, 11, 4)$ extended Hamming code. The code can correct one error. Two errors are detected, but not corrected. Thus we may find the probability of decoding failure from (3.10) as

$$P_{\text{fail}} = 1 - (1-p)^{16} - 16p(1-p)^{15} = 120p^2 + \cdots$$

If p is not very large, decoding errors occur mostly as a result of three errors. From (3.11) we find $T(3,1,4) = 4$, and P_{err} can be found from (3.12) using $A_4 = 140$. Actually three errors are always decoded to a wrong codeword. Thus the first term of the expansion in powers of p is

$$P_{\text{err}} = 560p^3 + \cdots$$

3.3.2 Bounds for maximum likelihood decoding of binary block codes

It is often possible to decode more than $\frac{d}{2}$ errors, but for such algorithms it is much more difficult to calculate the exact error probability. In this section we give some bounds for this case.

The following concept is particularly important:

Definition 3.3.1. Maximum likelihood (ML) *decoding maps any received vector, r, to a codeword c such that the distance $d(r,c)$ is minimized.*

The term likelihood refers to the conditional probability of receiving r given that c is transmitted. An ML decoder selects the codeword that maximizes this likelihood. If we assume that all codewords are used equally often, ML decoding minimizes the probability of decoding error. If the closest codeword is not unique, we choose one of them. The corresponding error patterns are then corrected, while other error patterns of the same weight cause decoding error. In Section 1.3 we discussed how decoding could be based on syndrome tables. In principle, one error pattern in each coset may be decoded, and if these error patterns are known, we could extend the summation in (3.10) and calculate the error probability. However, this approach is feasible only for short codes.

This section gives some upper bounds for the error probability of ML decoding of binary codes. These bounds depend only on the weight enumerator, and as for Theorem 3.3.1 it is often sufficient to know the first terms of the numerator.

Theorem 3.3.3. *For a code with weight enumerator $A(z)$ used on the* BSC *with bit error probability p, an upper bound on the probability of decoding error is*

$$P_{\text{err}} \le \sum_{w>0} \sum_{j>\frac{w}{2}} A_w \binom{w}{j} p^j (1-p)^{w-j} + \frac{1}{2} \sum_{j>0} A_{2j} \binom{2j}{j} p^j (1-p)^j . \quad (3.14)$$

Proof. Since the code is linear, we may assume that the zero word is transmitted. An error occurs if the received vector is closer to some nonzero codeword than to the zero vector. For each codeword of odd weight $w > 0$, the probability that the received codeword is closer to this codeword than to the zero word is

$$\sum_{j>\frac{w}{2}} \binom{w}{j} p^j (1-p)^{w-j},$$

since there must be errors in more than $\frac{w}{2}$ of the positions where the nonzero codeword has a 1. We now get an upper bound on the probability of the union of these events by taking the sum of their probabilities. When w is even and there are errors in half of the positions, there are at least two codewords at distance $\frac{w}{2}$. If a pattern of weight $\frac{w}{2}$ occurs only in a single nonzero codeword, a decoding error is made with probability $\frac{1}{2}$. If the same pattern occurs in more than one word, the error probability is higher, but the pattern is then counted with weight $\frac{1}{2}$ at least twice in (3.14). □

Because most errors occur for j close to $\frac{w}{2}$, we can use this value of j for all error patterns. Since half of all error patterns have weights greater than $\frac{w}{2}$, we now find a simpler approximation

$$P_{\text{err}} < \sum_{w>0} A_w 2^{w-1} (p-p^2)^{\frac{w}{2}}. \tag{3.15}$$

Introducing the function $Z = \sqrt{4p(1-p)}$, which depends only on the channel, we get

$$P_{\text{err}} < \frac{1}{2} \sum_{w>0} A_w Z^w. \tag{3.16}$$

This is a useful bound when the number of errors corrected is fairly large. Equations (3.14) and (3.16) indicate that the error probability depends not only on the minimum distance, but also on the number of low-weight codewords. As p increases, codewords of higher weight may contribute more to the error probability, because their number is much greater.

In (3.14) we overestimate the error probability whenever a vector is closer to more than one nonzero word than to the zero word. Actually even an error pattern of weight $\frac{d}{2}$ may be equally close to the zero word and to several nonzero words, but that usually happens only in relatively few cases.

When calculating the terms in (3.14) we do not specify the value of the error patterns outside the w positions under consideration. Thus if there are several errors in other positions, it is more likely that the vector is also close to another codeword, and vectors of high weight are clearly counted many times (but of course they have small probability). We can get a better bound by keeping track of the weight of the entire error pattern.

The number of error patterns of weight $j > \frac{w}{2}$ that are closer to a particular nonzero word of weight w than to the zero word may be found using Lemma 3.3.2 as

$$\sum_{j \geq \frac{w}{2}} \sum_{l<j} T(j,l,w).$$

We may again take a union bound by multiplying this number by A_w and summing over w. This sum is an upper bound on the number of errors of weight j that may cause a decoding error. As long as the j-th term is less than the total number of weight j vectors, we include it in the bound. However, when j becomes larger, we simply assume that all vectors of weight j cause errors. Thus the bound (*Poltyrev's bound*) becomes

Theorem 3.3.4. *For an* (n,k,d) *code used on the* BSC *with transition probability* p*, an upper bound on the error probability is*

$$\begin{aligned} P_{\text{err}} \leq & \sum_{w>0} A_w \sum_{\frac{w}{2}<j\leq J} \sum_{l<j} T(j,l,w) p^j (1-p)^{n-j} \\ & + \sum_{j>J} \binom{n}{j} p^j (1-p)^{n-j} \\ & + \frac{1}{2} \sum_{0>j>J} \sum_{l<j} A_{2j} T(j,l,2j) p^j (1-p)^{n-j}. \end{aligned} \tag{3.17}$$

Inequality (3.17) holds for any choice of J, but the value should be chosen to minimize the left side. As in (3.14), an extra term has been added to account for error patterns of weight $\frac{w}{2}$. This bound usually gives an excellent approximation to the actual probability of error with maximum likelihood decoding for all p of interest.

Example 3.3.2. Consider the $(15,5,7)$ code with weight distribution $A(z) = 1 + 15z^7 + 15z^8 + z^{15}$. Using (3.14) we get the bound

$$\begin{aligned} P_{\text{err}} \leq 15 & \left[\frac{1}{2}\binom{7}{4} p^4(1-p)^3 + \binom{7}{5} p^5(1-p)^2 + \binom{8}{5} p^5(1-p)^3 + \cdots \right] \\ & + \frac{15}{2}\binom{8}{4} p^4(1-p)^4. \end{aligned}$$

In Theorem 3.3.4 we may include the weight 4 errors in the same way, and we again get 1050 as the first term. The number of error patterns of weight 5 in the expression above far exceeds the number of combinations of five positions among 15. In addition there are error patterns obtained by adding a single error to a weight 4 error pattern in any of the zero coordinates of the codeword. These errors were implicit in the first terms. Thus the Poltyrev bound becomes

$$P_{\text{err}} \leq 1050p^4(1-p)^{11} + \binom{15}{5}p^5(1-p)^{10} + \binom{15}{6}p^6(1-p)^9.$$

A more detailed analysis shows that some 4-tuples are shared between the codewords, and thus the number of weight 4 errors is 945 rather than 1050, and the error probability is overestimated by about 10%. Since the code can correct an error pattern for each of the 1024 cosets, and $1 + 15 + 105 + 455 + 420 = 996$ error patterns of weight $0 - 4$ are corrected, 28 weight-5 errors are also corrected.

3.4 Long codes and channel capacity

The importance of the capacity is related to coding theorems, which indicate that k message bits can be reliably communicated by using the channel a little more than $n = \frac{k}{C}$ times. Thus an encoder maps the k message bits to n encoded symbols using a code consisting of 2^k vectors. A coding theorem states that for any rate $R < C$ (in bits per channel symbol), there is a positive constant, $E(R)$, called the *error exponent*, such that for sufficiently large n there exist codes with error probabilities satisfying

$$P_{\text{err}} < 2^{-nE(R)}. \tag{3.18}$$

For most channels of interest it has not been possible to give direct proofs by constructing good codes. Instead the proofs rely on averages over large sets of codes. We give an outline of a proof for the BSC.

If a class of linear block codes use all nonzero vectors with equal probability, the average weight enumerator is obtained by scaling the binomial distribution to the right number of words.

$$A(z) = 1 + \sum_{w>0} 2^{-n+k}\binom{n}{w}z^w. \tag{3.19}$$

It may be noted that such distributions satisfy (1.5) with dimensions $k' + k'' = n$. For small weight, w, the number of codewords is less than 1, and the Varshamov–Gilbert bound, Theorem 1.2.2, indicates the first weight where the distribution exceeds 1.

Most codes have weight distributions that are quite close to this average. In particular, we note

Lemma 3.4.1. *If a binary code does not include positions that are identically* 0, *the average weight is*

$$E[w] = 2^{-k} \sum_j A_j j = 2^{-k} \sum_{c_i \in C} \sum_j c_{ij} = 2^{-k} \sum_j \sum_{c_i \in C} c_{ij} = n/2, \tag{3.20}$$

where c_{ij} is the j'th position in codeword c_i.

We interchange the summation and use the fact that each position is 0 and 1 equally often. Similarly, we can use the fact that a good code does not contain repeated symbols to prove that all weight distributions have the same variance:

Theorem 3.4.1. *If a code does not have two positions, j' and j'', such that $c_{ij'} = c_{ij''}$ for all i, then the variance of the weight distribution is*

$$\sigma_w^2 = E[w^2] - n^2/4 = n/4. \tag{3.21}$$

Proof. We can express the square of the weight as the sum over products of all pairs of symbols in a word

$$\sum_{j',j''} c_{ij'} c_{ij''} = w^2.$$

If two symbols are not identical (always 00 or 11), they have values 00, 01, 10, and 11 equally often. Thus the mean value of the product is 1/4. Now the summations may be interchanged as in the Lemma 3.4.1, and it follows that the variance always has the same value, $n/4$, as for the binomial distribution with $p = 1/2$. The difference in performance between different codes is mainly due to the relatively small number of codewords of low weight. □

Combining the bound (3.16) with the weight distribution (3.19), we get the following result:

Theorem 3.4.2. *For any rate $R < R_0$ there exist block codes such that the error probability on a* BSC *for sufficiently large block lengths n satisfies*

$$P_{\text{err}} < 2^{-n(R_0 - R)}, \tag{3.22}$$

where

$$R_0 = 1 - \log(1 + Z). \tag{3.23}$$

Proof. From (3.16) and (3.8) we get

$$P_{\text{err}} < \sum_w 2^{-n+k+nH\left(\frac{w}{n}\right)+w\log(Z)}.$$

Here we find the exponent in the bound by taking the largest term, which occurs for

$$\frac{w'}{n} = \frac{Z}{1+Z}$$

The result then follows. □

Thus for a range of code rates, the exponent is a linear function of R. Most errors occur when a received vector is decoded to one at distance w', which is independent of the code rate. These error events dominate the performance, because the product of the number of codewords and the relevant error probability is maximal. Thus a code could have more codewords of lower weight, and still reach approximately the same error rate. For very low rates, the minimum distance of the code could be larger than w', and in that case the performance is determined by the minimum weight codewords.

For higher rates the bound overestimates the number of error patterns, and as in the Poltyrev bound, we get a better result by assuming that all error patterns of a certain weight give errors. An error pattern that is decoded to a codeword at distance w' would typically have relative weight $\delta_c = w'/2n + (1-w'/n)p$, since there are on the average $(n-w')p$ errors in other random positions. This number does not depend on R, but the error probability increases with R, since the number of codewords increases. For a rate R_c, the error probability estimated from (3.22) equals the probability that the received word contains at least $n\delta_c$ errors. After some manipulation of the expressions, one can show that this happens for

$$R_c = 1 - H(\delta_c).$$

Thus the fraction of errors corrected is given by the Hamming bound for this code rate. For higher rates, the contributions to the probability of errors increase with the distance from the transmitted word, and the largest term is given by the probability that the weight of the error pattern exceeds the Hamming bound.

Theorem 3.4.3. *For any rate $R_c < R < 1 - H(p)$ (the capacity of the* BSC)*, for sufficiently large n, the probability of decoding error satisfies*

$$P_{\text{err}} < 2^{-n(T_p(\delta_H) - H(\delta_H))}, \tag{3.24}$$

where the function T was defined in (3.9).

Thus the exponent is the difference between the tangent to H and the entropy function itself, and it is positive for any rate less than the capacity. Moreover, since no code can correct more errors than what the Hamming bound gives, the value is also an upper bound on the exponent. So even though we do not know a specific good code, and no code can correct all errors of weight up to δ_H, we know that the performance of the best codes is virtually the same as for an average code, and it is what we would have if the minimum distance of the code were on the Hamming bound. At this time there is still no known way of achieving such a performance for codes of moderate lengths.

Example 3.4.1. In this example we demonstrate the potential of binary codes by calculating the error exponent and error probability for a moderately long code on a typical BSC. We choose the error probability on the channel to be $p = 0.0417$, so that the channel capacity (3.6) becomes 0.75. If $n = 8192$ binary symbols, we have on the average 342 errors. We find $Z = 0.400$, and thus $w' = 2340$ and $\delta_c = 1415$ (which may seem large). Consider the performance of a code of rate $2/3$, thus $n - k = 2730$. If we use the Hamming bound to get an upper bound on the number of errors that can be corrected with such a code, Section 1.3, we get $t_H = 505$. Note that while the binomial coefficients are very large, we can still calculate their logarithms directly. The approximation (3.8) gives 504, and thus it is also quite accurate in this range. From (3.24) we find the error exponent to be 0.0062 (this form of the exponent is valid for rates > 0.33), and the probability of decoding error is about

$$P_{\text{err}} = 10^{-15}.$$

It should be noted that the minimum distance of a code of length 8192 and rate 0.67 is probably only about 504, by the Gilbert bound. However, the error probability is not increased much by the low-weight codewords. For example, using the approximation to the weight distribution (3.19) we find the expected number of codewords of weight 275 to be about 2^{180}. There are less than 2^{550} ways of choosing 275 errors within each word, but the contribution to the error probability is still negligible, since the probability of each combination is only in the order 2^{-1750}. In Chapter 5 we give a construction that would provide a $(8192, 5461)$ code with distance 426, correcting 212 errors. The reduction in the minimum distance would not necessarily cause a worse performance, but decoding up to only $d/2$ errors is not sufficient on this channel. In later chapters we discuss long codes for which decoding of heavier error patterns is practical. Thus in principle one can get a rate close to channel capacity with a code of moderate length.

3.5 Problems

Problem 3.5.1. A memoryless source has three symbols and probability distribution $\left(\frac{1}{2}, \frac{1}{4}, \frac{1}{4}\right)$.

(1) What is the entropy of the source?

(2) What is the largest entropy of a source with three symbols?

(3) Give a variable-length binary code for the source.

(4) Let u_i be a sequence of independent variables, ± 1, with probability distribution $\left(\frac{1}{2}, \frac{1}{2}\right)$, and let v_i be generated as $v_i = \frac{u_i - u_{i-1}}{2}$. Find the probability distribution of v_i. Is V a memoryless source? Argue that the entropy of V is 1 bit.

Problem 3.5.2. A memoryless channel has transition probabilities

$$Q = \begin{pmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{2} \end{pmatrix}$$

(1) Find the output distribution when the input distribution is $\left(\frac{1}{2}, \frac{1}{2}\right)$.

(2) Find the mutual information between input and output for this distribution.

(3) What is the channel capacity?

Problem 3.5.3. A binary symmetric channel has transition probability $p = 0.05$.

(1) What is the capacity of the channel?

(2) If a code of length 256 is used, what is the average number of errors?

(3) What is the Hamming bound on the number of errors that can be corrected with a (256, 160) code?

(4) What would be the probability of decoding error if such a code could be used?

(5) What is the Gilbert bound for the same parameters?

Problem 3.5.4. The binary Z channel. In some binary channels one of the input symbols is much more likely to be in error than the other. Consider the channel with transition matrix

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

(1) Let the input symbols have probability $\frac{1}{2}$ each. What is the mutual information?

(2) How much bigger is the channel capacity for $p = 0.1$, $p = \frac{1}{4}$, $p = \frac{1}{2}$?

(3) Find an analytic expression.

Problem 3.5.5. A discrete memoryless channel has 17 input and output symbols, both indicated by $[x_1, x_2, x_3, \ldots, x_{17}]$. The probability of error, i.e., the probability that $Y \neq X$, is p.

(1) What is the channel capacity if all other symbols occur with the same probability, $\frac{p}{16}$?

(2) What is the capacity if the only error symbols are $x_{(j \pm 1 \mod 17)}$ when x_j is transmitted, and their probability is $\frac{p}{2}$?

(3) Evaluate the capacities for $p = 0.11$.

Problem 3.5.6. Consider the $(16, 11, 4)$ code used on a BSC with $p = 0.01$.

(1) What are the mean value and the standard deviation of the number of errors in a block?

(2) What are the probabilities of $0, 1, 2, 3$, and 4 errors?

(3) Compare these results to the approximation using the Poisson distribution.

(4) What is the probability of decoding failure?

(5) What is the probability of decoding error (as indicated in Example 3.3.1, this number is closely approximated by the probability of 3 errors)?

Problem 3.5.7. The weight distribution of a $(15, 7)$ code is

$$[1, 0, 0, 0, 0, 18, 30, 15, 15, 30, 18, 0, 0, 0, 0, 1].$$

(1) How many errors can the code correct?

Assume minimum distance decoding.

(2) If the error probability is $p = 0.01$, what is the probability of decoding failure?

(3) How many error patterns of weight 3 cause decoding error (either an exact number or a good upper bound)?

(4) Give an approximate value of the probability of decoding error based on (3.12).

Problem 3.5.8. There exists a $(16, 8, 6)$ code with weight enumerator $A(z) = 1 + 112z^6 + 30z^8 + 112z^{10} + z^{16}$ (this is one of the few non-linear codes we shall mention, but all questions may be answered as for a linear code).

(1) How many errors can the code correct?

(2) What is the probability of decoding failure with $p = 0.01$?

(3) How many error patterns of weight 4 and 6 have distance 2 to a particular codeword of weight 6?

Problem 3.5.9. Consider the distribution of the number of errors in a code of length $n = 256$ and with $p = 0.01$.

(1) What is the average number of errors?

(2) What is the probability that more than eight errors occur?

(3) Is the Poisson distribution sufficiently accurate in this case?

Problem 3.5.10. A $(32, 16, 8)$ code has weight enumerator

$$A(z) = 1 + 620z^8 + 13888z^{12} + 36518z^{16} + 13888z^{20} + 620z^{24} + z^{32}.$$

(1) If maximum likelihood decoding is assumed, find an upper bound on the error probability.

(2) Is (3.16) a useful approximation?

(3) Is the bound significantly improved by using (3.17)?

Note that since all codewords have even weight, half of the cosets contain only error patterns of even weight, and the other half of the cosets only errors of odd weight.

(4) Is it possible that there are $35A_8$ cosets containing 2 error patterns of weight 4?

Problem 3.5.11. A BSC has error probability $p = 0.1$.

(1) Find the channel capacity.

(2) Find the error exponent from equations (3.22) and (3.24). What is the rate where they meet?

(3) Find the error exponent for a code of rate $1/3$ and the approximate probability of decoding failure when $n = 1024$.

Problem 3.5.12. *Project.*

(1) Using a syndrome decoder, simulate minimum distance decoding of a $(32, 16, 8)$ code.

(2) Does the error probability agree with the calculated value?

(3) Similarly, perform ML decoding and compare the results with the theoretical value.

Chapter 4

Reed–Solomon Codes and Their Decoding

Here we start the analysis of codes and decoding using polynomials and their roots. This approach was first used for constructing Reed–Solomon codes (1959). After deriving the parameters of the codes, we start the presentation of algebraic decoding, i.e., decoding based on solving equations. Reed–Solomon codes are not only important as a basis for coding theory, but they are also useful in applications ranging from CDs and DVDs to high-speed optical communications.

4.1 Basic definitions

Before introducing the codes, we will prove an upper bound on the minimum distance of any code.

Theorem 4.1.1. *The Singleton bound*
Let C be an (n, k) code with minimum distance d. Then

$$d \le n - k + 1.$$

Proof. We give three different proofs of the theorem, in one of which we do not assume that the code is linear.

1. Choose the information vector to consist of $k - 1$ zeroes and one non-zero. Then the weight of the corresponding codeword is at most $(n - k) + 1$.
2. The rank of a parity check matrix H for the code is $n - k$ so $n - k + 1$ columns of H are linearly dependent. Therefore the minimum number of dependent columns (i.e., d) must be smaller than or equal to $n - k + 1$.
3. If we delete $d - 1$ fixed positions from all the q^k codewords, they are still different, since each pair differ in at least d positions. There are q^{n-d+1} vectors with the remaining positions and thus $k \le n - d + 1$, and the result follows. □

Definition 4.1.1. *Let $x_1, \ldots, x_n$ be different elements of a finite field $\mathbb{F}_q$. For $k \le n$ consider the set $\mathbb{P}_k$ of polynomials in $\mathbb{F}_q[x]$ of degree less than k. A* Reed–Solomon code *consists of the codewords*

$$(f(x_1), f(x_2), \ldots, f(x_n)), \quad \textit{where} \ \ f \in \mathbb{P}_k.$$

It is clear that the length of the code is $n \leq q$. The code is linear, since if $c_1 = (f_1(x_1), \dots, f_1(x_n))$ and $c_2 = (f_2(x_1), \dots, f_2(x_n))$, then $ac_1 + bc_2 = (g(x_1), \dots, g(x_n))$, where $a, b \in \mathbb{F}_q$ and $g(x) = af_1(x) + bf_2(x)$.

The polynomials in $\mathbb{P}_k$ form a vector space over $\mathbb{F}_q$ of dimension k since there are k coefficients. We now invoke a fundamental theorem of algebra (Theorem 2.2.1) twice: Two distinct polynomials cannot generate the same codeword, since the difference would be a polynomial of degree $< k$ and it cannot have n zeroes, so the dimension of the code is k. A codeword has weight at least $n - k + 1$, since a polynomial of degree $< k$ can have at most $k - 1$ zeroes. Combining this with Theorem 4.1.1 we get

Theorem 4.1.2. *The minimum distance of an* (n, k) *Reed–Solomon code is* $n - k + 1$.

In many applications one takes $x_i = \alpha^{i-1}$, $i = 1, 2, \dots, q - 1$, where α is a primitive element of $\mathbb{F}_q$, so in this case we have $x_i^n = 1$, $i = 1, \dots, n$, and $n = q - 1$.

From the definition of the codes it can be seen that one way of encoding these codes is to take k information symbols $u_0, u_1, \dots, u_{k-1}$ and encode them as

$$(u(x_1), u(x_2), \dots, u(x_n)), \quad \text{where} \quad u(x) = u_{k-1}x^{k-1} + \cdots + u_1 x + u_0.$$

This is a non-systematic encoding; we will describe a systematic encoding in Problem 5.5.9.

Since for Reed–Solomon codes we must have $n \leq q$, there are no interesting binary codes. Codes over $\mathbb{F}_q$ where q is a prime make easier examples and in particular the field $\mathbb{F}_{11}$ is useful for decimal codes, since there is no field with ten elements. However, in most practical cases we have $q = 2^m$.

Example 4.1.1. Reed–Solomon codes over $\mathbb{F}_{11}$.
Since 2 is a primitive element of $\mathbb{F}_{11}$, we can take $x_i = 2^{i-1} \bmod 11, i = 1, 2, \dots, 10$, with $k = 5$ and $u(x) = u_4x^4 + \cdots + u_1x + u_0$; we get as the corresponding codeword

$$(u(1), u(2), u(4), u(8), u(5), u(10), u(9), u(7)u(3), u(6)).$$

So

$(1, 0, 0, 0, 0)$	is encoded into	$(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$
$(0, 1, 0, 0, 0)$	is encoded into	$(1, 2, 4, 8, 5, 10, 9, 7, 3, 6)$
$(0, 0, 1, 0, 0)$	is encoded into	$(1, 4, 5, 9, 3, 1, 4, 5, 9, 3)$
$(0, 0, 0, 1, 0)$	is encoded into	$(1, 8, 9, 6, 4, 10, 3, 2, 5, 7)$
$(0, 0, 0, 0, 1)$	is encoded into	$(1, 5, 3, 4, 9, 1, 5, 3, 4, 9)$

and these five codewords can be used as the rows of a generator matrix of the code, which is a $(10, 5, 6)$ code over $\mathbb{F}_{11}$.

In general a generator matrix for a Reed–Solomon code is

$$G = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \dots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_n^{k-1} \end{bmatrix},$$

and in the case where $x_j = \beta^{j-1}$ with β an element of order n in $\mathbb{F}_q$ we get

$$G = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \beta & \dots & \beta^{n-1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \beta^{k-1} & \dots & \beta^{(n-1)(k-1)} \end{bmatrix}.$$

It now follows from Lemma A.1.1 that

$$H = \begin{bmatrix} 1 & \beta & \dots & \beta^{n-1} \\ 1 & \beta^2 & \dots & (\beta^2)^{n-1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \beta^{n-k} & \dots & (\beta^{n-k})^{n-1} \end{bmatrix}. \tag{4.1}$$

Example 4.1.2. (Example 4.1.1 continued).
A parity check matrix for the $(10, 5, 6)$ code over $\mathbb{F}_{11}$ is then

$$\begin{bmatrix} 1 & 2 & 4 & 8 & 5 & 10 & 9 & 7 & 3 & 6 \\ 1 & 4 & 5 & 9 & 3 & 1 & 4 & 5 & 9 & 3 \\ 1 & 8 & 9 & 6 & 4 & 10 & 3 & 2 & 5 & 7 \\ 1 & 5 & 3 & 4 & 9 & 1 & 5 & 3 & 4 & 9 \\ 1 & 10 & 1 & 10 & 1 & 10 & 1 & 10 & 1 & 10 \end{bmatrix}.$$

4.2 Decoding Reed–Solomon codes

In this section we describe the first of three minimum distance decoding algorithms for Reed–Solomon codes (in the next section we present an algorithm for correcting more errors). We will first present the idea and give a formal algorithm later.

Let $r = c + e$ be a received word, and assume that $w(e) \le t = \left\lfloor \frac{n-k}{2} \right\rfloor$. The idea is to determine a bivariate polynomial

$$Q(x, y) = Q_0(x) + yQ_1(x) \in \mathbb{F}_q[x, y] \setminus \{0\}$$

such that

1. $Q(x_i, r_i) = 0, \quad i = 1, \dots, n.$
2. $\deg(Q_0) \le n - 1 - t.$
3. $\deg(Q_1) \le n - 1 - t - (k - 1).$

The polynomial $Q(x, y)$ is called an interpolation polynomial for the received word. We first prove:

Theorem 4.2.1. *If less than $\frac{d}{2}$ errors have occurred, then there is at least one nonzero polynomial $Q(x, y)$ which satisfies conditions $1 - 3$.*

Proof. The condition 1 gives n homogeneous linear equations in the coefficients and there are $n-1-t+1+n-1-t-(k-1)+1 \geq n+1$ possible coefficients, so indeed the system has a nonzero solution. $\square$

We also have

Theorem 4.2.2. *If the transmitted codeword is generated by $f(x)$ and the number of errors is less than $\frac{d}{2}$, then $f(x) = \frac{-Q_0(x)}{Q_1(x)}$.*

Proof. $c = (f(x_1), \ldots, f(x_n))$ and $r = c + e$ with $w(e) \leq t$. The polynomial $Q(x, y)$ satisfies $Q(x_i, f(x_i) + e_i) = 0$, and since $e_i = 0$ for at least $n - t$ indices i, we see that the univariate polynomial $Q(x, f(x))$ has at least $n - t$ zeroes, namely the x_is where $f(x_i) = r_i$. But $Q(x, f(x))$ has degree at most $n - t - 1$, so $Q(x, f(x)) = 0$ and therefore $Q_0(x) + f(x)Q_1(x) = 0$ i.e., $f(x) = -\frac{Q_0(x)}{Q_1(x)}$. $\square$

The maximal degrees of the components of Q will be used frequently in the following, so we define

$$l_0 := n - 1 - t \quad \text{and} \quad l_1 := n - 1 - t - (k - 1).$$

Note that since $Q(x, y) = Q_1(x)\left(y + \frac{Q_0(x)}{Q_1(x)}\right) = Q_1(x)(y - f(x))$ the x_is where the errors occurred are among the zeroes of $Q_1(x)$, therefore the polynomial $Q_1(x)$ is called an *error locator polynomial.*

The algorithm now can be presented as follows:

Algorithm 4.2.1.
Input: *A received word $r = (r_1, r_2, \ldots, r_n)$*

1. *Find a non-zero solution to the system of linear equations*

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{l_0} & r_1 & r_1x_1 & \ldots & r_1x_1^{l_1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{l_0} & r_2 & r_2x_2 & \ldots & r_2x_2^{l_1} \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{l_0} & r_n & r_nx_n & \ldots & r_nx_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,l_0} \\ Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.2}$$

2. *Put*

$$Q_0(x) = \sum_{j=0}^{l_0} Q_{0,j} x^j$$

$$Q_1(x) = \sum_{j=1}^{l_1} Q_{1,j} x^j$$

$$g(x) = -\frac{Q_0(x)}{Q_1(x)}$$

3. *If* $g(x) \in \mathbb{F}_q[x]$ *and* $d(r, (g(x_1), g(x_2), \ldots, g(x_n))) \leq t$

Output:

$$(g(x_1), g(x_2), \ldots, g(x_n))$$

else

Output: failure

Notice in the system above that each row of the matrix corresponds to a pair (x_i, r_i). We have already seen that if the number of errors is smaller than half the minimum distance, then the output of the algorithm is the sent word.

Example 4.2.1. Decoding the (10, 5, 6) Reed–Solomon code over $\mathbb{F}_{11}$.
We treat the code from Example 4.1.1 and suppose we receive $r = (5, 9, 0, 9, 0, 1, 0, 7, 0, 5)$. We have $l_0 = 7$ and $l_1 = 3$ and therefore we get 10 equations with 12 unknowns. The matrix becomes

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 5 & 5 & 5 & 5 \\ 1 & 2 & 4 & 8 & 5 & 10 & 9 & 7 & 9 & 7 & 3 & 6 \\ 1 & 4 & 5 & 9 & 3 & 1 & 4 & 5 & 0 & 0 & 0 & 0 \\ 1 & 8 & 9 & 6 & 4 & 10 & 3 & 2 & 9 & 6 & 4 & 10 \\ 1 & 5 & 3 & 4 & 9 & 1 & 5 & 3 & 0 & 0 & 0 & 0 \\ 1 & 10 & 1 & 10 & 1 & 10 & 1 & 10 & 1 & 10 & 1 & 10 \\ 1 & 9 & 4 & 3 & 5 & 1 & 9 & 4 & 0 & 0 & 0 & 0 \\ 1 & 7 & 5 & 2 & 3 & 10 & 4 & 6 & 7 & 5 & 2 & 3 \\ 1 & 3 & 9 & 5 & 4 & 1 & 3 & 9 & 0 & 0 & 0 & 0 \\ 1 & 6 & 3 & 7 & 9 & 10 & 5 & 8 & 5 & 8 & 4 & 2 \end{bmatrix}.$$

The system has as a solution $(4, 1, 2, 2, 2, 9, 1, 0, 7, 3, 10, 0)$ corresponding to $Q_0(x) = x^6 + 9x^5 + 2x^4 + 2x^3 + 2x^2 + x + 4$ and $Q_1(x) = 10x^2 + 3x + 7$. We then get $g(x) = x^4 + x^3 + x^2 + x + 1$ corresponding to the codeword $c = (5, 9, 0, 6, 0, 1, 0, 7, 0, 4)$, so we have corrected two errors in positions corresponding to $2^3 (= 8)$ and $2^9 (= 6)$, and one sees that indeed 8 and 6 are the zeroes of $Q_1(x)$.

4.3 A list decoding algorithm

In this section we will extend the previous decoding method for Reed–Solomon codes in a way that allows correction of errors of weight greater than half the minimum distance. In this case the decoder gives a list of closest codewords, so the method is called *list decoding*.

Let $r \in \mathbb{F}_q^n$ be a received word, and suppose r is the sum of a codeword c and an error vector e of weight at most τ. The idea here is an extension of the method presented in the previous section, namely to determine a bivariate polynomial

$$Q(x, y) = Q_0(x) + Q_1(x)y + Q_2(x)y^2 + \cdots + Q_l(x)y^l$$

such that:

1. $Q(x_i, r_i) = 0, \quad i = 1, 2, \ldots, n.$
2. $\deg(Q_j(x)) \le n - \tau - 1 - j(k-1), \quad j = 0, 1, \ldots, l.$
3. $Q(x, y) \neq 0.$

We then have

Lemma 4.3.1. *If $Q(x, y)$ satisfies the above conditions where $r = c + e$ with* weight$(e) \le \tau$ *and $c = (f(x_1), f(x_2), \ldots, f(x_n))$ with* $\deg(f(x)) < k$*, then $(y - f(x)) | Q(x, y)$.*

Proof. The polynomial $Q(x, f(x))$ has degree at most $n - \tau - 1$, but since $r_i = f(x_i)$ except in at most τ cases we have that $Q(x_i, f(x_i)) = 0$ in at least $n - \tau$ cases and therefore $Q(x, f(x)) = 0$. If we consider the polynomial $Q(x, y)$ as a polynomial in y over $\mathbb{F}_q[x]$, we then conclude that $(y - f(x))$ divides $Q(x, y)$. □

This means that we can find all the codewords that are within distance at most τ from the received word by finding factors of the polynomial $Q(x, y)$ of the form $(y - f(x))$ with $\deg(f(x)) < k$. If τ is greater than half the minimum distance it is possible that we get more than one codeword (a list), but since the y-degree of $Q(x, y)$ is at most l, there are at most l codewords on the list. Not all factors may correspond to a codeword within distance τ to the received word.

It is not obvious under which conditions on the numbers τ and l a polynomial $Q(x, y)$ that satisfies the conditions actually exists, but the following analysis addresses this question.

The first condition is a homogeneous system of n linear equations in the coefficients of $Q_0(x), \ldots, Q_l(x)$, so if the number of unknowns is greater than n this system indeed has a non-zero solution. The number of unknowns is

$$\begin{aligned}&(n - \tau) + (n - \tau - (k-1)) + (n - \tau - 2(k-1)) + \cdots + (n - \tau - l(k-1)) \\ &\quad = (l+1)(n - \tau) - \frac{1}{2}l(l+1)(k-1),\end{aligned}$$

so the condition becomes

$$(l+1)(n-\tau) - \frac{1}{2}l(l+1)(k-1) > n, \tag{4.3}$$

where l is the largest integer for which

$$(n-\tau) - l(k-1) > 0 \tag{4.4}$$

in order to ensure that $Q_l(x) \neq 0$.

If $l = 2$ we get $k-1 < (n-\tau)/2$. This is better than $k-1 = n-2\tau$, which we get from decoding up to half the minimum distance, only if

$$\tau > n/3,$$

that is

$$k/n < 1/3 + 1/n. \tag{4.5}$$

This illustrates the general situation: In order to get an improvement on half the minimum distance, the parameters should satisfy

$$\frac{k}{n} < \frac{1}{l+1} + \frac{1}{n},$$

and in this case

$$\tau < n\frac{l}{l+1} - \frac{l}{2}(k-1).$$

The list of l decoding algorithm can be presented as follows.

Algorithm 4.3.1. *Sudan's algorithm.*
Input: *A received word $r = (r_1, r_2, \ldots, r_n)$, and a natural number τ*

1. *Find a non-zero solution to the system of linear equations*

$$\sum_{j=0}^{l} \begin{bmatrix} {r_1}^j & \ldots & 0 & 0 \\ 0 & {r_2}^j & \ldots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \ldots & {r_n}^j \end{bmatrix} \begin{bmatrix} 1 & x_1 & \ldots & x_1^{l_j} \\ 1 & x_2 & \ldots & x_2^{l_j} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & \ldots & x_n^{l_j} \end{bmatrix} \begin{bmatrix} Q_{j,0} \\ Q_{j,1} \\ Q_{j,2} \\ \vdots \\ Q_{j,l_j} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where

$$l_j = n - \tau - 1 - j(k-1).$$

2. *Put*

$$Q_j(x) = \sum_{r=0}^{l_j} Q_{j,r} x^r$$

and

$$Q(x, y) = \sum_{j=0}^{l} Q_j(x) y^j .$$

3. *Find all factors of* $Q(x, y)$ *of the form* $(y - f(x))$ *with* $\deg(f(x)) < k$.

Output: *A list of the factors* $f(x)$ *that satisfy*

$$d\left((f(x_1), f(x_2), \ldots, f(x_n)), (r_1, r_2, \ldots, r_n)\right) \leq \tau$$

The problem of factorization $Q(x, y)$ can be approached in the following way: We want to find polynomials $u(x) = u_{k-1}x^{k-1} + \cdots + u_1 x + u_0$ such that

$$Q(x, u(x)) = \sum_{i=0}^{l_i} \sum_{j=0}^{l} Q_{ij} x^i (u_{k-1}x^{k-1} + \cdots + u_1 x + u_0)^j = 0. \qquad (4.6)$$

The idea is now to consider this equation modulo increasing powers of x; this will make it possible to determine the $u_i's$ recursively. We find a sequence of polynomials $M_i(x, y)$ by substituting the relevant u_i and factoring out a power of x. Initially we set $M_0(x, y) = Q(x, y)$. In the first step we look at the equation modulo x, this is the same as the equation $M_0(0, u_0) = 0$. Here we can suppose that $M(0, y) \neq 0$ for some j, since if not $M_0(x, y) = xM'(x, y)$ and we could use $M'(x, y)$ instead. This means that we can determine u_0 as a zero in $\mathbb{F}_q$ of the polynomial $M_0(0, y)$. We find the M_i recursively by applying the following steps:

1. Find u_i as a root for y in $M_i(0, y)$.
2. Substitute $xy + u_i$ for y in M_i to cancel the constant term.
3. Factor out the largest possible power of x, x^{m_i}.
4. $M_{i+1} = x^{-m_i} M_i(x, xy + u_i)$.

After finding u_{k-1}, test that $M_{k-1}(x, u_{k-i})$ is identically 0 to verify that $y + u(x)$ is actually a factor. Observe that the y degrees of $M_i(x, y)$ are the same for all i, so m_i is well defined. The coefficients u_i can be found by solving an equation of degree l, but as we shall see the total number of solutions $u(x)$ is at most l. This can be concluded from the following

Lemma 4.3.2. *Let* β *be a zero of* $M_i(0, y)$ *of multiplicity* m_β. *Define*

$$M_{i+1}(x, y) = x^{-m_i} M_i(x, xy + \beta),$$

where m_i *is the largest integer such that* x^{m_i} *divides* $M_i(x, xy + \beta)$. *Then* $\deg_y M_{i+1}(0, y) \leq m_\beta$.

Proof. Let $\hat{M}(x, y) = M_i(x, y + \beta) = \sum_{j=0}^{l} q_j(x)y^j$. Then $q_j(0) = 0$ for $0 \leq j < m_\beta$ and $q_{m_\beta} \neq 0$, or equivalently x divides $q_j(x)$ for $0 \leq j < m_\beta$, but it does not divide $q_{m_\beta}(0)$. This means that x divides $\hat{M}(x, xy)$, but $x^{m_\beta+1}$ does not, so $m_i \leq m_\beta$. Since $M_{i+1}(x, y) = x^{-m_i} M_i(x, xy + \beta) = \sum_{j=m_\beta}^{l} q_j(x)x^{j-m}y^j$, we get $M_{i+1}(0, y) = \sum_{j=m_\beta}^{l} (q_j(x)x^{j-m})|_{x=0}y^j$, so $\deg_y M_{i+1}(0, y) \leq m_\beta$. □

Corollary 4.3.1. *The number of different $u(x)'s$ is at most l.*

Proof. Denote by U_i the set of all solutions $u = (u_0, \ldots, u_i)$ the algorithm finds after i steps. We can describe the set of solutions as a tree. From the root, we have at most l solutions for u_0 defining a set of nodes. For each such value, we may branch out with different values of u_1, etc. Each node has a multiplicity indicating the multiplicity of u_i as a root of $M_i(0, y)$. It now follows by induction on i that the sum of the multiplicities over all nodes of depth i is $\leq l$. □

Example 4.3.1. List of 2 decoding of a $(15, 3)$ Reed–Solomon code over $\mathbb{F}_{16}$. Let α be a primitive element of $\mathbb{F}_{16}$, where $\alpha^4+\alpha+1 = 0$, and consider the $(15, 3)$ Reed–Solomon code obtained by evaluating polynomials of degree at most 2 in the powers of α. The code has minimum distance 13 and is thus 6-error correcting. However, using the above with $l = 2$ we see that it is possible to decode up to seven errors with list size at most 2.
If $r = (\alpha^4, \alpha^5, 1, \alpha^{11}, \alpha^{10}, \alpha^8, \alpha^{13}, \alpha^7, \alpha^2, 1, \alpha^5, \alpha^5, 0, \alpha^{10})$ is received, one finds $Q(x, y) = x^4 + x^3 + x^2 + \alpha x + \alpha^5 + (x + 1)y + y^2$. Using the factorization algorithm one finds the two solutions $(u_0, u_1.u_2) = (\alpha, 0, 1)$ and $(u_0, u_1.u_2) = (\alpha^4, 1, 1)$, so we get $Q(x, y) = (y + x + \alpha)(y + x^2 + x + \alpha^4)$ and then the corresponding two codewords are

$$\left(\alpha^4, \alpha^5, 1, \alpha^{11}, \alpha^{10}, \alpha^8, \alpha^{13}, \alpha^7, 0, \alpha^9, \alpha^2, \alpha^{14}, \alpha^3, \alpha^6, \alpha^{12}\right) \quad \text{and}$$
$$\left(\alpha^4, \alpha^8, \alpha^2, \alpha^{10}, \alpha^8, \alpha, 0, 1, \alpha^2, 1, \alpha, \alpha^5, \alpha^5, 0, \alpha^{10}\right).$$

4.4 Another decoding algorithm

In this section we present another version of minimum distance decoding of RS codes (historically this was the first such algorithm).

We treat the case where $x_i = \beta^{i-1}$, with β an element of order n in $\mathbb{F}_q$.

Here the decoding problem is split into two stages. First we find an error locator polynomial, i.e., $Q_1(x)$, and then we determine the error values.

Let $r = c + e$ be a received word with $w(e) < \frac{d}{2}$. Let the syndrome $S = (S_1, S_2, \ldots, S_{n-k})$ be

$$S = Hr^T,$$

where H is the parity check matrix (4.1). With $r = (r_1, r_2, \ldots, r_n)$ and $r(x) = r_n x^{n-1} + \cdots + r_2 x + r_1$, this means that $S_i = r(\beta^i) = e(\beta^i)$. In the following we also refer to the individual S_i's as syndromes.

Theorem 4.4.1. *An error locator Q_1 is a solution to the system of linear equations*

$$\begin{bmatrix} S_1 & S_2 & \ldots & S_{l_1+1} \\ S_2 & S_3 & \ldots & S_{l_1+2} \\ \vdots & \vdots & \ldots & \vdots \\ S_{l_1} & S_{l_1+1} & \ldots & S_{2(l_1)} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

Proof. We use the results of Section 4.2 to prove first that if $Q_1(x)$ is an error locator, then the system of equations is satisfied, and then if we have a solution to the system of equations we can get the interpolating polynomial $Q(x, y)$.

We note that the system of equations to determine $Q(x, y) = Q_0(x) + Q_1(x)y$ can be written as

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{l_0} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{l_0} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^{l_0} \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,l_0} \end{bmatrix} + \begin{bmatrix} r_1 & r_1 x_1 & \ldots & r_1 x_1^{l_1} \\ r_2 & r_2 x_2 & \ldots & r_2 x_2^{l_1} \\ \vdots & \vdots & \ldots & \vdots \\ r_n & r_n x_n & \ldots & r_n x_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (4.7)$$

Since the first part of this system is independent of the received word we can remove it by multiplying the system with

$$B = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & \ldots & \vdots \\ x_1^{l_1} & x_2^{l_1} & \ldots & x_n^{l_1} \end{bmatrix},$$

it follows from Section A.1 that we get

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \dots & \vdots \\ x_1^{l_1} & x_2^{l_1} & \dots & x_n^{l_1} \end{bmatrix} \begin{bmatrix} r_1 & r_1x_1 & \dots & r_1x_1^{l_1} \\ r_2 & r_2x_2 & \dots & r_2x_2^{l_1} \\ \vdots & \vdots & \dots & \vdots \\ r_n & r_nx_n & \dots & r_nx_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

This is the same as

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \dots & \vdots \\ x_1^{l_1} & x_2^{l_1} & \dots & x_n^{l_1} \end{bmatrix} \begin{bmatrix} r_1 & \dots & 0 & 0 \\ 0 & r_2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & r_n \end{bmatrix}$$
$$\times \begin{bmatrix} 1 & x_1 & \dots & x_1^{l_1} \\ 1 & x_2 & \dots & x_2^{l_1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_n & \dots & x_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

If we let

$$D(r) =$$
$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \dots & \vdots \\ x_1^{l_1} & x_2^{l_1} & \dots & x_n^{l_1} \end{bmatrix} \begin{bmatrix} r_1 & \dots & 0 & 0 \\ 0 & r_2 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & r_n \end{bmatrix} \begin{bmatrix} 1 & x_1 & \dots & x_1^{l_1} \\ 1 & x_2 & \dots & x_2^{l_1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_n & \dots & x_n^{l_1} \end{bmatrix}$$

then an easy calculation shows that $d_{ij} = S_{i+j-1}, i = 1, \dots, l_1\ j = 1, \dots, l_1 + 1$, and that $D(r) = D(e)$.

So $Q_1(x)$ is a solution to the system

$$\begin{bmatrix} S_1 & S_2 & \dots & S_{l_1+1} \\ S_2 & S_3 & \dots & S_{l_1+2} \\ \vdots & \vdots & \dots & \vdots \\ S_{l_1} & S_{l_1+1} & \dots & S_{2(l_1)} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \qquad (4.8)$$

On the other hand if $Q_1(x)$ solves (4.8) we have that

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \dots & \vdots \\ x_1^{l_1} & x_2^{l_1} & \dots & x_n^{l_1} \end{bmatrix} \begin{bmatrix} r_1 & r_1x_1 & \dots & r_1x_1^{l_1} \\ r_2 & r_2x_2 & \dots & r_2x_2^{l_1} \\ \vdots & \vdots & \dots & \vdots \\ r_n & r_nx_n & \dots & r_nx_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

and therefore the vector

$$\begin{bmatrix} r_1 & r_1x_1 & \dots & r_1x_1^{l_1} \\ r_2 & r_2x_2 & \dots & r_2x_2^{l_1} \\ \vdots & \vdots & \dots & \vdots \\ r_n & r_nx_n & \dots & r_nx_n^{l_1} \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ \vdots \\ Q_{1,l_1} \end{bmatrix}$$

is in the nullspace of B. This, as follows from (4.7), is contained in the space spanned by the columns of the matrix

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{l_0} \\ 1 & x_2 & x_2^2 & \dots & x_2^{l_0} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{l_0} \end{bmatrix},$$

and therefore (4.7) has a solution $Q_0(x)$. □

The above argument holds for any solution $Q_1(x)$, so in the case where there is a choice we select the *solution of lowest degree.*

Once $Q_1(x)$ has been determined, its zeroes $\beta^{i_1}, \beta^{i_2}, \dots \beta^{i_t}$ are found, usually by testing all q elements of $\mathbb{F}_q$. Since $He^T = (S_1, S_2, \dots, S_{2(l_1)})$, the error values can then be found by solving the system of equations.

$$\begin{bmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_t} \\ x_{i_1}^2 & x_{i_2}^2 & \dots & x_{i_t}^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_{i_1}^t & x_{i_2}^t & \dots & x_{i_t}^t \end{bmatrix} \begin{bmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_t} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_t \end{bmatrix} \tag{4.9}$$

By Cramer's rule,

$$e_{i_S} = \frac{\begin{vmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_{S-1}} & S_1 & x_{i_{S+1}} & \dots & x_{i_t} \\ x_{i_1}^2 & x_{i_2}^2 & \dots & x_{i_{S-1}}^2 & S_2 & x_{i_{S+1}}^2 & \dots & x_{i_t}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i_1}^t & x_{i_2}^t & \dots & x_{i_{S-1}}^t & S_t & x_{i_{S+1}}^t & \dots & x_{i_t}^t \end{vmatrix}}{\begin{vmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_{S-1}} & x_{i_S} & x_{i_{S+1}} & \dots & x_{i_t} \\ x_{i_1}^2 & x_{i_2}^2 & \dots & x_{i_{S-1}}^2 & x_{i_S}^2 & x_{i_{S+1}}^2 & \dots & x_{i_t}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i_1}^t & x_{i_2}^t & \dots & x_{i_{S-1}}^t & x_{i_S}^t & x_{i_{S+1}}^t & \dots & x_{i_t}^t \end{vmatrix}},$$

and therefore using Corollaries A.1.1 and A.1.2

$$e_{i_S} = \frac{x_{i_1} \dots x_{i_{S-1}} x_{i_{S+1}} \dots x_{i_t} \prod_{1 \le l < S < j \le t} (x_{i_j} - x_{i_l})}{x_{i_1} \dots x_{i_t} \prod_{1 \le l \le j \le t} (x_{i_j} - x_{i_l})} \sum_{r=1}^{t} P_r^{(S)} S_r$$

$$= \frac{\sum_{r=1}^{t} P_r^{(S)} S_r}{P^{(S)}(x_{i_S})}. \tag{4.10}$$

We now have the following

Algorithm 4.4.1. *Peterson's algorithm.*
Input: *A received word* $r = (r_1, r_2, \dots, r_n)$

1. *Calculate the syndromes* $S_i = r\left(\beta^i\right), i = 1, 2, \dots, n-k$*, where* $r(x) = r_{n-1}x^{n-1} + \dots + r_1 x + r_0$.
2. *Find the solution* $Q_1(x)$ *of lowest degree to the system*

$$\begin{bmatrix} S_1 & S_2 & \dots & S_{l_1+1} \\ S_2 & S_3 & \dots & S_{l_1+2} \\ \vdots & \vdots & \dots & \vdots \\ S_{l_1} & S_{l_1+1} & \dots & S_{2l_1} \end{bmatrix} \begin{bmatrix} q_{1,0} \\ q_{1,1} \\ \vdots \\ q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

3. *Find the zeroes of* $Q_1(x), \beta^{i_1}, \dots, \beta^{i_t}$*, say.*
4. *Find the error values by solving the system* (4.9) *or use the Formula* (4.10).

Output: *The error vector* $(e_1, e_2, \dots, e_n)$

In the above calculations we have supposed that d is odd. If d is even, l_1 should be replaced by $l_1 - 1$.

Example 4.4.1. (Example 4.2.1 continued). We use the $(10, 5, 6)$ code over $\mathbb{F}_{11}$ and receive $r = (5, 9, 0, 9, 0, 1, 0, 7, 0, 5)$.
With $r(x) = 5x^9 + 7x^7 + x^5 + 9x^3 + 9x + 5$ we get the syndromes

$$S_1 = r(2) = 8, S_2 = r(4) = 8, S_3 = r(8) = 3, S_4 = r(5) = 10.$$

The corresponding system of equations is

$$\begin{bmatrix} 8 & 8 & 3 \\ 8 & 3 & 10 \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

A solution is $Q_1(x) = 10x^2 + 3x + 7$ with zeroes $8 (= 2^3)$ and $6 (= 2^9)$, so the error polynomial has the form $bx^9 + ax^3$. To find a and b we get from the equation $He^T = S$ the two equations $8a + 6b = 8$ and $9a + 3b = 8$ which give $a = 3$ and $b = 1$ so the error polynomial is $x^9 + 3x^5$, and therefore the codeword is $c(x) = w(x) - e(x) = 4x^9 + 7x^7 + x^5 + 6x^3 + 9x + 5$ corresponding to the result obtained in Example 4.2.1.

4.5 Problems

Problem 4.5.1.

(1) Find two binary codes of length 4 that satisfy Theorem 4.1.1 with equality.

(2) Find a $(4, 2)$ ternary code that satisfies Theorem 4.1.1 with equality.

Problem 4.5.2. Consider a $(6, 4)$ Reed–Solomon code over $\mathbb{F}_7$ where $x_i = 3^{i-1}$.

(1) Find a generator matrix for the code using the polynomials $1, x, x^2$, and x^3.

(2) Find a generator matrix in systematic form. Which polynomials generate the rows?

(3) What is the minimum distance?

(4) Find a codeword from $f(x) = x^3 + x$ and add 2 to position 3.

(5) What are the degrees of $Q_0(x)$ and $Q_1(x)$ and how many coefficients are there in $Q(x, y)$?

(6) Use the decoding algorithm of Section 4.2 to correct the error (Algorithm 4.2.1).

(7) Find a parity check matrix of the code.

(8) How are single errors corrected from the two syndromes?

Problem 4.5.3. Consider a $(7, 3)$ Reed–Solomon code over $\mathbb{F}_8$ with $x_i = \alpha^{i-1}$, where α is a primitive element.

(1) What is the codeword corresponding to $f(x) = x^2 + x$?

(2) Add a new position by including $x_8 = 0$. What are the parameters of the new code?

(3) Add two errors to the codeword from (1) and use the decoding algorithm from Section 4.2.

(4) Find a parity check matrix for the code.

(5) How can you tell from the syndrome whether a received word contains one or two errors?

Problem 4.5.4. Consider the (10,3,8) Reed–Solomon code over $\mathbb{F}_{11}$ with $x_i = 2^{i-1}, i = 1, 2, \ldots, 10$.

(1) How many errors can be corrected with list size 2?

(2) If $(0, 0, 6, 9, 1, 6, 0, 0, 0, 0)$ is received and we decode with list size 2, show that $Q(x, y) = y^2 - y(x^2 - 3x + 2)$.

(3) What are the possible sent codewords?

Problem 4.5.5.

(1) Find the error locator for the received word in problem 4.5.3 (3) using Peterson's algorithm (Algorithm 4.4.1).

(2) Check that the right positions are roots.

(3) Calculate the error values.

Problem 4.5.6.

(1) Show that 3 is a primitive element of $\mathbb{F}_{17}$.

Consider a Reed–Solomon code over $\mathbb{F}_{17}$ with $x_i = 3^{i-1}$ $i = 1, \ldots, 16$, where we encode $u(x) = u_9 x^9 + \cdots + u_1 x + u_0$.

(2) What are the parameters of this code?

(3) If we include 0 as a point and take as codewords

$$(u(0), u(x_1), \ldots, u(x_{16})),$$

what are then the parameters of this code?

Add the information symbol i_9 as an extra code position. (One may think of this as $u(\infty)$; why?)

(4) If $u_9 = 0$, how many other positions can be 0? Find the parameters of this code.

We will use the code from (3) to correct three errors.

(5) What are the coefficients of the interpolating polynomial $Q(x, y)$, where $y = r(x)$ is the received word? Set up the equations to determine $Q(x, y)$ assuming $u(x) = 0$. Try also $u(x) = 1$.

Problem 4.5.7. Reed–Solomon code over $\mathbb{F}_{16}$ (*Project*).
Let α be a primitive element of $\mathbb{F}_{16}$ satisfying $\alpha^4 + \alpha + 1 = 0$ and consider the $(15, 9, 7)$ Reed–Solomon code over $\mathbb{F}_{16}$ determined by $x_i = \alpha^{i-1}$ $i = 1, 2, \ldots, 15$.

(1) Write a program that generates the powers of α, a list of inverses, and a multiplication table (represent the field elements as integers from 0 to 15).

(2) Using the results of (1) to encode the information sequence $1, 1, 1, 1, 1, 1, 1, 1, 1$.

(3) Similarly, write a program for calculating the syndromes of the received word $\left(\alpha^4, \alpha, \alpha^2, \alpha^3, \alpha, \alpha^5, \alpha^6, \alpha^7, \alpha, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}\right)$.

(4) Write a program for solving linear equations in $\mathbb{F}_{16}$. Use this program to find the error locator.

(5) Find the error positions by evaluating the error locator for each element in $\mathbb{F}_{16}$.

(6) Find the error values by solving a system of linear equations.

Chapter 5

Cyclic Codes

The description of codes by polynomials, as started in the previous chapter, leads to a nice mathematical structure called a cyclic code. Most of the best codes, as least for moderate lengths, are cyclic or closely related to cyclic codes. In this chapter we describe some important classes of cyclic codes and find their parameters, or at least estimates.

5.1 Introduction to cyclic codes

Definition 5.1.1. *An (n, k) linear code C over $\mathbb{F}_q$ is called* cyclic *if any cyclic shift of a codeword is again a codeword, i.e., if*

$$c = (c_0, c_1, \dots, c_{n-1}) \in C \Longrightarrow \widehat{c} = (c_{n-1}, c_0, \dots, c_{n-2}) \in C.$$

Example 5.1.1. The $(7, 3)$ code over $\mathbb{F}_2$ that consists of the codewords

$$(0,0,0,0,0,0,0), (1,0,1,1,1,0,0), (0,1,0,1,1,1,0), (0,0,1,0,1,1,1),$$
$$(1,0,0,1,0,1,1), (1,1,0,0,1,0,1), (1,1,1,0,0,1,0), (0,1,1,1,0,0,1)$$

can be seen to be a cyclic code.

The properties of cyclic codes are more easily understood if we treat words as polynomials in $\mathbb{F}_q[x]$. This means that if $(a_{n-1}, \dots, a_1, a_0) \in \mathbb{F}_q^n$ we associate the polynomial $a(x) = a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 \in \mathbb{F}_q[x]$.

In the following we will not distinguish between codewords and code polynomials.

The first observation is

Lemma 5.1.1. *If*

$$c(x) = c_{n-1}x^{n-1} + \cdots + c_1 x + c_0$$

and

$$\widehat{c}(x) = c_{n-2}x^{n-1} + \cdots + c_0 x + c_{n-1},$$

then

$$\widehat{c}(x) = xc(x) - c_{n-1}(x^n - 1).$$

The lemma is proved by direct calculation.

Theorem 5.1.1. *Let C be a cyclic (n, k) code over $\mathbb{F}_q$ and let $g(x)$ be the monic polynomial of lowest degree in $C\backslash\{0\}$.*

Then

1. *$g(x)$ divides $c(x)$ for every $c \in C$.*
2. *$g(x)$ divides $x^n - 1$ in $\mathbb{F}_q[x]$.*
3. *$k = n - \deg(g(x))$.*

We first note that $g(x)$ is uniquely determined, since if there were two, their difference (since the code is linear) would have lower degree and would be a codeword.

Proof of the theorem. It is clear from the definition of $g(x)$ that $k \leq n - \deg(g(x))$, since there are only $n - \deg(g(x))$ positions left.

If $g(x)$ has degree s, then $g(x) = g_{s-1}x^{s-1} + \cdots + g_1x + g_0 + x^s$, so from Lemma 5.1.1 we get that $x^j g(x)$ is in C if $j \leq n - 1 - s$. Therefore, $a(x)g(x)$, where $\deg(a(x)) \leq n - 1 - s$ are also codewords of C.

It is also easy to see that $x^j g(x)$, where $j \leq n-1-s$, are linearly independent codewords of C, so $k \geq n - s$, and we then have $k = n - s$, proving 3.

To prove 1., suppose $c(x) \in C$; then $c(x) = a(x)g(x) + r(x)$, where $\deg(r(x)) < \deg(g(x))$. Since $\deg(a(x)) \leq n - 1 - s$, we have that $a(x)g(x)$ is a codeword and therefore that $r(x) = c(x) - a(x)g(x)$ is also in the code. Since $\deg(r(x)) < \deg(g(x))$, this implies that $r(x) = 0$ and therefore $g(x)$ divides $c(x)$, and 1. is proved.

2. follows directly from the lemma since $g(x)$ divides $c(x)$ and also $\widehat{c}(x)$. □

The polynomial $g(x)$ in the theorem is called the *generator polynomial* for the cyclic code C.

Example 5.1.2. (Example 5.1.1 continued). We see that $g(x) = x^4+x^3+x^2+1$ and that the codewords all have the form $(a_2x^2 + a_1x + a_0)g(x)$ where $a_i \in \mathbb{F}_2$ and that $x^7 - 1 = (x^4 + x^3 + x^2 + 1)(x^3 + x^2 + 1)$.

It follows from the above that the codewords in a cyclic code have the form $c(x) = u(x)g(x)$, where $g(x)$ is a divisor of $x^n - 1$. On the other hand, we have

Theorem 5.1.2. *Suppose $g(x) \in \mathbb{F}_q[x]$ is monic and divides $x^n - 1$.*

Then $C = \{u(x)g(x) | u(x) \in \mathbb{F}_q[x], \deg(u(x)) < n - \deg(g(x))\}$ is a cyclic code with generator polynomial $g(x)$.

Proof. It is obvious that C is a linear code, and if it is cyclic, then the generator polynomial is $g(x)$ and hence the dimension is $n - \deg(g(x))$ so we only have to prove that C is cyclic.

To this end let $g(x) = x^s + g_{s-1}x^{s-1} + \cdots + g_1x + g_0$ and $h(x) = \frac{(x^n-1)}{g(x)} = x^{n-s} + h_{n-s-1}x^{n-s-1} + \cdots + h_1x + h_0$.

Let $c(x) = u(x)g(x)$ where $\deg(u(x)) < n - s$, then

$$\begin{aligned}\widehat{c}(x) &= xc(x) - c_{n-1}(x^n - 1) = xu(x)g(x) - c_{n-1}h(x)g(x)\\ &= (xu(x) - c_{n-1}h(x))g(x).\end{aligned}$$

Now $c_{n-1} = u_{n-s-1}$ so indeed $(xu(x)-c_{n-1}h(x))$ has deg $< n-s$ and therefore $\widehat{c}(x)$ is also in C. $\square$

The two theorems combined tell us that we can study cyclic codes by studying the divisors of $x^n - 1$. In the case $q = 2$ and n odd we gave a method for finding divisors of $x^n - 1$ in Section 2.4.

Example 5.1.3. Binary cyclic codes of length 21.
Using the algorithm of Section 2.4 we have

$$\begin{aligned}x^{21} - 1 = (x - 1)&(x^6 + x^4 + x^2 + x + 1)(x^3 + x^2 + 1)\\ &\cdot (x^6 + x^5 + x^4 + x^2 + 1)(x^2 + x + 1)(x^3 + x + 1).\end{aligned}$$

With $g(x) = (x^6 + x^4 + x^2 + x + 1)(x^3 + x^2 + 1)$ we get a $(21, 12)$ binary code. With $g_1(x) = (x^6 + x^4 + x^2 + x + 1)(x^3 + x + 1)$ we also get a $(21, 12)$ code.

5.2 Generator and parity check matrices of cyclic codes

Let C be an (n, k) cyclic code over $\mathbb{F}_q$. As proved in Section 5.1, the code C has a generator polynomial $g(x)$ of degree $n - k$, that is, $g(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \cdots + g_1x + g_0$, and we also saw that $x^j g(x)$, $j = 0, 1, \ldots, k-1$ gave linearly independent codewords. This means that a generator matrix of C is

$$G = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots \\ 0 & g_0 & g_1 & \cdots \\ 0 & 0 & g_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

So G has as its first row the coefficients of $g(x)$ and the remaining $k - 1$ rows are obtained as cyclic shifts.

To get a parity check matrix we observe that $g(x)h(x) = x^n - 1$, since $h(x)$ was defined exactly in this way and, if $c(x) = u(x)g(x)$, we get that $c(x)h(x) = u(x)g(x)h(x) = u(x)(x^n-1)$. Hence, the polynomial $c(x)h(x)$ does not contain any terms of degrees $k, k+1, \ldots, n-1$, and therefore $\sum_{i=0}^{n-1} c_i h_{j-i} = 0$ for $j = k, k+1, \ldots, n-1$, where $h_s = 0$ if $s < 0$.

This shows that the vectors

$$(h_k, h_{k-1}, \ldots, h_0, 0, \ldots, 0), \ldots, (0, \ldots, h_k, h_{k-1}, \ldots, h_0)$$

give $n-k$ independent parity check equations, so a parity check matrix is

$$H = \begin{pmatrix} h_k & h_{k-1} & h_{k-2} & \ldots \\ 0 & h_k & h_{k-1} & \ldots \\ 0 & 0 & h_k & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

Thus, H has as its first row the coefficients of $h(x)$ in reverse order and the remaining $n-k-1$ rows are cyclic shifts of the first row. Therefore, we have

Theorem 5.2.1. *If C is an (n, k) cyclic code with generator polynomial $g(x)$, then the dual code $C^\perp$ is also cyclic, and has generator polynomial $g^\perp(x) = h_0 x^k + \cdots + h_{k-1}x + h_k = x^k h(x^{-1})$, where $h(x) = \frac{x^n - 1}{g(x)}$.*

Example 5.2.1. For the code considered in Example 5.1.1 we get

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

and

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

5.3 Cyclic Reed–Solomon codes and BCH codes

In this section we shall see that a large class of the Reed–Solomon codes as defined in Chapter 4 are cyclic codes and we will determine their generator polynomials. We will also consider (in the case where $q = 2^m$), the subcode consisting of the binary words of the original code and show that this is also a cyclic code with a generator polynomial that is easy to determine. It then follows from the definition that indeed we have a lower bound on the minimum distance. We shall also indicate how the Peterson's decoding algorithm works for decoding these binary codes.

5.3.1 Cyclic Reed–Solomon codes

Let α be a primitive element of $\mathbb{F}_q$, n be a divisor of $q-1$, and $\beta = \alpha^{\frac{q-1}{n}}$. Let C_s be the Reed–Solomon code with $x_i = \beta^{(i-1)}$, $i = 1, 2, \ldots, n$, obtained by evaluating polynomials in $\mathbb{P}_s = \{f(x) \in \mathbb{F}_q[x] | \deg(f(x)) < s\}$.

Theorem 5.3.1. *C_s is a cyclic (n, s) code over $\mathbb{F}_q$ with generator polynomial*

$$g(x) = (x-\beta)(x-\beta^2)\cdots(x-\beta^{n-s}).$$

Proof. The code is cyclic, since if $c = (f(\beta^0), f(\beta), \ldots, f(\beta^{(n-1)}))$, then $\hat{c} = (f(\beta^{(n-1)}), f(\beta^0), f(\beta), \ldots, f(\beta^{(n-2)})$, so if we define $f_1(x) = f(\beta^{-1}x)$, then $\hat{c} = (f_1(\beta^0), f_1(\beta), \ldots, f_1(\beta^{(n-1)}))$, because $\beta^n = 1$.

It follows from Lemma A.1.1 that a parity check matrix for the code C_s is

$$H = \begin{bmatrix} 1 & \beta & \ldots & \beta^{(n-1)} \\ 1 & \beta^2 & \ldots & \beta^{2(n-1)} \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \beta^{n-s} & \ldots & (\beta^{n-s})^{n-1} \end{bmatrix}.$$

That means that the generator polynomial has zeroes $\beta, \beta^2, \ldots, \beta^{n-s}$, so the generator polynomial of this Reed–Solomon code is

$$g(x) = (x-\beta)(x-\beta^2)\ldots(x-\beta^{n-s}). \qquad \square$$

5.3.2 BCH codes

In the special case where $q = 2^m$ we look at the binary words in the Reed–Solomon codes C_s, denoted $C_s(\text{sub})$; this is the so-called *subfield subcode*.

Theorem 5.3.2. *The code $C_s(\text{sub})$ is a linear cyclic code whose generator polynomial is the product of the different minimal polynomials of $\beta, \beta^2, \ldots, \beta^{n-s}$. The code has minimum distance at least $n-s+1$.*

Proof. The code is linear, since the sum of two codewords again is in the code, and cyclic, since the Reed–Solomon code is. The generator polynomial has among its zeroes $\beta, \beta^2, \ldots, \beta^{n-s}$ and the binary polynomial of lowest degree with these zeroes is exactly the product of the different minimal polynomials of $\beta, \beta^2, \ldots, \beta^{n-s}$. Since the codewords are still words from the Reed–Solomon code the minimum distance is at least $n-s+1$. $\square$

In this case we can omit all the even powers of β, since if γ is a zero of a binary polynomial so is γ^2. If we choose a basis for $\mathbb{F}_2^m$ as a vectorspace over $\mathbb{F}_2$ and and replace the powers of β in the parity check matrix for the Reed–Solomon code with binary m-columns (the coordinates), then we get a parity check matrix (with possibly linearly dependent rows) for the subfield subcode. From this follows that the dimension of $C_s(\text{sub})$ is at least $n - m\left(\frac{n-s}{2}\right)$. Usually this is a very weak bound; in specific cases where we have the generator polynomial $g(x)$ there is of course no problem, since then $k = n - \deg(g(x))$. The true minimum distance of these codes can be hard to find, however in many cases the bound actually gives the true value. With $n = 2^m - 1$ and $d_{\min} = 2t + 1$ we get $n - k \leq t \log(n)$. For high rates this is very close to the Hamming bound.

The codes obtained in this way are called (narrow sense) BCH codes after Bose, Chaudhuri, and Hocquenghem, who first considered them. One of the nice things is that you can guarantee (a lower bound on) the minimum distance of the code; this was exactly the original rationale for the construction.

Example 5.3.1. (Example 5.1.3 continued).
The binary $(21, 12)$ code with $g(x) = m_\beta(x)m_{\beta^3}(x) = (x^6 + x^4 + x^2 + x + 1)(x^3 + x^2 + 1) = x^9 + x^8 + x^7 + x^5 + x^4 + x + 1$, where $\beta = \alpha^3$ and α is a primitive element of $\mathbb{F}_{2^6}$, is a subfield subcode of the $(21, 17, 5)$ Reed–Solomon code, so it has minimum distance at least 5. Since $c(x) = (x^2 + x + 1)g(x) = x^{11} + x^9 + x^4 + x^3 + 1$ is a codeword, the minimum distance is 5. This is actually best possible.

The Peterson decoding algorithm can be used to decode these codes as well, as we are going to illustrate.

Example 5.3.2. Decoding of a $(15, 5, 7)$ BCH code.
This code corresponds to the case where $n = 15, s = 9$, and $q = 2^4$, so the generator polynomial has among its roots $\beta, \beta^2, \ldots, \beta^6$ and in this case $\beta = \alpha$, where α is a primitive element of $\mathbb{F}_{16}$. Since $m_\beta(x) = m_{\beta^2}(x) = m_{\beta^4}(x)$ and $m_{\beta^3}(x) = m_{\beta^6}(x)$, we get that the generator polynomial is $m_\beta(x)m_{\beta^3}(x)m_{\beta^5}(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$, where we have used the table to find the minimal polynomials. The dimension of the code is therefore $15 - 10 = 5$ and since the generator has weight 7 and we know that the minimum distance is at least 7, we have that $d_{\min} = 7$.

In the following we use the version of $\mathbb{F}_{16}$ that has a primitive element α which satisfies $\alpha^4 + \alpha + 1 = 0$. Actually, we already used this in the determination of the minimal polynomials above.

If we receive the word $r(x) = x^{11} + x^{10} + x^9 + x^8 + x^7 + x^2$, we first calculate the syndromes.

We get $S_1 = r(\alpha) = \alpha^{14}$ $S_2 = r(\alpha^2) = {S_1}^2 = \alpha^{13}$, $S_3 = r(\alpha^3) = \alpha^6$, $S_4 = r(\alpha^4) = {S_2}^2 = \alpha^{11}$, $S_5 = r(\alpha^5) = \alpha^5$ and $S_6 = r(\alpha^6) = {S_3}^2 = \alpha^{12}$.

We now solve the system

$$\begin{bmatrix} S_1 & S_2 & S_3 & S_4 \\ S_2 & S_3 & S_4 & S_5 \\ S_3 & S_4 & S_5 & S_6 \end{bmatrix} \begin{bmatrix} Q_{1,0} \\ Q_{1,1} \\ Q_{1,2} \\ Q_{1,3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and get

$$Q_{1,0} = \alpha^7, Q_{1,1} = \alpha^4, Q_{1,2} = \alpha^{14}, Q_{1,3} = 1.$$

Therefore $Q_1(x) = x^3 + \alpha^{14}x^2 + \alpha^4 x + \alpha^7$ that has zeroes α^0, α^{10} and α^{12}.

From this we get as codeword $c(x) = r(x) + x^{12} + x^{10} + 1 = x^{12} + x^{11} + x^9 + x^8 + x^7 + x^2 + 1$. It turns out that $c(x) = (x^2 + x + 1)g(x)$.

5.4 Cyclic codes from $\mathrm{PG}(2, \mathbb{F}_{2^m})$

In this section we present a class of cyclic codes that are easily decoded. The idea is to use a numbering of the points and lines in $\mathrm{PG}(2, \mathbb{F}_{2^m})$ such that the incidence matrix is cyclic. If we use this matrix as the parity check matrix for a code, this code becomes cyclic. We present an efficient decoder that corrects all errors up to half the minimum distance.

Recall from Subsection 2.5.2 that $\mathrm{PG}(2, q)$ with $q = 2^m$ has $n = q^2 + q + 1$ points and lines, that a line contains $q + 1$ points, and that two different lines have exactly one point in common. Consider the field $\mathbb{F}_{q^3}$ constructed by using an irreducible polynomial $f(x) \in \mathbb{F}_q[x]$ of degree 3 such that a primitive element $\alpha \in \mathbb{F}_{q^3}$ satisfies $f(\alpha) = 0$. By selecting a basis for $\mathbb{F}_{q^3}$ as a vector space over $\mathbb{F}_q$ every power of α corresponds to a triple (x, y, z) of elements in $\mathbb{F}_q$. Since $n = \frac{q^3-1}{q-1}$, we have $\alpha^n \in \mathbb{F}_q$, and therefore the powers α^i, $i = 0, 1, \ldots, n-1$, can be used as representatives of the different points of $\mathrm{PG}(2, q)$. In Problem 2.6.12 we introduced the map $\mathrm{Tr} : \mathbb{F}_{q^3} \to \mathbb{F}_q$ by $\mathrm{Tr}(x) = x + x^q + x^{q^2}$. Since Tr is a linear map, the elements α^i where $\mathrm{Tr}(\alpha^i) = 0$ constitute a line l in $\mathrm{PG}(2, q)$, and the other $n - 1$ lines are the zeroes of the polynomial $\mathrm{Tr}(\alpha^{-j}x)$, $j = 1, 2, \ldots, n - 1$, respectively. This implies that if we label the points of l by the powers $i_1, i_2, \ldots, i_{q+1}$ of α, then the lines are cyclic shifts of l. So $P_j \in l_i$ iff $\mathrm{Tr}(\alpha^{-i}\alpha^j) = 0$ Hence if we use this labeling and construct the incidence matrix M we get

$$m_{ij} = 1 - [\mathrm{Tr}(\alpha^{j-i})]^{q-1} = \begin{cases} 1, & \text{if } P_j \in l_i, \\ 0, & \text{otherwise.} \end{cases}$$

Here the last equality follows from the fact that $\mathrm{Tr}(x) \in \mathbb{F}_q$, so $[\mathrm{Tr}(x)]^{q-1} = 1$ if $\mathrm{Tr}(x) \neq 0$ and $= 0$ if $\mathrm{Tr}(x) = 0$.

Since two different lines have exactly one point in common, the $q + 1$ lines through a point cover the whole plane. Another way of expressing this is by noting that the $(q + 1)q$ differences $i_j - i_s \mod n$, $j \neq s \in \{1, 2, \dots, q + 1\}$ are the numbers $1, 2, \dots, n - 1$, each taken exactly once. To see this suppose $i_{j_1} - i_{s_1} = i_{j_2} - i_{s_2} \mod n$, where $j_1 \neq j_2$. Then the cyclic shift of l which sends the point α^{j_1} into the point α^{j_2} gives a new line which meet l in two points which is impossible.

Definition 5.4.1. *The code with M as a parity check matrix is denoted $C(m)$.*

Example 5.4.1. PG$(2, 4)$.
Let $\mathbb{F}_4 = \{0, 1, \beta, \beta^2\}$ with $\beta^2+\beta+1 = 0$ and construct $\mathbb{F}_{64}$ using the polynomial $f(x) = x^3 + x^2 + x + \beta$. Let α be a primitive element of $\mathbb{F}_{64}$ with $f(\alpha) = 0$ and use $1, \alpha, \alpha^2$ as a basis for $\mathbb{F}_{64}$ over $\mathbb{F}_4$. The zeroes of the polynomial $\mathrm{Tr}(x) = x + x^4 + x^{16}$ are then $\alpha^7, \alpha^9, \alpha^{14}, \alpha^{15}, \alpha^{18}$ and the incidence matrix is

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

If we put

$$\theta(x) = x^{i_1} + x^{i_2} + \cdots + x^{i_{q+1}}, \tag{5.1}$$

then it follows from the above that

$$\theta(x)\theta(x^{-1}) = 2^m + 1 + x + x^2 + \cdots + x^{n-1} \mod (x^n - 1)$$

so calculating mod 2 we have

$$\theta(x)\theta(x^{-1}) = 1 + x + x^2 + \cdots + x^{n-1} \mod (x^n - 1),$$

and therefore

$$(x-1)\theta(x)\theta(x^{-1}) = 0 \mod (x^n - 1).$$

Example 5.4.2. PG(2, 4) continued.
We get

$$h(x) = \text{g.c.d.}(x^{18} + x^{15} + x^{14} + x^9 + x^7, x^{21} - 1) = x^{11} + x^8 + x^7 + x^2 + 1$$

and $g(x) = x^{10} + x^6 + x^4 + x^2 + 1$, so we have a $(21, 11,)$ code with minimum distance at most 6 (since $x + 1$ divides $g(x)$ the minimum distance is even). We will show that the code corrects two errors, so the minimum distance is exactly 6. The parity check equations containing c_0 (from M) are

$$\begin{array}{ccccccccccccc}
c_0 & + & c_{10} & + & c_{12} & + & c_{17} & + & c_{18} & = & 0, \\
c_0 & + & c_3 & + & c_{13} & + & c_{15} & + & c_{20} & = & 0, \\
c_0 & + & c_1 & + & c_4 & + & c_{14} & + & c_{16} & = & 0, \\
c_0 & + & c_5 & + & c_6 & + & c_9 & + & c_{19} & = & 0, \\
c_0 & + & c_2 & + & c_7 & + & c_8 & + & c_{11} & = & 0.
\end{array}$$

If we receive $(r_0, r_1, \ldots, r_{20})$ with two errors and calculate the sums corresponding to the equations above, we get

$$\begin{array}{lll}
1,1,1,1,0 & \text{(in some order)} & \text{if} \quad r_0 \neq c_0, \\
1,1,0,0,0 & \text{or } 0,0,0,0,0 & \text{if} \quad r_0 = c_0,
\end{array}$$

so we can determine c_0 by the majority of these five sums. It is easy to see that this remains true if there are one or zero errors. Since the code is cyclic, we can determine the remaining bits of the transmitted codeword by cyclic shifts of the equations above using the same kind of majority voting.
We note that in this case $g(x) = (x+1)(x^6 + x^4 + x^2 + x + 1)(x^3 + x^2 + 1) = (x+1)m_\beta(x)m_{\beta^3}(x)$, where $\beta = \alpha^3$ and α is a primitive element of $\mathbb{F}_{64}$, so $g(x)$ has among its zeroes $\beta^0, \beta, \beta^2, \beta^3, \beta^4$. Hence, so also from this we have that the minimum distance is at least 6.

Theorem 5.4.1. *The code $C(m)$ has minimum distance $2^m + 2$.*

Proof. The decoding method illustrated in the example can be used in the general case where 2^{m-1} errors are corrected and therefore the codes have minimum distance at least $2^m + 2$ (since it is even). On the other hand, if we consider the set of points

$$S = \{(t : 1 : t^2) | t \in \mathbb{F}_q\} \cup \{(0 : 0 : 1)\} \cup \{(1 : 0 : 0)\},$$

then it can be seen (see Problem 5.5.17) that S has two or zero points in common with every line $ax + by + cz = 0$, so the incidence vector of S is a codeword, since it is orthogonal to all rows of M. Therefore the minimum distance of the code is exactly $q + 2 = 2^m + 2$. □

Theorem 5.4.2. *The code $C(m)$ has dimension $n - (3^m + 1)$.*

This together with Theorem 5.4.1 implies that we have a class of cyclic codes with parameters

$$(2^{2m} + 2^m + 1, 2^{2m} + 2^m - 3^m, 2^m + 2)$$

that are easily decoded up to half the minimum distance.

Proof. As noted before, the dimension k equals the degree of the polynomial $h(x) = \text{g.c.d.}(\theta(x), x^n - 1)$, so if we put $\beta = \alpha^{q-1}$, where α as before is a primitive element of $\mathbb{F}_{q^3}$, then we get

$$k = |u \in \{0, 1, \dots, n-1\} : \theta(\beta^u) = 0|$$

From (5.1) we have $\theta(x) = \sum_{j=0}^{n-1} c_j x^j$, where $c_j = 1 - [\text{Tr}(\alpha^j)]^{q-1}$, so if we put

$$d_u = \sum_{j=0}^{n-1} c_j \beta^{uj} = \sum_{j=0}^{n-1} c_j \alpha^{ju(q-1)},$$

we have that $k = |u \in \{0, 1, \dots, n-1\} : d_u = 0|$.

In particular, if $u = 0$ we get $d_0 = \sum_{j=0}^{n-1} c_j = q + 1 \equiv 1 \mod 2$. In the following we only consider the cases where $0 < u \le n - 1$. Inserting the value of c_j we get

$$d_u = \sum_{j=0}^{n-1} (1 - [\text{Tr}(\alpha^j)]^{q-1})\alpha^{ju(q-1)}.$$

In the sense of Appendix A, the vectors c and d are a transform pair.

$$\begin{aligned} d_u &= \sum_{j=0}^{n-1} \alpha^{ju(q-1)} - \sum_{j=0}^{n-1} [\text{Tr}(\alpha^j]^{q-1} \alpha^{ju(q-1)} \\ &= -\sum_{j=0}^{n-1} [(\alpha^j + \alpha^{jq} + \alpha^{jq^2})\alpha^{ju}]^{q-1} \end{aligned}$$

since the first sum is 0.

If we put here $y = \alpha^j$, we get

$$\begin{aligned}\hat{c_j} &= \left[y + y^q + y^{q^2}\right]^{q-1} = \left[y + y^q + y^{q^2}\right]^{2^m-1} \\ &= \left[y + y^q + y^{q^2}\right]^{1+2+\cdots+2^{m-1}} = \prod_{l=0}^{m-1}\left(y^{2^l} + y^{2^{l+m}} + y^{2^{j+2m}}\right),\end{aligned}$$

where $\hat{c_j}$ is the complement of c_j. If we write a $0 < b < q^3 - 1$ as

$$b = \sum_{i=0}^{m-1} a_i 2^i + 2^m \sum_{i=0}^{m-1} b_i 2^i + 2^{2m} \sum_{i=0}^{m-1} c_i 2^i$$

with $a_i, b_i, c_i \in \{0, 1\}$, and define

$$S = \{b | a_i + b_i + c_i = 0, i = 0, 1, \ldots, m-1\},$$

then we have

$$\hat{c_j} = \prod_{l=0}^{m-1}\left(y^{2^l} + y^{2^{l+m}} + y^{2^{j+2m}}\right) = \sum_{b \in S} y^b.$$

The number of elements in S is clearly 3^m, and this is the number of nonzero coefficients in $\hat{c}$. Since the coefficients are the transforms of the values of a polynomial, we get that u, with $1 \leq u \leq n-1$ such that $d_u \neq 0$ is exactly the number of elements in S. Since $d_0 = 1$, the dimension of the code $C(m)$ is therefore $n - (3^m + 1)$. □

5.5 Problems

Problem 5.5.1. $g(x) = x^6 + x^3 + 1$ divides $x^9 - 1$ in $\mathbb{F}_2[x]$.

(1) Show this.

$g(x)$ can be used as generator polynomial of a binary cyclic $(9, k)$ code C, i.e.,

$$C = \{i(x)g(x) | i(x) \in \mathbb{F}_2[x], \deg(i(x)) < 3\}.$$

(2) What is the dimension of C?

(3) Determine a generator matrix of C.

(4) Is $x^8 + x^6 + x^5 + x^3 + x^2 + 1$ a codeword of C?

(5) What can you say about the minimum distance of C?

Problem 5.5.2. The polynomial $x^{15} - 1$ can be factored into irreducible polynomials over $\mathbb{F}_2$ as

$$x^{15} - 1 = (x+1)(x^2+x+1)(x^4+x+1)(x^4+x^3+1)(x^4+x^3+x^2+x+1).$$

Let C be the binary cyclic code of length 15 that has generator polynomial

$$g(x) = (x+1)(x^4+x+1).$$

(1) What is the dimension of C?

(2) Is $x^{14} + x^{12} + x^8 + x^4 + x + 1$ a codeword in C?

(3) Determine all cyclic binary $(15, 8)$ codes.

(4) How many cyclic binary codes of length 15 are there?

Problem 5.5.3. Let C be the cyclic code of length 15 that has generator polynomial $g(x) = x^4 + x + 1$.

(1) Determine a parity check matrix of C.

(2) Find the minimum distance of C.

(3) What is C?

(4) What is the dimension of $C^\perp$?

Problem 5.5.4. Let $g(x)$ be the generator polynomial of a cyclic code of length n over $\mathbb{F}_2$.
Show that $g(1) = 0$ if and only if all codewords have even weight.

Problem 5.5.5. Let C be a binary cyclic code of odd length n.
Show that if C contains a word of odd weight then it contains the all 1s word, i.e., $(1, 1, 1, 1 \ldots, 1)$

Problem 5.5.6. Let C be the cyclic $(21, 12)$ code over $\mathbb{F}_2$ that has generator polynomial $g(x) = m_\beta(x)m_{\beta^3}(x)$.

(1) What can you say about the minimum distance of C?

(2) Determine the generator polynomial of $C^\perp$.

(3) What can you say about the minimum distance of $C^\perp$?

Problem 5.5.7. Let C be the cyclic code of length 63 over $\mathbb{F}_2$ that has generator polynomial $(x+1)(x^6+x+1)(x^6+x^5+1)$.
What can you say about the minimum distance of C?

Problem 5.5.8.

(1) Determine a generator polynomial for a cyclic code of length 31 over $\mathbb{F}_2$ that can correct 3 errors.

(2) What is the dimension of the code you found?

(3) Can you do better?

Problem 5.5.9. Let C be a binary (n, k) code. Recall that an encoding rule is called *systematic* if the information $(u_{k-1}, \dots, u_1, u_0)$ is encoded into a codeword $(c_{n-1}, \dots, c_1, c_0)$ that contains $(u_{k-1}, \dots, u_1, u_0)$ in k fixed positions. A cyclic (n, k) code consists of all the words of the form $u(x)g(x)$ where $u(x)$ has degree $< k$ and $g(x)$ is the generator polynomial of the code.

(1) Show that the encoding rule $u(x) \to u(x)g(x)$ is not systematic by looking at the $(7, 4)$ code with generator polynomial $x^3 + x + 1$.

We now use the rule $u(x) \to x^{n-k}u(x) - (x^{n-k}u(x) \mod g(x))$.

(2) Show that $x^{n-k}u(x) - (x^{n-k}u(x) \mod g(x))$ is a codeword in the cyclic code with generator $g(x)$.

(3) Prove that this is a systematic encoder.

(4) Use the same code as above to encode 1011 using the systematic encoder.

Problem 5.5.10. Let C be a cyclic (n, k) code with generator polynomial $g(x)$. Let $h(x) = \frac{x^n - 1}{g(x)}$ and let $a(x)$ be a polynomial of degree $< k$ and $\gcd(h(x), a(x)) = 1$.
Let $C' = \{c(x) = u(x)a(x)g(x) \mod (x^n - 1) | \deg(u(x)) < k\}$.
Show that $C' = C$.

Problem 5.5.11. Here we treat the famous binary Golay code (1949). You will prove that the code is perfect. It was later proven that this and the Hamming codes are the only linear binary perfect codes with $k > 1$. We have seen that $x^{23} - 1 = (x+1)m_\beta(x)m_{\beta^5}(x)$, where $m_\beta(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$. Let C be the $(23, 12)$ code that has generator polynomial $m_\beta(x)$.

(1) Show that C has minimum distance at least 5.

Let C_{ext} be the $(24, 12)$ code obtained by adding an overall parity check.

(2) Show that the weights of all the codewords in C_{ext} are divisible by 4.

(3) Show that C has minimum distance 7.

(4) Show that C is a perfect code.

Problem 5.5.12. Let C be a cyclic (n, k) code whose generator polynomial $g(x)$ is the product of the different minimal polynomials of the elements $\beta, \beta^2, \beta^3, \dots, \beta^{2t}$, where $\beta \in \mathbb{F}_{2^m}$ has order n.
We then know that $d_{\min}(C) \geq 2t + 1$.
Consider the case where $n = a(2t + 1)$.

(1) Show that $\beta, \beta^2, \beta^3, \dots, \beta^{2t}$ are not zeroes of $x^a - 1$.
(2) Show that $m_{\beta^s}(x), s = 1, 2, \dots, 2t$, does not divide $x^a - 1$.
(3) Show that $\gcd(g(x), x^a - 1) = 1$.
(4) Show that $p(x) = \frac{x^n - 1}{x^a - 1}$ is a binary polynomial.
(5) Show that $g(x) | p(x)$.
(6) What is the weight of $p(x)$?
(7) Show that the minimum distance of C is $2t + 1$.

Problem 5.5.13. We have

$$\begin{aligned} x^{31} - 1 =& (x + 1)(x^5 + x^2 + 1)(x^5 + x^3 + 1)(x^5 + x^4 + x^3 + x^2 + 1) \\ & \cdot (x^5 + x^3 + x^2 + x + 1)(x^5 + x^4 + x^2 + x + 1) \\ & \cdot (x^5 + x^4 + x^3 + x + 1). \end{aligned}$$

(1) How many binary cyclic codes of length 31 are there?
(2) Is there one of these with dimension 10?

Let C be the code with generator polynomial $g(x) = (x + 1)(x^5 + x^2 + 1)$.

(3) Is $x^7 + x^5 + x^4 + x^2 + 1$ a codeword of C?
(4) Is $x^7 + x^5 + x^4 + 1$ a codeword of C?
(5) What can you say about the minimum distance of C?

Problem 5.5.14. Determine the generator polynomials and the dimension of the cyclic codes of length 63 that corrects 4, 5, 9, 10, 11 and 15 errors.

Problem 5.5.15. Show that there exists cyclic $(n, k, d_{\min})$ binary codes with the following parameters:
$(21, 12, \geq 5)$, $(73, 45, \geq 10)$, $(127, 71, \geq 19)$.

Problem 5.5.16. When we construct binary cyclic codes of length n we use minimal polynomials for powers of an element β of order n. It is natural to ask if the choice of this element β is important. This problem treats this question in a special case.
Let n be an odd number and j a number such that $\gcd(n, j) = 1$.

(1) Show that the mapping $\pi : \{0, 1, \dots, n - 1\} \to \{0, 1, \dots, n - 1\}$ defined by $\pi(x) = xj \mod n$ is a permutation, i.e., a bijective mapping.

Let C_1 be the cyclic code that has $m_\beta(x)$ as generator polynomial, and let C_2 be the code that has $m_{\beta^j}(x)$ as generator polynomial.

(2) Show that $(c_0, c_1, \dots, c_{n-1}) \in C_1 \iff (c_{\pi(0)}, c_{\pi(1)}, \dots, c_{\pi(n-1)}) \in C_2$.
(3) What does this mean for the two codes?

Problem 5.5.17. Consider in $\mathrm{PG}(2, q)$, where $q = 2^m$, the set of points

$$S = \{(t : 1 : t^2) | t \in \mathbb{F}_q\} \cup \{(0 : 0 : 1)\} \cup \{(1 : 0 : 0)\}.$$

Prove that S has two or zero points in common with every line $ax + by + cz = 0$ of $\mathrm{PG}(2, q)$.

Chapter 6

Frames

So far we have considered transmission of independent information symbols and encoding using block codes. However, in order to assure correct handling and interpretation, data are usually organized in files with a certain structure. In a communication system the receiver also needs structure in order to decode and further process the received sequence correctly. A general discussion of the architecture of communication systems is outside the scope of this text. However, we can capture some essential aspects of the discussion by assuming that the information is transmitted in frames (also called packets in some systems) of a fixed length.

By setting error correction in this context we can discuss the performance of an error-correcting code in terms of the quality of the frames. We define some commonly used parameters for frame quality and relate them to the error probability of the codes. The concept of frames will also turn useful for the discussion of other code structures in the following chapters.

6.1 Definitions of frames and their efficiency

Definition 6.1.1. *A* data file *is an entity of data that can be independently stored, communicated, and interpreted. It consists of a* header*, a* data field*, and a* parity field.

We assume that the data field consists of a fixed number of *source symbols*, although files may have a variable-length identified in the header.

The parity field is discussed in detail in Section 6.2. It usually serves to detect errors in the file, and no correction takes place. If a file of n_{d} data bits is corrupted, we may assume that the probability of having m parity check bits satisfied is of the order 2^{-m} independent of n_{d}.

The header serves to indicate the beginning of the file, and it may contain such information as the file title and date. Some of the information may be related to the length or positions within the file. Thus this information has either a length that is independent of n_{d} or is also of the order $\log(n_{\mathrm{d}})$. If the beginning of the file is marked by a fixed pattern, it is important that this pattern occurs infrequently in the data. Thus the length of the pattern has to be of the order $\log(n_{\mathrm{d}})$. If the length of the header is h_{d}, we may characterize the efficiency of the file by the following definition.

Definition 6.1.2. *The* file efficiency *is given by the ratio of data symbols to the total length of the file.*

Thus we may express the efficiency as

$$\eta_{\mathrm{d}} = n_{\mathrm{d}}/(n_{\mathrm{d}} + h_{\mathrm{d}} + m). \tag{6.1}$$

Lemma 6.1.1. *The file efficiency can be made arbitrarily close to* 1 *if* n_{d} *is chosen large enough.*

When information is transmitted over a channel, it is encoded using a suitable error-correcting code.

Definition 6.1.3. *A* frame *is an entity of encoded information transmitted over the channel. It consists of a* header (*possibly separately encoded*) *and a* body.

The frame consists of a fixed number of *channel symbols*, which in general belong to an alphabet that can be different from the information symbols. We note that in general files and frames are different, and in some systems they are handled independently, they have separate headers, and they may have different lengths and boundaries.

The headers identify the beginnings of the frames, a function which is referred to as frame synchronization. Usually the header includes a fixed synchronization pattern, which is not encoded, and it must be identified in the presence of some errors. However, the length is still of the order $\log(N_{\mathrm{f}})$, where N_{f} is the length of the body. The frame header may contain information relevant to the interpretation of the contents, such as a frame number, some identification of the contents, and possibly pointers for unpacking. In addition, the frame header can contain information about the routing, including the receiver address. In the present context we shall not discuss the security of the information, i.e., the cryptographic coding and signatures. The data are encoded using a channel code of rate R. The header information may be encoded by the same code, or it may be treated separately to simplify the processing. We let H_{f} indicate the total number of channel symbols used for transmitting the headers.

For simplicity the following definition refers to channels with binary input symbols:

Definition 6.1.4. *The* transmission efficiency, η_{f}, *is the ratio of data symbols represented in the body to the total capacity of the transmitted channel symbols.*

It follows from the definition that we have

$$\eta_{\mathrm{f}} = \frac{N_{\mathrm{f}} R}{(N_{\mathrm{f}} + H_{\mathrm{f}})C}, \tag{6.2}$$

where C is the capacity of the channel. Thus $1 - \eta_{\mathrm{f}}$ combines the loss due to the length of the header and the loss due to the rate of the code.

Lemma 6.1.2. *The transmission efficiency can be made arbitrarily close to* 1 *if* N_f *is chosen large enough.*

Proof. The header is short relative to the frame. The rate can be chosen close to the channel capacity when the code is long enough. □

It follows from the lemmas above that even with the additional symbols needed to get a realistic communication structure we have

Theorem 6.1.1. *The average number of channel symbols used to communicate* n_d *data bits over a memoryless channel of capacity* C *can be made arbitrarily close to* n_d/C *when the data file and the transmission frame are sufficiently long.*

6.2 Frame quality

6.2.1 Measures of quality

In a communication network, the transmission of a frame can go wrong in a number of ways. The address may be corrupted, or the frame may be lost or delayed. It is the purpose of the error correction to ensure that the frame is delivered to the user without errors, or at least that transmission errors are detected. The performance can be characterized by two numbers:

Definition 6.2.1. *The* probability of undetected error*,* P_ue*, is the probability that a frame is passed on to the user as error-free while in fact parts of the data are in error.*

Definition 6.2.2. *The* probability of frame error*,* P_fe*, is the probability that the receiver detects errors in the data, but is not able to correct them, or at least is not able to do so with sufficient reliability.*

These parameters are relevant in applications where the receiver may choose to discard unreliable frames (the user may be able to have them retransmitted). The probability of frame error should be small enough to allow normal communication to take place, but it does not have to be extremely small, 10^{-5} may be a typical target value. On the other hand, the probability of undetected error should be much smaller ($< 10^{-15}$).

If the receiver does not have the option of discarding degraded frames (either because a delay cannot be tolerated, or it is not practical to flag bad frames), the data are passed on with as few errors as possible. The performance is usually characterized by the bit error rate:

Definition 6.2.3. *The* bit error rate*, P_{bit}, is the average number of errors in the decoded data divided by the length of the decoded body.*

We may calculate (a close approximation to) the bit error rate by assuming that the number of bit errors is $\frac{d}{2}$ for a block that is not decoded and d for a wrong word. These bits are equally likely to hit the information symbols and the parity symbols in the code, so the number should be multiplied by the rate R. Clearly it is an advantage in this situation that the encoding is systematic, since decoding might otherwise change more information bits. We may assume that the frames are independent, but clearly the distribution of errors within a frame do not follow a simple probability distribution. In applications like speech, it is acceptable to have a moderate value of P_{bit}, but in applications where the data are compressed or otherwise sensitive, the requirement may be $P_{\text{bit}} < 10^{-15}$.

6.2.2 Parity checks on frames

If we can assume that there are few errors in a decoded frame, it may be sufficient to verify the integrity of received data by a system of parity checks called a frame (or file) check sequence. Often (shortened) cyclic versions of Hamming codes or subcodes of Hamming codes are used for this purpose, and we shall discuss only this case.

Lemma 6.2.1. *A cyclic binary code defined by the generator polynomial*

$$g(x) = p(x)(x+1),$$

where $p(x)$ is a primitive polynomial of degree m, has parameters

$$(n,k,d) = (2^m - 1, 2^m - m - 2, 4).$$

The maximum number of information symbols is $2^m - m - 2$, but the code can be shortened by leaving out leading information bits. The $m+1$ parity bits are known as the *frame check sequence* or the CRC (cyclic redundancy check). Usually the frame check sequence is not used for error correction, but errors in the transmission are detected. If more than two errors occur, we may estimate the probability that the frame is accepted as the probability that a random sequence produces the zero frame check sequence, i.e.,

$$P_{\text{ue}} \approx P[t \geq 2] \cdot 2^{-m-1} \tag{6.3}$$

Thus the length of the sequence is chosen to give a sufficient frame length and an acceptable reliability.

Example 6.2.1. Standard CRCs.

The following polynomials are used for standard CRCs:

$$\begin{gathered}x^{16}+x^{12}+x^5+1\\ =(x+1)\left(x^{15}+x^{14}+x^{13}+x^{12}+x^4+x^3+x^2+x+1\right)\ \text{CRC-ITU-T;}\\ x^{16}+x^{15}+x^2+1=(x+1)\left(x^{15}+x+1\right)\ \text{CRC-ANSI;}\\ x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}\\ +x^8+x^7+x^5+x^4+x^2+x+1\ \text{CRC-32.}\end{gathered}$$

The first two polynomials are products of $x+1$ and irreducible polynomials of degree 15 and period 32767. Thus they generate distance-4 subcodes of Hamming codes, and all combinations of at most three errors in at most 32751 data bits are detected. The last polynomial is irreducible.

6.2.3 Header protection codes

As indicated earlier, the synchronization pattern in the header is not encoded (or decoded), but the rest of the header may contain information which will cause the frame to be lost if it is corrupted (such as an address). Since it is practical to let the receiver read the header without decoding the body, the header may be protected by a short (n,k,d) code. To assure a low probability of false decoding, the decoder may correct $t' < (d-1)/2$ errors, leaving the remaining parity symbols for error detection. As a result, the rate of the code is reduced, but this has little effect on the efficiency.

6.3 Short block codes in frames

In principle we can get a good performance by using long frames and applying a single block code to protect an entire frame. However, the decoding of such a code may not be practical. In this section we consider the possibility of breaking the frame into shorter blocks. In Chapters 7 and 8 we discuss other approaches.

A frame of length N_f may be broken into N_f/n codewords from a (n,k) code. Here we assume that k is chosen to divide the number of information symbols in the frame.

In Chapter 3 we presented bounds on the error probability for block codes. Assume that based on a bound or on simulations of the particular code, we know the probability of decoding error, P_err.

Lemma 6.3.1. *If the frame consists of $\frac{N_f}{n}$ blocks, and the noise is independent, the probability of a correct frame is*

$$P_{ue} = 1 - \left(1 - P_{err}\right)^{\frac{N_f}{n}}. \tag{6.4}$$

Thus we can get a very small probability of undetected errors only if the error rate on the channel is very small, and the system is not very efficient. In order to improve the performance, we may restrict the number of errors corrected in each block. However, with a probability of decoding failure P_{fail} we similarly get a bigger probability of frame error.

Lemma 6.3.2. *If the frame consists of $\frac{N_f}{n}$ blocks, and the noise is independent, the probability of frame error is*

$$P_{fe} = 1 - \left(1 - P_{fail}\right)^{\frac{N_f}{n}}. \tag{6.5}$$

For the situation of main interest here, it is more efficient to combine error correction with a frame check sequence, as discussed in the previous section. By using 16 or 32 bits for this purpose it is possible to make the probability of undetected error very small, and we can still use the full power of the short block code for error correction.

Example 6.3.1. Assume that the frames are transmitted over a BSC with error probability $p = 0.01$, and that the body consists of 100 codewords of a $(128, 85, 14)$ block code. Thus the code corrects $t = 6$ errors, but in most cases it is possible to correct 8 errors, since the number of such error patterns is smaller than the number of syndromes. Assuming bounded distance decoding with six errors, we find the probability of decoding failure from the binomial distribution and (3.10)

$$P_{fail} \simeq P(7) + P(8) = 0.000268 + 0.000040 = 0.000308.$$

Thus the average number of blocks that are not decoded is 0.03 in each frame. This is also the probability of frame error. Since each event causes more than 7 errors to be left in the block, the average bit error probability would be

$$P_{bit} \simeq \frac{0.21}{N} = 1.6 \cdot 10^{-5}.$$

The probability of undetected errors can be found with a sufficiently good approximation from the probability that 8 errors occur within the support of a codeword of weight 14. The number of such codewords can be approximated by scaling the binomial distribution on 127 positions and adding the terms for weights 13 and 14 to get even weights. This gives a little less than 400,000 codewords.

$$P_{ue} \simeq 400{,}000 \binom{14}{8} 0.01^8 = 10^{-7}.$$

None of these numbers are satisfactory. We can improve P_{fe} and P_{bit} by about an order of magnitude by correcting 7 errors (this only gives a small number of decoding errors in words of weight 14, and the correction of an extra error is not very difficult). By adding a frame check sequence, we can make P_{ue} much smaller. If necessary, we can use a $(128, 78, 16)$ block code to protect the header and correct 6 errors to achieve a higher reliability of the header information.
The example shows that with a short block code supplemented with a CRC check we can get an acceptable performance, but the rate, $\frac{85}{128} = 0.66$, is not close to the capacity 0.92.

The decoder must somehow know how to segment the received sequence into blocks in the correct way. In the setting we consider here, the obvious solution to the problem of block synchronization is to partition the frame starting from the sync pattern. However, block codes, and cyclic codes in particular, are quite vulnerable to shifts of the symbol sequence within the frame. In addition, all linear codes have the undesirable feature that the all 0s word appears. For this reason, a fixed coset of such a code is sometimes used rather than the original code. One such method is to add a pseudorandom sequence at the transmitter and the receiver. If the sequence is not a codeword, it may serve to shift the code to a suitable coset.

6.3.1 Reed Solomon codes and long BCH codes

If the received data as delivered to the user are not sufficiently reliable, a Reed–Solomon code or a binary BCH code of high rate can provide a much more powerful alternative for the parity field in the data frame. Such a code allows several errors to be corrected, often 8, but at the same time the probability of undetected error is low. As indicated earlier, the probability of decoding (to a wrong codeword) when more than t errors are present is about $1/t!$ (3.13).

It is also possible to include this extra decoding stage (sometimes referred to as an outer code) in the decoding of the frame. In this case the channel code becomes a product code, as discussed in Chapter 8.

Example 6.3.2. QR (quick response) codes.
Many of the concepts in this chapter can be illustrated by considering two-dimensional versions of bar codes (A simple bar code is used on the back cover of this volume for the ISBN number). The QR codes are a square arrays of black or white squares used to convey short texts, web addresses, ticket information, etc. The QR code on the back cover links to a web page containing supplementary material for this book. It may be scanned by the camera on a smart phone or tablet. QR codes come in many versions, the one used here (version 4) uses 33 by 33 bits. Of these 1089 bits 252 have fixed values to allow proper alignment of the array and synchronization, most of these bits are arranged in eye patterns at the corners. About 800 bits represent characters of a (shortened) RS code with $m = 8$. There

is a choice of several rates for this code. The header consists of 16 bits, including a header protection code (the header is found twice near the eye patterns). These bits indicated the particular code version and the binary pattern that is added to the body to avoid too many adjacent bits of the same value. Each RC character occupies a small concentrated area, since it is expected that most errors are caused by local damage to the printed surface.

6.4 Problems

Problem 6.4.1. Consider frames consisting of 128 words of 32 bits. The first word is the frame header. The remaining words are used to transmit 16 data bits protected by the $(32, 16, 8)$ code discussed in earlier problems. The last word is a parity check on the frame excluding the header (the $\bmod 2$ sum of the data words).

(1) The frame is transmitted over a BSC with bit error probability $p = 0.01$. Find the transmission efficiency.

(2) Prove that the last word is also a word in the $(32, 16)$ code.

(3) With the given bit error probability, and assuming that the decoder corrects up to 3 errors, what is the probability that a word

   a) Is corrected?

   b) Is not decoded?

   c) Is decoded to a codeword different from the one sent?

(4) What are the probabilities of frame error and undetected error?

(5) Find these probabilities if the frame check is used to correct a single non-decoded word.

(6) Extend the frame to 512 words and use the last 4 words as parity symbols in a RS code. Find the probability of frame error and undetected error if the RS code is used to correct 2 errors or 4 non-decoded words.

Problem 6.4.2. Frames are transmitted on a BSC with $p = \frac{1}{128}$. The length of the frame is 4096 bits excluding the unencoded header of the frame.

(1) What is the capacity of the channel?

(2) Is it possible, in principle, to communicate reliably over the channel using codes of rate

   a) $\frac{14}{15}$?

   b) $\frac{15}{16}$?

(3) Assuming a code rate of $R = \frac{3}{4}$,

   a) How many information bits are transmitted?
   b) What is the transmission efficiency?

(4) Suppose a block code of length 256 is used.

   a) What is the average number of bit errors in a block?
   b) How many errors, t, can be corrected (assuming that this number is found from the Hamming bound)?

(5) If t errors cause a decoding failure and $t + 1$ errors a decoding error, what are the probabilities of these events? The probability distribution for the number of errors is the binomial distribution, but it may be approximated by the Poisson distribution.

(6) If only this code is used, what are the probabilities of frame error and undetected error?

(7) A 16 bit CRC is used to reduce the probability of undetected error.

   a) Is the sequence long enough?
   b) How much is the transmission efficiency reduced ?

Chapter 7

Maximum Likelihood Decoding and Convolutional Codes

The most important limitation on algebraic decoding algorithms is that only $t < d/2$ errors can be corrected, while in most cases error patterns of significantly higher weight could in principle be corrected. This chapter deals with a method for organizing parity checks that is particularly suitable for Maximum Likelihood (ML) decoding, although it is still practical only for small values of t.

Even though each parity check is limited to include a small set of encoded symbols, a long frame can be encoded in ways that do not divide the frame into independent sub-blocks. Here we present the definition and basic properties of a class of encoding methods referred to as *convolutional codes*. The characteristic of these codes is that the block length is variable, often it covers the entire frame, while the code is defined by local parity checks. Even though convolutional codes have been studied for several decades, and there are several approaches to the theory, no analysis of their structure has had much impact on the applications. For this reason this chapter focuses on the aspects that are needed to understand convolutional codes as they are currently used. The final section contains material that relates the properties of convolutional and block codes.

7.1 Definitions of convolutional codes

Throughout this chapter N/R denotes the length of the encoded frame and R the rate of the convolutional code. All codes considered in this chapter are binary. The case $R = 1/2$ is by far the most important, and this case is considered in the examples. The extension to $R = 1/n$ is usually made in the text or it is left as an exercise. In Section 7.5 we discuss how codes of other rates are constructed from these special cases.

Whereas in a block code all encoded symbols are functions of the same k information symbols, each sub-block of n encoded symbols in a convolutional code is a function of the current and M most recent information symbols (and in some cases the most recent encoded symbols). Thus the encoder is a *finite state machine*. We describe those properties of finite state machines that are needed in this chapter.

Definition 7.1.1. *A* finite state machine *is characterized by a finite input alphabet, a finite output alphabet, a finite set of states, $\Sigma = \{\sigma_0, \sigma_1, \ldots\}$, and two mappings: the next state mapping gives the state in the following instance of time as a function of the present state and the input symbol, $\sigma(j+1) = NS(\sigma(j), u_j)$, the output mapping similarly defines the current output symbol, $y_j = \phi(\sigma(j), u_j)$.*

Finite state machines are closely related to finite state sequences (or languages). The set of all sequences from the input (or output) alphabet is a trivial finite state sequence (with a single state). Other finite state sequences can be generated as the set of outputs from a finite state machine when this set is applied as the input.

Lemma 7.1.1. *The set of outputs produced by a finite state machine when the input set is a finite state sequence is again a finite state sequence.*

Proof. Combine the given machine with a machine that produces the input set from a trivial set. □

This chapter describes binary convolutional codes, and decoding for the BSC. Thus the alphabets are binary, and the mappings are always linear functions in $\mathbb{F}_2$. As for block codes, this gives us the advantage of allowing the error correcting properties of the codes to be studied as weights of encoded sequences rather than the distances between sequences. However, the Maximum Likelihood decoding algorithm, which is the focus of this chapter, does not depend on the linearity of the code. Thus it is often applied to channels with other output alphabets (as described in Appendix B), and to nonlinear codes designed for non-binary channels.

Let the information be given as a vector $u = [u_j]$ of length N.

Definition 7.1.2. *A rate $1/n$ convolutional encoding of the information vector, u, by n binary* generating vectors, *g_r, of length $M+1$, maps u to n encoded N-vectors*

$$y_{jr} = \sum_{i=0}^{M} g_{ri} u_{j-i}, \tag{7.1}$$

where the sum is modulo 2, $N > M$, and the indices are interpreted modulo N. The set of encoded sequences is referred to as the convolutional code. We refer to M as the memory of the code. The n sequences y_r are interleaved to a single binary sequence y of length nN for transmission over the channel.

We refer to the composition in (7.1) as a cyclic convolution of u and g, and the term convolutional code refers to this form of the encoding. It is simple to perform the encoding as a non-cyclic convolution, and we can make this modification by inserting M 0s at the end of the input. The reduction of the information efficiency is small when N is much larger than M.

The encoding of a convolutional code may be described by a finite state machine (the encoder). The input is an information symbol, u_j. It is usually most convenient to identify the state by the previous M input symbols, $[u_{j-1}, u_{j-2}, \dots, u_{j-M}]$, and we shall refer to the state index as the binary expansion of the integer $\sum u_{j-i} 2^i$ with the most recent symbol as the least significant bit.

The next state is obtained by shifting the state variables to the right and adding the new input as the first state variable. *Thus a transition from state σ_i leads to either state σ_{2i} or σ_{2i+1} modulo 2^M.* An important consequence of this labeling of the states is that all encoders with a given memory have the same states and next state mapping.

The output, y_j, is a string of n channel symbols.

Lemma 7.1.2. *A finite state encoder for a code with memory M has 2^M states.*

The output function is given as a $2^M \times 2^M$ matrix Φ in which the row and column indices $0 \leq i < 2^M$ refer to a numbering of the states given above. In particular, state 0 corresponds to an all 0 vector of state variables. Each entry in Φ, $\phi_{i'i}$, is the binary vectors associated with the transition from state i to state i'.

Example 7.1.1. A rate $\frac{1}{2}$ convolutional code A code with $R = \frac{1}{2}$ and $m = 2$ is defined by $g_0 = [1, 1, 1], g_1 = [1, 0, 1]$. A encoded sequence consists of blocks, each of two symbols (often separated by periods). Similarly we may write the generating sequence as $g = (11.10.11)$. For $N = 8$ the input $u = (11001000)$ convolved with g_0, g_1 gives $y = (11.01.01.11.11.10.11.00)$. The encoder has four states. With the chosen numbering of the states we may write Φ as

$$\Phi = \begin{bmatrix} 00 & - & 11 & - \\ 11 & - & 00 & - \\ - & 01 & - & 10 \\ - & 10 & - & 01 \end{bmatrix}.$$

If the encoder is initially in state 00, the sequence of states is

$$(01, 11, 10, 00, 01, 10, 00, 00).$$

We do not discuss the details of implementations as electronic circuits (which are highly dependent of the choice of technology), but it is of historical significance that an encoder for a convolutional code can be implemented in a particularly simple device. Thus in the early applications of coding in space telemetry, convolutional codes were a very effective choice. In an encoder for a block code the segmenting of the blocks adds to the complexity.

In a way that resembles the notation of cyclic block codes, we can write g_r as a polynomial, $g_r(x)$, and interpret the convolution as a product of the information polynomial and the generating polynomial. Note that a rate $1/n$ convolutional code is defined by n generating polynomials. We assume that the polynomials are always chosen in such a way that they have no non-trivial common factor (the presence of such a factor gives an undesirable encoder referred to as 'catastrophic'). With this assumption we have

Lemma 7.1.3. *A given rate $1/n$ linear convolutional code has a unique set of encoding polynomials.*

If one of the polynomials is chosen simply as $g_0 = 1$, we have a systematic code. However, this approach makes poor use of the available states. It is easily seen that we can make the first encoded sequence equal a given string of information symbols by replacing the input to the encoder by a suitable binary string. In terms of the generating polynomials we can replace a pair $[g_0(x), g_1(x)]$, by a constant and a rational function $[1, g_1(x)/g_0(x)]$ if $g_{00} = 1$ (this is always the case, or we would have shifted the coefficients to make the constant term nonzero). Note that this approach is quite similar to our choice of a systematic encoder for a cyclic code in Chapter 5.

The encoding of a convolutional code may be expressed in matrix form. The matrix that corresponds to the encoding rule has a particular form since only a narrow band of symbols can be nonzero.

Definition 7.1.3. *The N by nN* generator matrix *for a convolutional code, $G(N)$, is defined by*

$$y = uG(N)$$

where y is the encoded vector defined by (7.1).

Example 7.1.2. (Example 7.1.1 continued).
For the code with $R = \frac{1}{2}$ and $g = (11.10.11)$, the generator matrix $G(8)$ is obtained in the following way: in the first row of the generator matrix, g is followed by $2N - 6$ zeros. Row 2 and the following rows are obtained as cyclic shifts of the previous row by two positions:

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

For codes of rate $R = \frac{1}{2}$, the parity check matrix has the same form as G, and the nonzero sequence h, which we refer to as the parity sequence, is obtained by simply reversing g. The orthogonality of G and H follows from the fact that each term in the inner product appears twice.

Example 7.1.3. (Example 7.1.2 continued).
The parity check matrix has rows that are shifts of $h = (11.01.11)$. Thus by letting the first row represent the first parity check involving the first block, $H(8)$ becomes

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

The parity symbol in block j is associated with a row of H that has its last nonzero entry in position $2j$. Thus the parity symbol is a function of previously received symbols.

Definition 7.1.4. *A* syndrome *for a rate* $1/2$ *convolutional code defined as*

$$s_j = \sum_{i=0}^{2M+1} r_{2j-i} h_i, \tag{7.2}$$

where r *is the received sequence;* $s_j = 0$ *if* r *is a codeword.*

Lemma 7.1.4. *A single error affects at most* $M+1$ *syndrome bits.*

We leave the generalization to $R = 1/n$ as an exercise.

7.2 Codewords and minimum weights

For block codes the Hamming distance between codewords gives important information about the error-correcting properties. In particular, the minimum distance is important. We present a similar parameter for convolutional codes.

If we eliminate the last M bits of the input u (or force them to be zero), the encoding (7.1) becomes a regular non-cyclic convolution. This way of adapting the convolutional encoding to a particular frame length is called *termination*.

Definition 7.2.1. *Consider* j *consecutive rows of the generator matrix and the segment of the encoded sequence where these rows are not all zero, the length of* *the sequence is $N' = (j + M)n$. The set of encoded sequences is a linear (N', j) code called the j*-th terminated code.

Thus a terminated code has rate less than R, but for a long frame the difference is small. However, in the analysis of convolutional codes it is also useful to consider short terminated codes.

Definition 7.2.2. *The j*-th row weight *of the convolutional code is the minimum weight of a nonzero vector spanned by j consecutive rows.*

Thus the j-th *row weight* is the minimum distance of the j-th terminated code. Since each terminated code contains all shorter codes, we have

Lemma 7.2.1. *The row weight is a decreasing function of j.*

Definition 7.2.3. *The* free distance*, d_f, is the minimum value of the row weights.*

In most codes of interest the row weight is (almost) constant from $j = 1$, or it is reached with a very small value of j. The free distance d_f is also the minimum distance of the convolutional code as defined in Definition 7.1.2 for any sufficiently large N. The free distance is the most important measure of the error-correcting capability of a convolutional code.

Theorem 7.2.1. *A convolutional code with free distance d_f can correct any error pattern of weight less than $\frac{d_\mathrm{f}}{2}$ if N is sufficiently large.*

Theorem 7.2.2. *The free distance of a convolutional code is upper bounded by the minimum distance of the best $((j + M)n, j)$ block code.*

Proof. The terminated code is a block code with these parameters, and its minimum distance, the row weight, is an upper bound on the free distance of the convolutional code. □

Since the largest value of the minimum distance of any block code with given parameters is known for a large number of cases, it is possible to obtain close bounds on the free distance. In spite of this important connection between the minimum distances, the result does not suggest any constructions.

Example 7.2.1. We may obtain an upper bound on the distance of a $R = 1/2$ code with $M = 3$ by considering the parameters of block codes obtained by termination: $(8, 1), (10, 2), (12, 3), \ldots$. It follows that the upper bound is 6.

If a convolutional encoder is started in an arbitrary state and makes j consecutive state transitions, we call the corresponding output an encoded segment of length j. We have used the term encoded sequences to refer to the output for the entire input sequence, whereas so far we have avoided the terms ‘codeword’ in relation to convolutional codes. It is convenient to break long encoded vectors into shorter segments, which are often referred to as the codewords of the convolutional code.

Definition 7.2.4. *A* codeword *of length j in a linear convolutional code is an encoded segment that starts in state* 0*, makes j transitions to end in state* 0*, and does not go through this state.*

Thus the zero word is the only word of length 1, and other words have length at least $M + 1$.

Example 7.2.2. (Example 7.1.1 continued).
The $(2, 1, 2)$ code has one word of length $1, 00$, and no words of length 2. The generator, 11.10.11 is the only word of length 3 and also the only word of weight 5. There are two words of weight 6, one with length 4 and one with length 5: (11.01.01.11) and (11.10.00.10.11).

Theorem 7.2.3. *The free distance is the minimum weight of a nonzero codeword.*

We previously defined row weights as minimum distances of terminated codes. The following definition introduces a concept that describes how the weights of codewords increase with increasing length.

Definition 7.2.5. *The j*-th extended row weight *is the minimum weight of a nonzero codeword of length $j + M$.*

We shall show in detail that a decoding error consists in choosing an encoded vector that differs from the one transmitted by the addition of a codeword. This is the motivation for studying the weight enumerator of a convolutional code.

Definition 7.2.6. *The weight enumerator $A(z)$ for a convolutional code is a formal power series*

$$A(z) = \sum_w A_w z^w, \tag{7.3}$$

where A_w is the number of codewords of weight w.

Sometimes it is convenient to replace z by two variables indicating the number of 0s and 1s in the codeword. In that way it is also possible to determine the length of the codewords. However, we leave this extension as an exercise.

The weight enumerator may be found by enumerating the codewords that start and end in state 0. In order to eliminate sequences that pass through this state, we need the matrix Φ^- which is obtained from Φ by eliminating all transitions from state zero (i.e., the first column is set to –).

Since we are only interested in the weights, we set up a system of linear equations where the entries are polynomials in z. The matrix $\Phi(z)$ is obtained from Φ by replacing all entries of weight w by z^w, and $\Phi^-(z)$ is obtained from Φ^- in the same way. An output of weight 0 is replaced by a 1, and zeros are filled in where no transitions occur (replacing –). The functions $\rho_i(z)$ indicate the weight enumerator for the segments of codewords that start in state 0, but end in state i (the row index); these functions now satisfy

$$\Phi^-(z)\begin{bmatrix}\rho_0(z)\\ \rho_1(z)\\ \vdots\\ \rho_r(z)\end{bmatrix}=\begin{bmatrix}\rho_0(z)\\ \rho_1(z)\\ \vdots\\ \rho_r(z)\end{bmatrix}-\begin{bmatrix}0\\ \phi_{10}(z)\\ \vdots\\ 0\end{bmatrix}, \tag{7.4}$$

where the entry $\phi_{10}(z)$ on the right side represents the transitions from state 0 to state 1. Since the transitions from state 0 were eliminated from Φ^-, the function ρ_0 enumerates the chains that end here, which are the codewords. Thus we can find $A(z)=\rho_0(z)$ by solving a system of symbolic equations. With a suitable programming tool that can be done even for a moderately large number of states. It follows from this calculation that we have

Theorem 7.2.4. *The weight enumerator of a convolutional code is the rational function $\rho_0(z)$.*

Example 7.2.3. (Example 7.1.1 continued).
The codewords of the code from Example 7.1.1 may be found by removing the zero cycle of length 1. This gives the system of equations

$$\begin{bmatrix}0&0&z^2&0\\0&0&1&0\\0&z&0&z\\0&z&0&z\end{bmatrix}\begin{bmatrix}\rho_0\\ \rho_1\\ \rho_2\\ \rho_3\end{bmatrix}=\begin{bmatrix}\rho_0\\ \rho_1-z^2\\ \rho_2\\ \rho_3\end{bmatrix}.$$

We obtain the weight enumerator as

$$A(z)=\rho_0(z)=\frac{z^5}{1-2z}=z^5+2z^6+4z^7+\cdots$$

The weight enumerator is primarily of interest as a tool for obtaining upper bounds on the error probability. By following the derivation of the union bound in Chapter 3, we get an upper bound on the probability that the decoder makes an error in the first block. Assume that the zero word is transmitted. An ML decoder will select one of the nonzero words leaving state 0 in the first block, and this will happen if the received sequence is closer to one of these words than to the zero sequence. Thus the same calculation as the one used to derive (3.14) gives

$$P_{\text{err}} < \sum_{w>0} \sum_{j \geq \frac{w}{2}} A_w \binom{w}{j} p^j (1-p)^{w-j}, \tag{7.5}$$

which may be approximated by

$$P_{\text{err}} < \frac{1}{2} A(Z), \tag{7.6}$$

where Z is the function

$$Z = \sqrt{4p(1-p)}.$$

Actually, the probability of making an error in the first block is smaller because later errors can cause the decoder to select a sequence where the error starts after the first block.

The bound (7.5) is closely related to (3.16) in Chapter 3. For a convolutional code we can use the rational form of the weight enumerator, or the numerical value of Z can be substituted into the linear equations before we solve them.

7.3 Maximum likelihood decoding

The process of decoding a convolutional code is conveniently described as a search for an encoded sequence that disagrees with the received sequence in the smallest number of positions. We present several properties of maximum likelihood decoding which lead to the important Viterbi decoding algorithm for convolutional codes (we use this name although the algorithm may be seen as a special case of the principle of dynamic programming).

Earlier we introduced the transition matrix Φ with entries $\phi_{i'i}$, which are the blocks that the encoder outputs in making a transition from state i to state i'. For a received block, $r_j = \zeta$, we may find the number of differences between $\phi_{i'i}$ and r_j as

$$e_{i'i} = w_{\text{H}}(\phi_{i'i} + \zeta)$$

where w_{H} indicates Hamming weight. Thus we may store these values in a matrix E_ζ (or a table) indexed by the values of r_j.

Example 7.3.1. Consider the decoding of the $R = 1/2$ code with $M = 3$ and

$$g = (11.10.01.11)$$

The encoder has memory 3 and eight states. The state transition matrix is

$$\Phi = \begin{bmatrix} 00 & - & - & - & 11 & - & - & - \\ 11 & - & - & - & 00 & - & - & - \\ - & 10 & - & - & - & 01 & - & - \\ - & 01 & - & - & - & 10 & - & - \\ - & - & 01 & - & - & - & 10 & - \\ - & - & 10 & - & - & - & 01 & - \\ - & - & - & 11 & - & - & - & 00 \\ - & - & - & 00 & - & - & - & 11 \end{bmatrix}.$$

For a received zero block, the $e_{i'i}$ are found as the weight of each entry of this matrix. We refer to this integer matrix as E_{00}. Similarly, for each of the other three values of a received block we need a table of the corresponding error weights. For 01 we have

$$E_{01} = \begin{bmatrix} 1 & - & - & - & 1 & - & - & - \\ 1 & - & - & - & 1 & - & - & - \\ - & 2 & - & - & - & 0 & - & - \\ - & 0 & - & - & - & 2 & - & - \\ - & - & 0 & - & - & - & 2 & - \\ - & - & 2 & - & - & - & 0 & - \\ - & - & - & 1 & - & - & - & 1 \\ - & - & - & 1 & - & - & - & 1 \end{bmatrix}.$$

Here the symbol '$-$' is used instead of ∞ (or a sufficiently large number).

For a BSC the Maximum Likelihood (ML) sequence for the frame is the one that maximizes the conditional probability, which is the one that minimizes the number of errors. The decoding decisions rely on the following observation:

Lemma 7.3.1. *If the maximum likelihood sequence passes through state i at time j, it has the smallest number of errors among all segments that start from the initial state and reach state i at time j*

Proof. Assume that another segment reaching state i at time j had a smaller total number of errors. We could then follow this segment for $j' < j$, but since the possible transitions after time j depend only on the state at time j, the rest of the sequence could be chosen as the ML sequence. In this way we would get a smaller total number of errors, contradicting the assumption. □

The principle of dynamic programming is that for each instance of time, j, and for each state, the optimal segment leading to that state may be computed by extending the optimal segments at time $j - 1$. Let the accumulated number of errors on the segment leading to state i be $\mu_i(j - 1)$. Each segment may be extended in two different ways, and similarly each state may be reached from two previous states. For each possible received block $r(j) = \zeta$ we find the number of errors associated with a particular state transition, $e_{i'i}(j)$, from the relevant E_ζ table. The new errors are added to the accumulated number, and we select the best segment into each state:

$$\mu_{i'}(j) = \min_i [\mu_i(j-1) + e_{i'i}(j)]. \tag{7.7}$$

The updated μ-vector is used in the next computation, and in the examples we store them in a matrix. However, the old values are not needed, and in an actual decoder it is sufficient to have storage for the two vectors involved in the current computation. The segment reaching state i at time j is selected among the transitions from possible previous states. For each state and each instance of time, the index, $\tau_i(j)$, of the selected previous state must be stored (if the minimum in (7.7) is not unique, one is selected arbitrarily). When we reach the end of the frame and find the smallest cumulated number of errors, these return addresses are used to find the decoded sequence.

We can now state the Viterbi algorithm:

Algorithm 7.3.1. ML *Decoding* (*Viterbi*).
Input: *Received sequence.*

1. *Initialialize* $\mu_i(0)$ *as 0 for possible initial states and* ∞ *for other states.*
2. *For each received block, update* $\mu_i(j)$ *according to* (7.7) *and store the minimizing value of* i *as* $\tau_i(j)$.
3. *Among the possible final states, select the one,* $\sigma(N) = \sigma_{i'}$, *that has the smallest* $\mu_i(N)$.
4. *Working back from the final state, set* $b(N) = \tau_{i'}(N)$. *Find the states on the optimal path as* $\sigma(j-1) = \sigma_{b(j)}, \underline{\ }(j-1) = \tau_{b(j)}$.
5. *Find the encoded block,* $y'_j = \phi(b(j+1), b(j))$ *or the information symbol* $u'_j = b(j+1) \mod 2$.

Output: *Decoded information sequence* u'.

It follows from the derivation that we have

Theorem 7.3.1. *The Viterbi algorithm determines the maximum likelihood encoded sequence.*

For a terminated code, we let the zero state be the only initial and final state.

Example 7.3.2. (Example 7.3.1 continued).

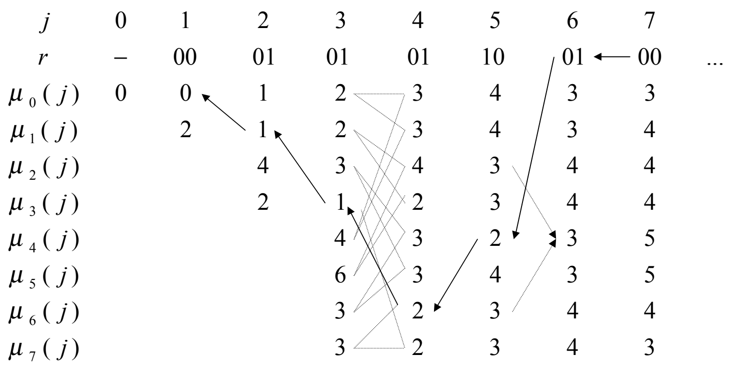

Figure 7.1. Decoding of an 8-state convolutional code. The columns show the vectors $\mu(j)$ for time $j = 1, \dots, 7$. $\mu(j)$ is calculated from the current input and $\mu(j-1)$ using (7.7), the calculation of $\mu_4(6)$ is indicated by forward arrows. The full set of connections between the states is shown as dotted lines between time 3 and 4, this graphical representation of the Φ matrix is often called a trellis. The state at time 7 is assumed to be 0, and the back arrows indicate the trace-back of the decoded sequence.

If the encoder is started in state 00 and the received sequence is

$$r = (00.01.01.01.10.01.00)$$

we calculate the vectors $\mu(j)$ using (7.7) and the matrices E_{00}, E_{01}, and E_{10}. Since each segment can be extended by an input of 0 or 1, two states are reached from state 0 at time $j = 1$, 4 at time $j = 2$, and all eight at time $j = 3$. From time 4 each state can be reached by two branches, and we select the one with the smallest number of errors. The decoding process is illustrated in Figure 7.1.
Thus at time 6 state 4 can be reached from state 2 or 6. In the first case the segment already had three errors and none is added in the transition, whereas in the last case there have been three errors in the first five blocks and the transition to state 4 adds two additional errors. Thus 2 is stored as $\tau_4(6)$ Assuming that the final state here is the zero state, we can choose $y = (00.11.01.11.10.11.00)$ as the transmitted sequence and $u = (0110000)$ as the information.

Any time the decoded sequence leaves the correct sequence of states, a decoding error occurs, and we shall say that the error event continues until the sequence reaches a correct state (whether all blocks between these two instances are correct or not). We shall refer to such a decoding error as a *fundamental error event*.

Lemma 7.3.2. *A fundamental error event of length, L, is the segment leaving state* 0 *at time j and reaching state* 0 *for the first time at time* $j + L$, *i.e., a codeword of length L.*

Proof. It follows from the linearity of the code that we may find the error sequences by adding the two sequences, and that the result is itself a codeword. As long as the segments agree, the sum is 0 and the finite state machine stays in state 0. The sum sequence returns to state 0 when the two sequences reach the same state. □

The probability of such an error event is related to the extended row weight.

Lemma 7.3.3. *For a fundamental error event of length L to occur the number of errors must be at least half of the $(L-M)$-th extended row weight.*

In Section 7.2 we derived the upper bound (7.5) on the probability of an error event starting in the first block. We can interpret the bound as an upper bound on the probability that an error event starts in any particular block j, sometimes called the *event error probability*. This probability is smaller than the probability of an error starting in the first block because the decoder may select a wrong branch starting before time j. To arrive at the expected number of wrong blocks, the probabilities of the error events must be multiplied by their length.

The Viterbi algorithm is a practical way of performing maximum likelihood decoding. The complexity is directly related to the number of states, and thus it increases exponentially with the memory of the code. Decoders with a few thousand states can be implemented. It is an important aspect of the algorithm that it can use 'soft decision' received values, i.e., real values of the noisy signal, rather than binary values. The weights e_{ij}, which we derived as the number of errors per block, may be replaced by the logarithm of the conditional probability or a suitable approximation. Decoding for Gaussian noise channels is discussed in Appendix B.

For the BSC we could get a slightly more compact version of the Viterbi algorithm by using the syndrome bit as the input in each transition and searching for the error pattern of minimal weight. However, the version above, which works directly on the encoded and received sequences, is more widely applicable.

7.4 Maximum likelihood decoding of block codes and tail-biting codes

In principle, a block code can be decoded by calculating the syndrome and using a table relating the syndrome to the error patterns of minimal weight. However, this approach is practical only for very short codes. There are a few special cases where other approaches are known. As a simple example of a block code we described the orthogonal and biorthogonal codes. In these cases the code vectors are also orthogonal as real vectors if the symbols $0, 1$ are replaced by $1, -1$, and the closest codeword can be found by correlating the received vector with the codewords. This approach can be implemented as a fast transform, and also works if the received vector has real values (soft decisions).

If a convolutional code is encoded by using circular convolutions of an input vector of length N, we refer to the code as *tail-biting*. For a given memory M, we consider a sequence of such codes with increasing N.

Lemma 7.4.1. *The rate of a tail-biting code is the same as the rate of the convolutional code when* $N > M$.

Proof. For $R = 1/2$ we get the two encoded sequences in polynomial form as $u(x)g_0(x)$ and $u(x)g_1(x)$, both modulo $x^N - 1$. Since $u(x)$ has degree less than N, these sequences can be all zero only if a nontrivial factor in $x^N - 1$ is also a factor in g_0 and g_1. However we have made the assumption that the two generators do not have any common factors. Thus the N shifts if the generating sequence are linearly independent. The same argument applies for other values of n. □

An equivalent form of the generator matrix for a rate $\frac{1}{2}$ code can be obtained by collecting the even numbered columns into a *circulant matrix* and the odd numbered columns into a matrix with the same structure. Such codes are sometimes called *quasi-cyclic*. Many good codes with small block lengths have generator matrices of this form.

Example 7.4.1. A short tail-biting code.
A $(16, 8, 5)$ block code may be described as a rate $\frac{1}{2}$ tail-biting code with generator sequence $g = (11.11.00.10)$. This code has the largest possible minimum distance for a linear (16, 8) code, although there exists a non-linear (16, 8, 6) code. Longer codes with the same generator also have minimum distance 5.

A tail-biting code can be decoded by applying the Viterbi algorithm. In principle, one should make several attempts, starting in each state at the beginning of the word and finding the closest codeword ending in the same state. However, a very good approximation can be obtained by starting in an arbitrary state and going through the codeword in a cyclical fashion. It is known that there are cases where this procedure fails because the resulting path goes through the N positions twice (or more times) before connecting to the same state.

It is possible to rearrange the positions of any block code to allow a similar sequential calculation of the ML codeword, however for most codes this will require an irregular algorithm and more states. If ML decoding is desirable, one would prefer to use a tail-biting version of a convolutional code, even though it may not have the best possible minimum distance (we have discussed earlier that the minimum distance is not necessarily essential to the performance of ML decoding).

For a tail-biting code of rate $\frac{1}{2}$, the typical minimum distance is a little more than M. For Viterbi decoding to be feasible, we cannot have M much greater than 10, but that means we can decode fairly good block codes of length up to about $n = 100$. Clearly such codes are much too long for an approach based on syndrome tables. Tail-biting codes are not long enough to be useful for communicating data, but they are an attractive solution as header protection and in other applications involving small blocks of data.

7.5 Punctured codes

In Problem 1.5.9 we have considered two methods for decreasing the length of a block code by one symbol. We give the definition here to emphasize the terminology.

Definition 7.5.1. *A code is* shortened *when an information symbol is first forced to be zero and then eliminated. A code is* punctured *when a parity symbol is eliminated.*

Thus the rate is decreased by shortening and increased by puncturing the code. The opposite operations are called lengthening and extending. We have

Lemma 7.5.1. *The dual of a shortened code is the punctured dual code.*

Proof. The dual of the original code has a parity symbol where the original code has an information symbol. If this symbol is forced to be zero, the remaining positions satisfy all the parity checks, and the position can be punctured. □

These concepts may be applied to convolutional codes, but usually symbols are punctured in a periodic pattern through the entire frame or a predefined part of the frame. The main advantage of puncturing is that the rate of the code can be increased while the same decoder can be used. In calculating the distance to the received sequence, the punctured symbol is just ignored.

Most of the high-rate codes that are used in current systems are obtained by puncturing codes of rate $\frac{1}{2}$. Puncturing may be used to change between different rates depending on the bit error probability, or the technique may be used to vary the number of correctable errors within a frame, so that some information symbols are given additional protection.

Example 7.5.1. Starting from the rate $\frac{1}{2}$ code in Example 7.1.1, we can puncture every other parity symbol to get a rate $\frac{2}{3}$ code with alternating generators $g' = (11.1.11)$ and $g'' = (1.10.1)$. We still have $M = 2$. In order to find the parity checks we can solve a system of linear equations, or we can shorten the original parity checks by eliminating the same symbols. Before we can eliminate the symbols, we must find a linear combination of in the parity check matrix such that all the relevant symbols are zero. In this case it is simply the sum of three consecutive shifts of h, $h' = (1.11.0.11.1.10)$.

7.6 Correctable error patterns and unit memory codes

In the definition of convolutional codes with memory M and rate $R = 1/n$, the encoder makes a state transition for each input symbol. It can be more practical to perform the encoding in fewer steps by segmenting the input sequence u into vectors u_j of length $k \geq M$ (thus in this section the index refers to a block of symbols rather than a single symbol). The part of the generator matrix that is nonzero for such an input vector consists of k rows and $(k+M)/R$ columns. We write this submatrix as G_0, G_1, where both have size k by $n' = k/R$ (possibly allowing some zero columns in G_1), where G_0 has rank k and G_1 has rank M. In this way we can express the encoding as

$$y'_j = u_j G_0 + u_{j-1} G_1. \tag{7.8}$$

When the encoding rule is expressed in this way, we refer to the code as a (partial) *unit memory code* (UMC). The term indicates that the memory of the code is (part of) a single block of information symbols. Measured in bits, the memory is still M. The encoder states are still given by the previous M input symbols, but in each step it is now possible to reach each of the states in 2^k ways.

When $M < k$, we can write the generator matrices as

$$G = [G_0, G_1] = \begin{bmatrix} G_{00} & 0 \\ G_{01} & G_{11} \end{bmatrix},$$

where G_{00} is a $k - M$ by n' matrix.

For block codes we know that all errors within a radius of $d/2$ are decoded, and that at least in principle most errors within a larger sphere can be corrected. In this section we provide a similar description of the error patterns that are corrected by a convolutional code. For this purpose we consider a more general class of codes where G_0, G_1 can be chosen freely with the given size and rank. In this way we can use generator matrices of known block codes as components in the construction. It is convenient to assume that the linear subspaces spanned by G_{00}, G_{01}, and G_{11} share only the zero vector. With this definition of a unit memory code we have

Lemma 7.6.1. *The dual of a* UMC *is also a* UMC.

Proof. Any vector in the code belongs to the space spanned by the rows of G_{00}, G_{01}, and G_{11}, and the dimension of this space is $k + M$ (it may be the whole space). Thus if H_{00} is a basis for the dual space, it generates the parity checks that are satisfied by all blocks (it might be just the zero vector).

We may find the parity checks of the code by solving the system of linear equations

$$\begin{bmatrix} G_{00} & 0 \\ G_{11} & 0 \\ G_{01} & G_{11} \\ 0 & G_{01} \\ 0 & G_{00} \end{bmatrix} \begin{bmatrix} H'_{00} & H'_{01} & 0 \\ 0 & H'_{11} & H'_{00} \end{bmatrix} = 0. \tag{7.9}$$

It follows from our assumptions that the rank of the coefficient matrix is $2k + M$, and the rank of H_{00} is $n' - k - M$. Thus the dimension of the remaining solutions is

$$2n' - 2k - M - 2(n' - k - M) = M.$$

Thus we have determined a repeated set of $n' - k$ parity checks of the form corresponding to the definition. Notice that without the assumption that the subspaces spanned by the three matrices in G be disjoint (except for the zero vector), the dual code might have a different structure, and in particular it could have a memory greater than 1 block. □

Example 7.6.1. A double error-correcting unit memory code.
An (n', k) UMC may be defined by the parity check matrix

$$H = [H_0, H_1] = \begin{bmatrix} e & 0 \\ H_{01} & H_{11} \end{bmatrix},$$

where the block length is $n' = 2^{m-1}$, and e is an all 1s vector. The remaining syndrome bits are specified as for the BCH codes correcting two errors: the columns of H_{01} are powers of a primitive $(n' - 1)$-th root of unity, α, and a zero column; the columns of H_{11} are powers of either α^3 or α^{-1}, which must also be a primitive $(n' - 1)$-th root. The algebraic structure allows us to give an outline of a decoding algorithm for correcting error patterns containing at most $i + 1$ errors in any segment of i consecutive blocks:

Algorithm 7.6.1. *Decoding a* UMC.
Input: *Received word.*

1. *Assume that the errors have been corrected up to block $j-1$. Calculate the next m syndrome bits, which involve only received block r_j, $s_j = H_0 r_j^T$. Since H_0 is also the parity check matrix for an extended Hamming code, we can immediately correct single errors and detect double error.*

2. *When a single error has been corrected, we can remove the term $H_1 r_j^T$ from the following syndrome and continue.*

3. *If there are two errors in block j, this will be detected as even parity and a nonzero syndrome. In this situation the decoding is delayed until a block with even parity occurs.*
4. *In the following blocks we detect at first only the parity. Blocks with odd parity contain single errors, and they will be decoded later.*
5. *Since we have assumed that i blocks contain at most $i + 1$ errors, and the first block contained two errors, the next block with even parity must be correct.*
6. *We can then decode the previous blocks in the reverse direction using H_{11}, since we assumed that all columns of this matrix are distinct.*
7. *Finally the two errors in block j can be decoded using a decoding method for double error correcting* BCH *codes, since we now have both syndromes available.*

Output: *Decoded word.*

The example shows how convolutional codes can use available syndrome bits to correct slightly more errors than a block code, but this advantage comes at the expense of a variable decoding delay.

The set of error patterns that are characterized by the maximum weight of i consecutive blocks (related to the extended row weights of the code) can be described by a finite state machine. If we consider a matrix, T, where the entry i, j indicates the number of error patterns (usually of the same weight) associated with a transition from state j to state i, we find the number of error patterns of length l between two states as T^l, and the total number of error patterns increases as λ^l, where λ is the maximal eigenvalue of T. Since the number of correctable error patterns cannot exceed the number of syndrome sequences, we have a convolutional code version of the Hamming bound:

Theorem 7.6.1. *The rate of a convolutional code that corrects all error patterns with weight characterized by the transition matrix T is upper bounded by $R \leq 1 - \log_2(\lambda)/n'$.*

We mention that the total set of error patterns corrected by a given convolutional code can be described by a finite state machine, but in most cases the number of states is extremely large.

Example 7.6.2. Consider the $(8, 4)$ unit memory code generated by the matrix:

$$G = \begin{bmatrix} e & 0 \\ G_0 & G_1 \end{bmatrix},$$

where

$$G_0 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}, \quad G_1 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

and e is a vector of 1s. This code has one nonzero codeword of length 1 block. It has weight 8. Other nonzero codewords have weight at least 4 in the first and last nonzero block and weight at least 2 in the block between these. Thus the sequence of extended row distances is $[8, 8, 10, 12, \ldots]$. The code can correct at least 3 errors in one block and $j + 1$ errors in j blocks. (The code is a special case of the codes from the previous example, but actually corrects an additional error in some cases). The transition matrix for a finite state machine enumerating the correctable error patterns is

$$T = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 8 & 8 & 1 & 0 \\ 28 & 28 & 8 & 1 \\ 56 & 0 & 0 & 0 \end{bmatrix}.$$

The largest eigenvalue of this matrix is 13.7425 and $\log_2(\lambda) = 3.791$. Thus the rate of the code correcting this set of error patterns is upper bounded by $R < 4.209/8 = 0.526$.

7.7 Problems

Problem 7.7.1. Consider a rate $\frac{1}{2}$ convolutional code with generating vector $g = (11.10.01.11)$.

(1) How is the information sequence $u = (10111000)$ encoded?

(2) What is the memory of the code?

(3) How many states does the encoder have?

Problem 7.7.2. Consider the same code as in Problem 7.7.1.

(1) Find the parity check sequence for the code.

(2) Find the syndrome for a single error.

(3) In a long code the syndrome is found to be $(\ldots 00011000 \ldots)$. Find a likely error pattern.

(4) Same question with the syndrome $(\ldots 0001000 \ldots)$.

Problem 7.7.3. Consider the same code as in Problems 7.7.1.

(1) Find the row distances of the code and the free distance.

(2) How many errors can the code correct?

(3) Give examples of patterns with higher weight that lead to decoding error.

Problem 7.7.4. Consider the code from Problem 7.7.1.

(1) Find the weight enumerator for the code.

(2) How many codewords are the of weight 6, 7 and 8?

(3) Evaluate the upper bound on the event error probability for $p = 0.01$.

Problem 7.7.5. A convolutional code with $M = 6$ has generator matrix

$$G(x) = [g_0(x), g_1(x)] = \left[x^6 + x^5 + x^2 + x + 1, x^6 + x^4 + x^3 + x^2 + 1\right].$$

The free distance is 10.

(1) Show that g_0 is irreducible, while g_1 factors into two binary polynomials.

(2) Find the information polynomial, $u(x)$, of lowest degree such that for some j,

$$u(x)g_1(x) = 1 + x^j.$$

(3) Find the weight of the encoded sequence for this information polynomial.

Problem 7.7.6. Consider again the rate $\frac{1}{2}$ convolutional code with generating vector

$$g = (11.10.01.11).$$

(1) Which of the following error patterns will always be correctly decoded?

$$\begin{aligned} a&: \ 00.10.00.00.10.00\ldots \\ b&: \ 01.10.00.00.00.00\ldots \\ c&: \ 11.00.01.00.00.00\ldots \\ d&: \ 11.00.00.00.00.11.00\ldots \\ e&: \ 11.00.10.00.00.00\ldots \\ f&: \ 00.00.00.01.11.00\ldots \end{aligned}$$

(2) Find an upper bound on the error probability using (7.5) when $p = 0.01$.

(3) What is the largest value of p such that (7.5) gives a probability < 1?

(4) Write out the E matrices needed for decoding.

(5) Assume that there is a single error in the first block. Find those weights $\mu(i)$ that are less than 4. Repeat the problem with two errors in the first block.

Problem 7.7.7. *Project.*

(1) Write a program for encoding and decoding the code with

$$G(x) = [g_0(x), g_1(x)] = \left[x^4 + x + 1, x^4 + x^3 + x^2 + 1\right]$$

(2) Decode terminated frames of length 1000 information bits with bit error probability $p = 0.02$ and $p = 0.05$.

(3) Compare the results to the upper bounds.

(4) Modify the encoder/decoder to use tail/biting frames.

Chapter 8

Combinations of Several Codes

Long codes are often required to obtain sufficient reliability, but a single long block code cannot provide the required performance with an acceptable amount of computation. Here we give an introduction to coding methods where several codes, or several stages of coding, are combined to make a long code which can be decoded using the decoding algorithms for the component codes. The difficulty of constructing good long codes led to this approach in an early stage of the study of error-correcting codes, and it continues to be a focal point of much development of communication systems. The total code may not have a very good minimum distance, but in most cases error patterns will be decoded even though the weight exceeds half the minimum distance. Thus a code with a relatively small number of low-weight words can still have an acceptable performance.

8.1 Product codes

Product codes are the simplest form of composite codes, is still a construction of practical interest.

Definition 8.1.1. *A* product code *in $\mathbb{F}_q$ is a vector space of n_1 by n_2 arrays such that each row is a codeword in a linear (n_1, k_1, d_1) code, and each column is a codeword in a linear (n_2, k_2, d_2) code.*

In this section we assume that the component codes use the same field. In a later section we extend the construction to two different fields.

Theorem 8.1.1. *The parameters of a product code are*

$$(N, K, D) = (n_1 n_2, k_1 k_2, d_1 d_2).$$

Proof. The information symbols may be arranged in a k_1 by k_2 array. The parity checks on rows and columns of information symbols may be computed independently. The parity checks on parity checks may be computed in two ways, but it follows from linearity that the results are the same: The last rows may be calculated as a linear combination of previous rows, and consequently they are codewords in the row code. Thus (N, K) are the products stated in the lemma. The product of the minimum distances is a lower bound, since a nonzero codeword has at least one nonzero row of weight at least $d_1$and each nonzero column has weight at least d_2. A minimum distance codeword of weight d_1 may be repeated in d_2 rows to form a codeword of weight exactly $d_1 d_2$. In fact these are the only codewords of that weight. □

For simplicity we shall assume that the row and column codes are identical with parameters (n, k, d) unless different parameters are explicitly given.

In the simplest approach to decoding, each column of the received word is first decoded separately, and a decoding algorithm for the row codes is then applied to the corrected word. We refer to such an approach as *serial decoding*.

Lemma 8.1.1. *If a product code is decoded serially, any error pattern of weight at most* $\frac{D-1}{4}$ *is decoded.*

Proof. If d is odd, the worst case situation is that several columns contain $\frac{d+1}{2}$ errors and are erroneously decoded. However, given the total number of errors, there can be at most $\frac{d-1}{2}$ such columns, and the errors will be decoded by the row codes. Of course, if the rows had been decoded first, the errors would have been decoded immediately. On the other hand, an error pattern with at least $\frac{d}{2}$ errors in each row and column cannot be decoded immediately, and thus we cannot be sure to decode $\left\lceil \frac{d}{2} \right\rceil^2$ errors. □

This is clearly a rather disappointing result. However, serial decoding can usually correct a much larger number of errors. We get a more realistic estimate of the error-correcting power of a product code from the following result:

Theorem 8.1.2. *A product code with component* (n, k, d) *codes can correct any error patterns such that the number of errors in any* j *by* j *submatrix is less than* $jd/2$*.*

Proof. Two codewords that differ on j' rows and j'' columns have a difference of weight at least jd, where $j = \max[j', j'']$. Thus there is a unique word satisfying the condition of the lemma. □

Clearly the total weight of the error pattern must be $W < nd$. With a random distribution of errors, we usually get the submatrix with the highest density (average number of errors per row and column) by removing rows and columns of weight $< d/4$. Thus this submatrix is usually very large. No simple decoding algorithm is known for correcting exactly the error patterns described by the lemma, but in Chapter 10 we discuss a practical algorithm that in most cases corrects even more errors.

The following lemma is useful for estimating the probability of low-weight errors:

Lemma 8.1.2. *If the numbers of minimum weight codewords in the two component codes are a_1 and a_2, the number of words of weight $d_1 d_2$ in the product code is $a_1 a_2$.*

Proof. Since the weight of a nonzero row is at least d_1 and the weight of a column is at least d_2, a minimum weight codeword can have only d_1 nonzero columns and d_2 nonzero rows. Thus the number of codewords of that weight is at most the number of ways of choosing the nonzero rows and columns so that there is a minimum weight codeword on exactly those positions. On the other hand, for any choice of a minimum weight row word and column word there is a corresponding minimum weight word in the product code. □

The number of codewords of higher weights is not related to the weight distribution of the component codes in any simple way. If the component code contains several words of weight d within j positions, these words could combine to words in the product code of weight jd. In the following theorem we apply a crude upper bound to the number of such words.

It follows from Theorem 8.1.2 that the contribution to the error probability from a word of low weight jd can be upper bounded by

$$\binom{n}{j}^2 \binom{j^2}{jd} 2^{jd} p^{jd/2}(1-p)^{jd/2}. \tag{8.1}$$

For a low-weight codeword, i.e., small values of d and j, and an error probability about $p = d/(2n)$, the exponent of n in this bound is seen to be $2j - jd/2$. Thus the probability decreases with increasing values of d, and it decreases with j when $d > 4$. That is to say, the probability of decoding error is dominated by words of weight d^2, and for lange n this probability can be quite small as long as $d > 4$. Products of (extended) Hamming codes, on the other hand, are subject to decoding errors of low weight, and error patterns with 2 errors in each row and column cause problems. For $d \geq 5$ the limitations of product codes are related to decoding failures involving a large number of rows and columns.

Example 8.1.1. Product of binary codes.
Consider a product code where both component codes are binary $(15, 4, 8)$ codes. In a cyclic version of this code, all 15 nonzero words are cyclic shifts of the same sequence. It follows from the results in this section that the minimum distance is 64 and there are 225 codewords of this weight. Clearly all codewords have weights that are multiples of 8, since any nonzero row or column has weight 8. For a word to have j zero columns, all rows must be component codewords with zeros in the j positions. Inspection of the sequence shows that there are 3 code-

words with zeros in 3 positions, but no set has zeros in exactly 2 positions. It follows that the only possible weights other than 0 and 64 are $8 \cdot 12, 8 \cdot 14, 8 \cdot 15$. We can use Theorem 3.4.1 and Lemma 3.4.1 from Chapter 3 to get the entire weight distribution as $1 + 225z^{64} + 7350z^{96} + 37800z^{112} + 20160z^{120}$. Theorem 8.1.2 indicates that any error pattern of weight less than 60 can be corrected if the errors are evenly distributed, but the probability of error is high if the errors are randomly distributed. Lemma 8.1.1 shows that serial decoding will always decode 15 errors, and in most cases several more errors are likely to be decoded. If errors occur randomly with probability $1/5$, there are on the average 45 errors. The contribution to the decoding error from the low-weight codewords is

$$P_{64} \leq 225 \cdot \binom{64}{32} (4/25)^{32} \simeq 1.5i0^{-5}.$$

An experiment shows that 45 errors are usually corrected by serial decoding.

We can find a close approximation to the probability of decoding failure with serial decoding by first bounding the number of errors after decoding columns. If errors occur independently with probability p, we can find the probability of decoding error in the columns from the binomial distribution. For even d, error patterns of weight $d/2$ are left undecoded. For weights $> d/2$, some errors are not decoded, while in other cases at most $t < d/2$ errors are added in wrong positions in the relevant columns. From a bound or an estimate on the probability of decoding error we can find the average number of errors, w_c, in the codeword after column decoding. The fraction of errors is consequently w_c/n^2. If we make the assumption that these errors are randomly distributed in the codeword, the probability of error in a row is upper bounded by the binomial distribution. This approach overestimates the error probability, since the symbol errors from the first stage of decoding are located in a limited number of columns. If the errors on the channel are likely to be concentrated in a few columns or rows (as may be the case in a storage medium), serial decoding is more effective.

Theorem 8.1.3. *The probability of decoding failure for a product code is upper bounded by*

$$P_{\text{fail}} \leq n \sum_{j>t} \binom{n}{j} \left(\frac{w_c}{n^2}\right)^j \left(\frac{n^2 - w_c}{n^2}\right)^{n-j} \tag{8.2}$$

Proof. Assuming that we have the actual value of w_c rather than an estimate, we replace the actual distribution of errors in the word by a distribution that is independent for each symbol. This approach overestimates the probability of having more than t errors in a row, since after an error is known to occur in a certain row, at least $t - 1$ are in the same column, and consequently cannot occur in the same row. After calculating the probability of error for a row, we apply the union bound by multiplying this number by the number of rows. □

Example 8.1.2. Product of RS codes.
Products of RS codes have been used in storage media. The commonly used codes are based on 8-bit symbols, $q = 256$. For a given overall dimension, $k_1 k_2$, the minimum distance is clearly maximized by taking $d_1 = d_2$. However, serial decoding is more effective when the first code to be decoded has larger minimum distance. We choose $d_1 = 8$ and $d_2 = 6$. Thus 3 errors are decoded in each column, and 4 errors are detected, but no correction takes place. If more than 4 errors occur, the extra syndrome ensures that the probability of decoding error is less than $1/256$, and we will ignore this contribution in the calculation. Assuming that errors occur independently with probability p, the average number of errors after column decoding is

$$w_c = \sum_{j>3} j \binom{256}{j} p^j (1-p)^{256-j}.$$

We then make the approximation that these errors are randomly distributed and an average of $w_c/256$ occur in each row. For $p = 1/256$ we get on the average 0.08 errors per column after decoding. Assuming that this is also the average per row, the average number of errors in a word of the product code is 0.06 after the second stage of decoding.

8.2 Products of Reed–Solomon and binary codes (concatenated codes)

8.2.1 Parameters of concatenated codes

In this section we extend the concept of product codes to constructions where the column codes are binary, but the row codes are Reed–Solomon codes. (Such codes are traditionally referred to as 'concatenated': two encoders/decoders were placed next to each other). The first step in such a construction is to represent the RS code as a binary code.

Definition 8.2.1. *The* binary image *of a codeword in a* (N, K, D) RS *code over a field of* 2^m *symbols is an array of size* m *by* N *where each column is a binary representation of the corresponding symbol in the* RS *code. We refer to the set of such codewords as the* binary image *of the* RS *code.*

Lemma 8.2.1. *If the field* $\mathbb{F}_{2^m}$ *is represented by* m *bit binary vectors in such a way that addition in the field is modulo* 2 *addition of vectors, the binary image of an* RS *code is a binary* (mN, mK, D') *code with* $D' \geq N - K + 1$.

Proof. We can find a basis for the binary image from a generator matrix for the RS code, where we assume that the matrix has a K by K unit matrix on the left. Each row is converted to m basis arrays for the image by scaling the first nonzero element to get the symbols represented by the m unit matrices and taking the binary images. Since all elements in the big field are linear combinations of these m elements, we can get all images of the RS codewords as linear combinations of the Km binary arrays (thus we do not need to perform multiplications). The minimum weight of the binary image is at least the Hamming weight of the RS codeword. □

The real minimum distance is difficult to analyze, and also of minor importance to the decoding. However, we can get an upper bound using the construction of BCH codes (Chapter 5):

Lemma 8.2.2. *The minimum distance of the binary image of an* (N, K, D) RS *code is upper bounded by the minimum distance of the* BCH *code* C_K(sub).

Proof. As discussed in Chapter 5, the BCH code is contained in the RS code. Taking the binary image of a codeword that is already binary does not change its weight. Since the generator polynomial of the BCH code contains all roots in the generator polynomial of the RS code and some additional roots, its minimum distance may be greater than D. For RS codes of low rates, the bound is trivial, $D \leq N$. □

Here we combine the binary images of RS codes with a second binary coding stage in a way that is similar to the product codes. Let the information bits be represented as an mI by K array. Each set of m rows of binary information symbols is encoded as the binary image of an RS code, the outer code. Each column, which consists of a symbol from each RS code, is then encoded as a codeword from an (n, Im, d) block code, the inner code. The parameter I is referred to as the interleaving degree.

Definition 8.2.2. *A* product *of (outer)* (N, K) *RS code, and a binary* (n, Im) *(inner) code is a binary* (nN, KIm)*, where each column is a codeword in the inner code, and each array of the* I *arrays of* m *bits from each inner codeword is the binary image of a codeword in the RS code.*

Note that since the code is usually designed for serial decoding, we do not place any restrictions on the $n - k$ rows of parity symbols. Clearly, we could modify the construction to require this number to be a multiple of m and make this part of the array images of RS codewords.

Theorem 8.2.1. *The minimum distance of the product code is at least* Dd.

Proof. In a nonzero word at least one RS code has $D = N - K + 1$ nonzero symbols. These are encoded by the inner code, which has weight at least d in each nonzero column. □

While a product code has many codewords with weight equal to the lower bound, a concatenated code may have none or very few. If D' is the minimum distance of C_K(sub), there are codewords of weight $D'd$.

If the probability of decoding failure for the column code is moderately low, say less than 0.01, a long RS row code can correct the errors with high reliability and a rate close to 1. From a theoretical point of view, it is possible to construct very long codes with minimum distances that are at least a large fraction of the lower bounds for the best codes. If the inner code is long enough to give a moderate performance close to the channel capacity, the product code can give a very low probability of error with only a minimal loss in overall rate.

Example 8.2.1. Let the column code be a binary $(64, 45, 8)$ code. With an error probability on the channel of $1/64$ we get a probability of decoding failure of less than 0.02. The row code can be chosen as a RS code over $\mathbb{F}_{2^{15}}$ with $I = 3$. With an average of less than 625 errors in each row, we can get a very reliable decoding with a code correcting 800 errors. Thus the rate of the row code is still 0.95.

8.2.2 Inner convolutional codes

The concatenated codes used in practice often have a convolutional code as the inner code. If the binary images of the I interleaved codes are written as an mI by N array, the array is encoded by the convolutional code one column at a time. The purpose of the interleaving, other than making the code longer, is to make the symbol errors in each RS code almost independent. A precise analysis of such a code is difficult.

However, we can get a very similar code using a tail-biting version of the convolutional code as the inner code for each column. The advantage from the point of view of the analysis is that the symbol errors in a particular outer code are mutually independent.

Let p be the bit error probability. For a given inner code and decoding method (which would usually be maximum likelihood decoding), the probability of decoding error for the inner code, p_{i}, can be found by one of the methods of Chapter 3.

The average number of symbol errors in a block of the RS code is $\mu = Np_{\text{i}}$. Since this is typically a small fraction of the block length and N is quite large, the Poisson distribution

$$P(t) = e^{\mu} \frac{\mu^t}{t!}$$

is a good approximation to the number of errors. We get a simple upper bound on the tail of the Poisson distribution by using a quotient series with quotient $\frac{\mu}{T+1}$.

Lemma 8.2.3. *If errors occur independently with probability p the probability that there are more than T errors in a block of N symbols is upper bounded by*

$$\Psi(p,T) = \frac{e^{-Np}(Np)^{T+1}}{(T+1)!\left(1-\frac{Np}{(T+1)}\right)}.$$

Thus we have

Theorem 8.2.2. *The probability of decoding failure for the concatenated code with an outer* RS *code correcting T errors and inner code decoding error probability p_{i} is upper bounded by*

$$P_{\text{fail}} < \Psi(p_{\text{i}}, T).$$

Actual decoding errors are usually quite rare. The following theorem is a useful approximation.

Theorem 8.2.3. *The probability of decoding error for the concatenated code is approximately*

$$P_{\text{err}} \approx \frac{P_{\text{fail}}}{T!}$$

Proof. Decoding of $T = \frac{D-1}{2}$ errors maps only a small fraction of the q^{N-K} syndromes on error patterns. Since there are about $\frac{N^T}{T!}$ sets of error positions and q^T sets of error values, the probability that a random syndrome corresponds to an error pattern is $\frac{1}{T!}$. □

We gave this type of approximation as (3.13). Actually a more detailed calculation like (3.12) shows that the approximation is very close.

In tail-biting codes (and convolutional codes), there is a length associated with an error event and a probability distribution on the lengths. What is relevant in this context is that the performance of the inner code may be characterized by probabilities $p_{\text{i}1}, p_{\text{i}2}, \ldots, p_{\text{i}I}$ that $1, 2, \ldots, I$ consecutive outer code symbols in a particular column are in error. Since a single long error event is more likely than two shorter events, we assume

$$p_{\text{i}j} > p_{\text{i}1}\, p_{\text{i},j-1}.$$

Clearly it may also occur that two non-consecutive symbols in the same column are in error, but since this would be the result of two independent decoding errors, we may treat this situation as errors in different columns. The probability that there is an error in a column is

$$p_{\text{c}} = \sum_j p_{\text{i}j}$$

The average symbol error probability may be found as

$$p_{\mathrm{s}} = \frac{1}{I}\sum_j j p_{\mathrm{i}j}$$

Clearly the probability of decoding failure is upper bounded by $\Psi(p_{\mathrm{c}}, T)$, which follows from using Theorem 8.2.1 with p_{c} as the probability of error for the inner code. However, we can get a stronger bound.

Theorem 8.2.4. *For a concatenated code with an inner convolutional code where the symbol error probability after decoding the inner code is* p_{s}, *the probability of failure is upper bounded as*

$$P_{\mathrm{fail}} < I\Psi(p_{\mathrm{s}}, T).$$

Proof. For each RS code the probability of decoding failure is $\Psi(p_{\mathrm{s}}, T)$. Thus for I interleaved codes the average number of RS codewords that are not decoded is I times this number. The errors in different RS words within a frame are not independent, but the theorem is true for any distribution of the words. □

The actual performance is a little better since longer error events in the inner code tends to cause symbol errors in different RS codes.

As discussed in Chapter 6, a construction with an outer RS codes can be used to protect a frame. If this point of view is adopted, the probability of failure is referred to as P_{fe} and the error probability as P_{ue}.

Even though several of the outer codewords cannot be decoded, there is often at least one decoded RS word. Since this word has a very high probability of being correct, it would be possible to iterate the decoding of the inner and outer codes. In the positions where errors have been corrected in the RS words, the decoding of the inner code should be repeated with the corrected symbols. However, it is difficult to analyze the improvement in performance of such a procedure.

8.3 Graph codes

It is possible to construct longer codes with better minimum distances from short component codes. Here we use graphs to describe how the component codes are connected.

8.3.1 Graphs and their adjacency matrices

A *graph* is defined by a set of vertices and a set of edges connecting pairs of vertices. We assume that no edge connects a vertex to itself, and that there are no multiple edges between any two vertices. We also assume that the graph is connected, i.e., starting from any vertex, we can go to a vertex connected to it by an edge (we refer to that as a step), and in several steps we can reach any other vertex. For our purpose it is useful that all vertices can be reached in few steps; we refer to a graph with this property as a *good expander*.

We call a graph $G = (V, E)$ *bipartite* if its vertex set can be written as $V = V_1 \cup V_2$ with $V_1 \cap V_2 = \varnothing$ and $|V_1| = |V_2| = m$. The graph is n*-regular* if each vertex of V_1 is connected to n vertices of V_2, and each vertex of V_2 is connected to n vertices of V_1. Let $x_1, x_2, \dots, x_m$ be the vertices in V_1 and $y_1, y_2, \dots, y_m$ the vertices in V_2 and define the $m \times m$ matrix $M = m_{ij}$ by

$$m_{ij} = \begin{cases} 1, & \text{if } x_i \text{ is connected to } y_j, \\ 0, & \text{otherwise.} \end{cases}$$

The *adjacency matrix* of the bipartite graph is

$$A = \begin{pmatrix} 0 & M \\ M^T & 0 \end{pmatrix}.$$

Thus each row has n 1s, the largest eigenvalue of A is n, and the corresponding eigenvector is the all 1s vector. Similarly $-n$ is an eigenvalue of A, and the corresponding eigenvector has 1s in the first half of the positions and -1 in the rest. It is known that for a connected graph $-n \leq \lambda_i \leq n$, where λ_i is any eigenvalue, and that the second largest eigenvalue λ_2 is closely related to what is known as the expansion property of the graph: A small value of λ_2 indicates that a large part of the graph can be reached in a few steps from most starting vertices.

8.3.2 Codes on graphs

We now construct a codes based on bipartite graphs. Let C_1 be a linear (n, k_1, d_1) code and C_2 a linear (n, k_2, d_2) code, both over the finite field $\mathbb{F}_q$. We now construct a code C of length $L = mn$ over $\mathbb{F}_q$ by associating $\mathbb{F}_q$ symbols with the edges of the graph (with a selected numbering) and demanding that the symbols connected to a vertex of V_1 (in the chosen order) shall be a codeword of C_1 and that the symbols on the edges connected to a vertex of V_2 (in the chosen order) shall be a codeword of C_2. It is clear that C is a linear code. Let K be its dimension.

Lemma 8.3.1. *The rate* $R = \frac{K}{L}$ *of* C *satisfies*

$$R \geq r_1 + r_2 - 1 \quad \textit{where} \quad r_1 = \frac{k_1}{n} \quad \textit{and} \quad r_2 = \frac{k_2}{n}. \tag{8.3}$$

Proof. The number of linearly independent parity checks is at most $m(n - k_1) + m(n - k_2)$, so $L - K \leq m(n - k_1) + m(n - k_2)$ and since $L = mn$, we get the result. □

If $d_1 = d_2 = d$ we have the following lower bound on the minimum distance, D, of C:

Theorem 8.3.1. *The minimum distance of C satisfies*

$$D \geq dm\frac{d - \lambda_2}{n - \lambda_2}, \tag{8.4}$$

where λ_2 is the second largest eigenvalue of the adjacency matrix.

Proof. Consider a minimum-weight codeword with nonzero component codewords associated with a set of vertices, V_z, consisting of a vertices on one side (say, V_1) and $b \leq a$ on the other side. Thus there are at least d edges connecting each of the vertices in V_z on the left with vertices in V_z on the right. We consider a vector which has $2a$ entries of $m-a$ in positions that correspond to V_z and entries $-a$ in the remaining positions. Thus this vector is orthogonal to the all 1s vector. In Section A.2 we derive the bound

$$v^T Av \leq \lambda_2 |v|^2. \tag{8.5}$$

Here λ_2 is the second largest eigenvalue. The left side is the sum over all branches multiplied by the values of v in the two endpoints. Since the two halves of v are the same, we consider just one side. There are at least ad branches connecting vertices in V_z, and thus at most $2a(n-d)$ connecting vertices in V_z with vertices in $V - V_z$. Finally there are at least $(m-a)n - a(n-d)$ branches with both endpoints in $V - V_z$. Now a direct calculation of both sides of (10.4) gives

$$am^2d - a^2mn \leq \lambda_2 am(m-a) \tag{8.6}$$

and the theorem follows, since $D \geq da$. □

We also note that in the case where the bipartite graph is complete, and hence $n = m$ and $\lambda_2 = 0$, we get the usual bound for product codes.

For short component codes, where $d \leq \lambda_2$, the bound is not useful, but we can get a simple lower bound by the following consideration: any vertex corresponding to a nonzero codeword on the right set is incident with at least d nonzero edges connecting to vertices in the left set, and these reach at least $d(d-1)$ vertices in the right set with nonzero edges. If the graph has no cycles of length 4, these vertices are distinct, and the minimum distance is always bounded from below as

$$D \geq d(d(d-1)+1) = d(d^2 - d + 1). \tag{8.7}$$

The potential of graph codes is related to the possibility of keeping the component code fixed while the size of the graph increases. In this way the performance can be improved with only a linear increase in decoding complexity. However, for the code C to have a reasonable rate, the component codes must have a high rate, and to get a positive bound from Theorem 8.3.1, the minimum distance of the component codes has to be larger than λ_2. The combination of these requirements tends to make the resulting code too long for any realistic applications. Thus our emphasis in this section is to improve the analysis of codes of moderate block length derived from specific good graphs. The use of RS component codes also allows a combination of good rates and distances for moderate code lengths.

8.3.3 RS codes on planes

In Chapter 2 we described planes over finite fields. We may derive bipartite graphs from such planes by associating the left set of vertices with the points of the plane and the right vertices with the lines. A line vertex has an edge connecting it to a point vertex when the point is on the line.

For the affine plane we label the points (x, y) and the lines (a, b), where a point is on a line whenever $y = ax + b$. For our construction we eliminate the lines $x = k$, which makes the graph q-regular and the adjacency matrix symmetric in the two sets of nodes. Similarly, we can derive a bipartite graph from the projective plane by using the labels $(x : y : z)$ for the left vertices and $(a : b : c)$ on the right. There is then an edge whenever $ax + by + cz = 0$.

We define a code on the affine plane graph by associating a symbol from $\mathbb{F}_q$ with each vertex and requiring that the symbols connecting to a particular vertex (whether on the left or right) form a codeword of the component (q, k, d) Reed–Solomon code, when the symbols are read in a fixed order.

Theorem 8.3.2. *We obtain codewords of a graph code on an affine $\mathbb{F}_q$ plane with* RS (q, k) *component codes over the same field by evaluating all polynomials $f(x, y, a, b) \in \mathbb{F}_q[x, y, a, b]$ of degree $< k$ in a, b and x, y.*

Proof. Consider a vertex on the left with label (x', y'). The lines through this point satisfy $b = -ax' + y'$, and substituting this relation into the polynomial gives a polynomial in a of degree k. Thus the values are a codeword in a (q, k) RS code when the code positions are numbered by the values of a. Similarly, any vertex on the right gives a codeword in the component code as an evaluation of a polynomial in x. □

Theorem 8.3.3. *Let K be the dimension of the graph code with (n, k) component codes, $k/n < 1/2$. The dimension K' for the dual $(n, n-k)$ component codes is*

$$K' = L - 2mk + K. \tag{8.8}$$

Proof. As in the previous lower bound on the dimension, the term $2mk$ is the total number of parity checks in the component codes. If these are all linearly independent, K' equals the lower bound, Lemma 8.3.1, and it is 0 for $k/n > 1/2$. However, if some of the checks spanned by the equations on the right and left are the same, we get a codeword in the low-rate code, and similarly the dimension of the high rate code is increased. □

Theorem 8.3.4. *The dimension of the graph code is $K \geq k^3$ for $k/n \leq 1/2$. For higher rates the dimension follows from the Theorem 8.3.3.*

Proof. The number of monomials of degree less than k in x, y and in a, b is $(k(k+1)/2)^2$. Since polynomials that are multiples of $y - ax - b$ evaluate to zero, we have to subtract the dimension of these multiples, which is similarly $(k(k-1)/2)^2$. The result now follows. □

In order to get lower bounds on these graph codes, we need the eigenvalues of the adjacency matrices. The argument is easier for the projective plane: In this case $m = q^2 + q + 1$ and $n = q + 1$. Now

$$A^2 = \begin{pmatrix} MM^T & 0 \\ 0 & M^TM \end{pmatrix}.$$

It is easy to see that

$$MM^T = M^TM = \begin{pmatrix} q+1 & 1 & \dots & 1 \\ 1 & q+1 & \dots & 1 \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & q+1 \end{pmatrix},$$

from which it follows that A^2 has eigenvalues n^2 and q, and therefore A has eigenvalues $\pm n$ with multiplicity 1 and $\pm\sqrt{q}$ each with multiplicity $m - 1$.

Theorem 8.3.5. *The minimum distance D of the projective plane code satisfies*

$$D \geq (q^2 + q + 1)d(d - \sqrt{q})/(n - \sqrt{q}). \tag{8.9}$$

For codes based on affine plane graphs we get a similar result with the same eigenvalue. The argument is slightly more complicated, since the matrix has eigenvalue 0 with multiplicity $q - 1$, while the rest are as in the projective case.

Theorem 8.3.6. *The minimum distance D of the affine plane code satisfies*

$$D \geq q^2 d(d - \sqrt{q})/(n - \sqrt(q)). \tag{8.10}$$

Example 8.3.1. Consider the code based on an affine plane and a component RS code over $\mathbb{F}_{32}$. Thus then length of the code is $32^3 = 32768$. If we take the parameters of the component code to be $(32, 25, 8)$, we get a minimum distance of at least

$$D \geq 32^2 \times 8 \times \frac{8 - \sqrt{32}}{32 - \sqrt{32}} \simeq 728.$$

The dimension of the code becomes

$$K = 32^2 \times 18 + 7^3 = 18775.$$

8.4 Problems

Problem 8.4.1. Consider a product of $(7, 4, 3)$ Hamming codes.

(1) What are the parameters of the code?

(2) How many errors can be corrected if the rows and columns are decoded serially?

(3) How many minimum weight words are there in the product code?

(4) Describe a parity check matrix for the product code (without necessarily writing it out).

(5) How can three errors be decoded if they are

  a) in three different rows or columns?

  b) in only two rows and columns?

Problem 8.4.2. Consider a product of $(16, 12)$ RS codes over $\mathbb{F}_{16}$.

(1) What are the parameters of the code?

(2) How many errors can be corrected if we assume minimum distance decoding?

(3) How many errors can be decoded if rows and columns are decoded serially?

Problem 8.4.3.

(1) Show that there is a cyclic binary $(21, 16, 3)$ code with generator polynomial $x^5 + x + 1$.

(2) Use this code as the component code in a product code of length $N = 441$. Find K and D.

(3) Consider the lower right 6 by 6 subarray (corresponding to the low degree terms in the code polynomials). Show that it contains a codeword of weight D with 3 nonzero rows and columns.

(4) Consider the lower right 7 by 7 subarray. Show that it contains a codeword of weight 16 with 5 nonzero rows and columns.

(5) Find a lower bound on the number of errors that can be corrected in square subarrays of sizes 3 to 7.

(6) Find a correctable error pattern of weight 7 among the nonzero positions of the weight 16 codeword found previously.

(7) Apply serial decoding. Are the errors corrected? If not, repeat the decoding of rows and columns.

Problem 8.4.4. Consider a product of the binary $(5, 4)$ and $(7, 6)$ even parity codes.

(1) What are the parameters of the code?

(2) Prove that the code is cyclic if the positions are taken in the proper order.

(3) Find the generator polynomial of the cyclic code.

Problem 8.4.5. Consider the binary image of a $(15, 10)$ RS code over $\mathbb{F}_{16}$.

(1) What are the parameters of the code?

(2) Change the code to a concatenated code with inner code $(5, 4, 2)$. What are the parameters?

(3) Change the code to interleaving degree 2 using the $(9, 8, 2)$ code as inner code. What are the parameters?

Problem 8.4.6. Consider a $(30, 20)$ RS code.

(1) If the symbol error probability is $\frac{1}{30}$, what is the probability of decoding failure?

(2) Under the same assumptions, what is the probability of decoding error?

(3) The code is used as the outer code in a concatenated code with $I = 2$ and inner $(14, 10, 3)$ code. If the bit error probability is 0.01, give an upper bound on the symbol error probability.

(4) Give an upper bound on the probability of decoding failure for the concatenated code.

Problem 8.4.7. Let $\mathbb{F}_{16}$ be expressed as in Chapter 2, Example 2.3.3.

(1) Find the coefficients of the generator polynomial for an RS code of length 15 correcting two errors.

(2) Which binary code is contained in the RS code?

(3) Let the generator matrix of a $(7, 4)$ Hamming code be as in Example 1.1.3. What are the codewords representing the coefficients found in (1)?

(4) Consider the concatenated code obtained by encoding the symbols of the RS code from question (1) using the code from (3). What are the parameters of the code? (You may use the result of (2) to find the exact minimum distance).

(5) Find a codeword in the concatenated code by taking the generator polynomial of the RS code and encoding the symbols as in (4). Introduce seven bit errors in the nonzero symbols, and decode the word.

Problem 8.4.8. *Project.*
Use the interleaved RS code from Problem 8.4.6 and use the convolutional code from Problem 7.7.1 as the inner code. Compare decoding results for a single inner code and for a tail-biting code in each column.

Problem 8.4.9. *Project.*
Write a decoding program for a $(255, 239)$ RS code. Use this decoder for a concatenated code with the $(2, 1, 6)$ convolutional code from Problem 7.7.5 as inner code. Perform a simulation with 2% bit errors.

Problem 8.4.10. Construct a graph code on the affine plane over $\mathbb{F}_4$ using component RS codes $(4, k)$.

(1) What is the length of the codes?

(2) Give bounds for the dimensions and minimum distances.

Chapter 9

Decoding Reed–Solomon and BCH Codes

In Chapter 4 we presented a decoding algorithm for RS and BCH codes consisting of the following parts:

- Syndrome calculation
- Solving for the error-locator polynomial
- Finding the error positions
- Calculating error values

Here we present some approaches to these calculations which are important for efficient implementations of decoders. If the decoding is implemented in software, the complexity is closely related to the total number of operations in the finite field. However, in many decoders implemented as integrated circuits or gate arrays, the important parameter is mostly the number of steps in the calculations whereas it is less critical that there are several parallel streams at each step. Throughout this chapter we assume that the BCH codes are binary and that the RS codes use the field $\mathbb{F}_{2^m}$.

9.1 Syndrome calculation

The syndrome calculation is conceptually easy: for a received word $r(x)$ and each root α^i of the generating polynomial find $S_i = r(\alpha^i)$. Clearly the calculation can be performed in parallel for the $n-k$ syndromes, each calculation using about n additions and multiplications. The calculation is often performed as

$$s^{(1)} = r_{n-1}.$$

For $j = 2$ to n:

$$\begin{aligned} s^{(j)} &= s^{(j-1)}\alpha^i + r_{n-j} \\ S_i &= s^{(n)} \end{aligned}$$

This approach requires little storage, and it is convenient if the symbols are received one at a time. However, it may be desirable to perform the calculation is fewer steps. The following methods are based on a simple observation.

Lemma 9.1.1. *If α^i is a root of $f(x)$ and*

$$r(x) = f(x)\phi(x) + \rho(x),$$

then

$$r(\alpha^i) = \rho(\alpha^i).$$

Thus rather than evaluating $r(x)$ directly, we can divide it by the minimal binary polynomial $m_i(x) = m_{\alpha^i}$, and find the syndromes $S_i, S_{2i}, \dots$ by evaluating the remainder. The division by a binary polynomial of degree m can be performed l bits at a time by taking the coefficients of r in segments of l coefficients,

$$r^{(s)} = [r_{sl-1}, r_{sl-2}, \dots, r_{(s-1)l}]$$

(if necessary adding some zeros as the leading coefficients). In each step we find a length l remainder ρ^j, and the multiplication is carried out by multiplying by a binary l by m matrix M_i, where row u of M_i is found as x^{m+l-u} modulo m_i,

$$\begin{aligned}\rho^{(1)} &= r^{(1)},\\ \rho^{(j)} &= \rho^{(j-1)}M_i + r^{(j)},\\ \rho &= \rho^{(n/l)} \mod m_i.\end{aligned}$$

Thus each step in the computation involves some additions in $\mathbb{F}_q$, but no multiplications, and the number of steps is reduced by a factor l.

We can also speed up the syndrome calculation by combining Lemma 9.1.1 with Example 2.4.1. If l is a factor of n, $x^n - 1$ factors into polynomials of the form $x^{n/l} - \beta^i$, and finding the remainders with respect to these polynomials requires only nl^2 multiplications and additions. Thus when l is small, this gives a convenient way of reducing the size of the remainders before the syndromes are evaluated. If n has several small factors, we can repeat the process on each remainder (this is the idea that reduces the number of operations in the Fast Fourier Transform to $n\log(n)$).

Example 9.1.1. A $(15, 5, 7)$ binary code.
Let α be a primitive element of $\mathbb{F}_{16}$, where $\alpha^4 + \alpha + 1 = 0$. Let the minimal polynomials be written as $m_i = m_{\alpha^i}$, and let C be the cyclic code with generator polynomial

$$\begin{aligned}g(x) &= m_1(x)m_3(x)m_5(x)\\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)\\ &= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1.\end{aligned}$$

So C is a $(15, 5, 7)$ code.

Let the received word be $r(x) = x^{13} + x^{11} + x^{10} + x^7 + x^4 + x^3$. We first divide r by the three minimal polynomials, m_1, m_3, m_5. Taking $l = 4$, we can divide r by each of the minimal polynomials using the matrices

$$M_1 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

$$M_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix},$$

$$M_5 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$

We extend r by a leading 0, and read the first 4 bits as $\rho^{(1)} = [0, 0, 1, 0]$. After that the division gives

$$\rho^{(2)} = [1, 0, 1, 0], \quad \rho^{(3)} = [0, 1, 0, 0], \quad \rho^{(4)} = [0, 1, 0, 0],$$

The remainder x^2 can then be evaluated to give S_1 (and S_2, S_4). Similarly, we divide by m_3 to get

$$\rho^{(2)} = [1, 1, 0, 1], \quad \rho^{(3)} = [0, 0, 0, 0], \quad \rho^{(4)} = [1, 0, 0, 0],$$

which gives S_3 (and S_6). Finally from m_5 we get

$$\rho^{(2)} = [1, 0, 0, 0], \quad \rho^{(3)} = [0, 1, 0, 1], \quad \rho^{(4)} = [1, 1, 0, 0].$$

The syndromes are

$$\begin{aligned} S_1 &= r(\alpha) = \alpha^2, \\ S_2 &= {S_1}^2 = \alpha^4, \\ S_3 &= r(\alpha^3) = \alpha^9, \\ S_4 &= {S_2}^2 = \alpha^8, \\ S_5 &= r(\alpha^5) = 1 + \alpha^{10} = \alpha^5, \\ S_6 &= {S_3}^2 = \alpha^3. \end{aligned}$$

The alternative approach to syndrome calculation uses the factorization

$$x^{15} - 1 = (x^5 - 1)(x^5 - \alpha^5)(x^5 - \alpha^{10}).$$

Segmenting r into $r^{(1)} = [0,1,0,1,1]$, $r^{(2)} = [0,0,1,0,0]$, $r^{(3)} = [1,1,0,0,0]$, we find the first remainder as the sum $r^{(1)} + r^{(2)} + r^{(3)} = [1,0,1,1,1]$, which can be evaluated to give S_3, S_6.

Example 9.1.2. A $(15,9,7)$ Reed–Solomon code over $\mathbb{F}_{16}$.
Let α be a primitive element of $\mathbb{F}_{16}$, where $\alpha^4 + \alpha + 1 = 0$, and let C be the $(15,9,7)$ Reed–Solomon code over $\mathbb{F}_{16}$ obtained by evaluation of polynomials of degree at most 8 in $\alpha^0, \alpha, \ldots, \alpha^{14}$. Then C has generator polynomial

$$\begin{aligned} g(x) &= (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6) \\ &= x^6 + \alpha^{10}x^5 + \alpha^{14}x^4 + \alpha^4x^3 + \alpha^6x^2 + \alpha^9x + \alpha^6. \end{aligned}$$

Let $r(x) = x^8 + \alpha^{14}x^6 + \alpha^4x^5 + \alpha^9x^3 + \alpha^6x^2 + \alpha$ be a received word.
We first divide r by the three minimal polynomials, m_1, m_3, m_5. Taking $l = 4$, and using the same matrices as in the previous example, we get the remainders

$$\begin{array}{llll} \rho^{(1)} = 0, & \rho^{(2)} = [0,0,0,1], & \rho^{(3)} = [0,\alpha^{14},\alpha,1], & \rho^{(4)} = [\alpha^4,\alpha^{10},\alpha^4,\alpha^4], \\ \rho^{(1)} = 0, & \rho^{(2)} = [0,0,0,1], & \rho^{(3)} = [1,\alpha^3,\alpha,1], & \rho^{(4)} = [\alpha^7,\alpha^6,\alpha^{14},1], \\ \rho^{(1)} = 0, & \rho^{(2)} = [0,0,0,1], & \rho^{(3)} = [1,\alpha^3,\alpha^4,0], & \rho^{(4)} = [0,\alpha^2,0,\alpha]. \end{array}$$

The syndromes are found by evaluating the first remainder in $\alpha, \alpha^2, \alpha^4$, the second in α^3, α^6, and the last in α^5. We get

$$\begin{aligned} S_1 &= \alpha^7 + \alpha^{12} + \alpha^5 + \alpha^4 = 1, \\ S_2 &= \alpha^{10} + \alpha^{14} + \alpha^6 + \alpha^4 = 1, \\ S_3 &= \alpha + \alpha^{12} + \alpha^2 + 1 = \alpha^3, \\ S_4 &= \alpha + \alpha^3 + \alpha^8 + \alpha^4 = \alpha^6, \\ S_5 &= \alpha^{12} + \alpha = \alpha^{13}, \\ S_6 &= \alpha^{10} + \alpha^3 + \alpha^5 + 1 = \alpha^3. \end{aligned}$$

We therefore have $S(x) = x^5 + x^4 + \alpha^3x^3 + \alpha^6x^2 + \alpha^{13}x + \alpha^3$.

9.2 The Euclidean algorithm

Let F be a field and let $a(x), b(x) \in F[x]$, and suppose that $\deg(a(x)) \geq \deg(b(x))$.

The Euclidean algorithm is used to determine a greatest common divisor, $d(x)$, of $a(x)$ and $b(x)$, and the extended version also produces two polynomials $f(x)$ and $g(x)$ such that

$$f(x)a(x) + g(x)b(x) = d(x).$$

The input to the algorithm is $a(x)$ and $b(x)$.

The algorithm operates with four sequences of polynomials, $r_i(x)$, $q_i(x)$, $f_i(x)$ and $g_i(x)$.

The initialization is

$$r_{-1}(x) = a(x), \quad f_{-1}(x) = 1, \quad g_{-1}(x) = 0$$

and

$$r_0(x) = b(x), \quad f_0(x) = 0, \quad g_0(x) = 1.$$

For $i \geq 1$ the algorithm performs polynomial division of r_{i-2} by r_{i-1} to obtain the quotient q_i and the new remainder r_i (see Section 2.3):

$$r_{i-2}(x) = q_i(x)r_{i-1}(x) + r_i(x), \text{ where } \deg(r_i(x)) < \deg(r_{i-1}(x)),$$

and then we update f and g as

$$f_i(x) = f_{i-2}(x) - q_i(x)f_{i-1}(x)$$

and

$$g_i(x) = g_{i-2}(x) - q_i(x)g_{i-1}(x).$$

Since the degrees of the $r_i(x)$ decrease, there is a *last* step, n say, where $r_n(x) \neq 0$. We then have that $\gcd(a(x), b(x)) = r_n(x)$ and $f_n(x)a(x) + g_n(x)b(x) = r_n(x)$. In the application to decoding, however, we are actually interested not in the gcd as such, but in some of the intermediate results.

Example 9.2.1. Let $a(x) = x^8$ and $b(x) = x^6 + x^4 + x^2 + x + 1$ be binary polynomials.
The algorithm gives:

i	f_i	g_i	r_i	q_i
-1	1	0	x^8	–
0	0	1	$x^6+x^4+x^2+x+1$	–
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	–	–	0	$x+1$

From this we see that $\gcd(x^8, x^6+x^4+x^2+1) = 1$ and that $(x^5+x^4+x^3+x^2)x^8 + (x^7 + x^6 + x^3 + x + 1)(x^6 + x^4 + x^2 + x + 1) = 1$.

That the algorithm works is based on the following,

Lemma 9.2.1. *For all $i \geq -1$:*

1. $\gcd(r_{i-1}(x), r_i(x)) = \gcd(r_i(x), r_{i+1}(x))$
2. $f_i(x)a(x) + g_i(x)b(x) = r_i(x)$
3. $\deg(g_i(x)) + \deg(r_{i-1}(x)) = \deg(a(x))$

Proof. The statements can be proven by induction. The first two are easy; we will do the induction step in 3.

Suppose $\deg(g_i(x)) + \deg(r_{i-1}(x)) = \deg(a(x))$. Then

$$\begin{aligned}
& \deg(g_{i+1}(x)) + \deg(r_i(x)) \\
= \; & \deg(g_{i+1}(x) + (\deg(r_{i-1}(x) - \deg(q_{i+1}(x)) \\
= \; & \deg(g_{i+1}(x) + ((\deg(a(x) - \deg(g_i(x))) - \deg(q_{i+1}(x)) \\
= \; & \deg(a(x)) + (\deg(g_{i+1}(x) - \deg(g_i(x)) - \deg(q_{i+1}(x)) \\
= \; & \deg(a(x)).
\end{aligned}$$
□

9.3 Reed–Solomon and BCH decoding with the Euclidian algorithm

Let C be a cyclic (n, k) code over $\mathbb{F}_q$ and suppose that the generator polynomial has among its zeroes the elements $\beta, \beta^2, \ldots, \beta^{2t}$, where $\beta \in \mathbb{F}_{q^m}$ is an element of order n.

For the RS codes we have $n|q-1$ and $t = \left\lfloor \frac{n-k}{2} \right\rfloor$ and for the binary BCH codes we have $n|2^m - 1$ and $d_{\min} \geq 2t + 1$.

We know from Chapters 4 and 5 that if the error locator polynomial, $Q(x) = q_0 + q_1 x + q_2 x^2 + \cdots + q_t x^t$ satisfies

$$\begin{bmatrix} S_1 & S_2 & \ldots & S_{t+1} \\ S_2 & S_3 & \ldots & S_{t+2} \\ \vdots & \vdots & \ldots & \vdots \\ S_t & S_{t+1} & \ldots & S_{2t} \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \vdots \\ q_t \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad (9.1)$$

where the syndromes S_i are defined as $S_i = r\left(\beta^i\right)$ when $r(x)$ is the received word, then the error locations (i.e., the powers of β that indicate the positions of the errors) are among the zeroes of $Q(x)$.

If we define

$$S(x) = \sum_{i=1}^{2t} S_i x^{2t-i}$$

and calculate $S(x)Q(x)$, then (9.1) gives that the terms of degree $t, t+1, \ldots, 2t-1$ in $S(x)Q(x)$ are all zero and therefore $\deg\left(S(x)Q(x) \mod x^{2t}\right) < t$.

On the other hand, if $Q(x)$, with $\deg(Q(x)) \leq t$, satifies that

$$\deg\left(S(X)Q(x) \mod x^{2t}\right) < t$$

then we have a solution to (9.1).

Theorem 9.3.1. *If the Euclidean algorithm is used on input* $a(x) = x^{2t}$ *and* $b(x) = S(x)$ *and* j *is chosen such that* $\deg(r_{j-1}(x)) \geq t$ *and* $\deg(r_j(x)) < t$, *then* $Q(x) = g_j(x)$.

Proof. It follows from Lemma 9.2.1 3. that $\deg(g_j(x)) = 2t - \deg(r_{j-1}(x)) \leq t$. By the definition of j and Lemma 9.2.1 2. $f_j(x)x^{2t} + g_j(x)S(x) = r_j(x)$. Since $\deg(r_j(x)) < t$, we then get that $\deg\left(g_j(x)S(x) \mod x^{2t}\right) < t$ and therefore the theorem is proved. □

Example 9.3.1. A $(15, 5, 7)$ binary code (continued).
We found $S(x) = \alpha^2x^5 + \alpha^4x^4 + \alpha^9x^3 + \alpha^8x^2 + \alpha^5x + \alpha^3$.
The Euclidean algorithm on x^6 and $S(x)$ gives:

i	g_i	r_i	q_i
-1	0	x^6	–
0	1	$\alpha^2x^5+\alpha^4x^4+\alpha^9x^3+\alpha^8x^2+\alpha^5x+\alpha^3$	–
1	$\alpha^{13}x+1$	$\alpha^3x^4+\alpha^5x^3+\alpha^{13}x^2+\alpha^2x+\alpha^3$	$\alpha^{13}x+1$
2	$\alpha^{12}x^2+\alpha^{14}x+1$	$\alpha^8x^3+\alpha^{10}x^2+\alpha x+\alpha^3$	$\alpha^{14}x$
3	$\alpha^7x^3+\alpha^9x^2+\alpha^9x+1$	$\alpha^7x^2+\alpha^{14}x+\alpha^3$	$\alpha^{10}x$

From this we see that $j = 3$ and $g_3(x) = \alpha^7x^3 + \alpha^9x^2 + \alpha^9x + 1$, which has α^5, α^8 and α^{10} as zeroes, so the error vector is $x^{10} + x^8 + x^5$ and the codeword is therefore $x^{13} + x^{11} + x^8 + x^7 + x^5 + x^4 + x^3$ $(= x^3g(x))$.

Example 9.3.2. A $(15, 9, 7)$ Reed–Solomon code over $\mathbb{F}_{16}$ (continued).
We found $S(x) = x^5 + x^4 + \alpha^3x^3 + \alpha^6x^2 + \alpha^{13}x + \alpha^3$.
The Euclidean algorithm on x^6 and $S(x)$ gives:

i	g_i	r_i	q_i
-1	0	x^6	–
0	1	$x^5+x^4+\alpha^3x^3+\alpha^6x^2+\alpha^{13}x+\alpha^3$	–
1	$x+1$	$\alpha^{14}x^4+\alpha^2x^3+x^2+\alpha^8x+\alpha^3$	$x+1$
2	$\alpha^2x^2+\alpha^4x$	$\alpha^{11}x^3+\alpha^{10}x^2+\alpha^7x$	$\alpha x+1$
3	$\alpha^4x^3+\alpha^3x^2+\alpha^9x+1$	$\alpha^7x^2+\alpha x+\alpha^3$	$\alpha^3x+\alpha^3$

The polynomial $g_3(x)$ has as zeroes $1, \alpha^4$ and α^7, so the error polynomial is $e(x) = e_1 + e_2x^4 + e_3x^7$.

The implementation of Euclid's algorithm is simplified if the division of r_i and r_{i-1} is replaced by a step which just cancels one of the leading coefficients by scaling one of the polynomials and adding them. It may also be noted that we can save space by storing r_i and g_{i-1} in a shared array (and similarly r_{i-1} and g_i).

It is not necessary to store all coefficients of the remainders in Euclid's algorithm. Each step depends on only the leading coefficients, and these may be computed as they are needed. In this version the algorithm is equivalent to Berlekamp's algorithm, which is also a common choice for decoding BCH codes. Since the number of steps in this part of the decoding algorithm is at most $2t$ and the storage space is a small multiple of t, it is usually not a critical part of the decoder. In principle, it is possible to obtain faster decoders for sufficiently large values of t.

9.4 Finding the error positions

Once we have the error locator polynomial, $Q(x)$, we get the error positions as the roots of $Q(x)$. The common approach in the part of the decoding is to test all elements of $\mathbb{F}_q$. This calculation is similar to the syndrome calculation, and it requires in the order of tn operations. However, t is usually a relatively small number, and we can choose to test as many positions as we want in parallel.

For very small values of t an alternative approach is to use an algorithm for factoring Q. In Chapter 2 we discussed the solution of quadratic equations. Actually the same approach gives a simple way of factoring a polynomial in $\mathbb{F}_{2^m}$ when the only nonzero coefficients are those of x^{2^i}. Let $t = 4$ and consider the error locator $q_0 + q_1x + q_2x^2 + q_3x^3 + x^4$. We want to shift the roots of Q in such a way that q_3 becomes zero. If there are only three errors, we can multiply Q by $x + q_3$ and get a polynomial of the right form. If $q_1 = 0$, but not q_3, we can reverse the coefficients and find the inverse roots. Assume that a constant, u, is added to all 4 error locations, p_1, p_2, p_3, p_4. Then we get a polynomial of degree 4 with

$$q_3' = p_1 + u + p_2 + u + p_3 + u + p_4 + u = q_3.$$

Similarly, we find

$$\begin{aligned}
q_2' &= (p_1+u)(p_2+u)+(p_1+u)(p_3+u)+\cdots+(p_3+u)(p_4+u)\\
&= q_2+uq_3,\\
q_1' &= (p_1+u)(p_2+u)(p_3+u)+\cdots+(p_2+u)(p_3+u)(p_4+u)\\
&= q_1+u^2q_3,\\
q_0' &= (p_1+u)(p_2+u)(p_3+u)(p_4+u) = q_0+uq_1+u_2^q+u_3^q.
\end{aligned}$$

Here we can take u as the unique solution to $q_1 + u^2q_3 = 0$, to get the new polynomial with $q_1' = 0$. We can then reverse the polynomial, get the inverse roots, and add u to recover the original positions. Thus the computation is largely reduced to solving a small system of binary equations.

Example 9.4.1. As in the earlier example take $g_3(x) = \alpha^4x^3+\alpha^3x^2+\alpha^9x+1$. In order to get a zero coefficient for x^3 we multiply by $x+\alpha^{14}$ and find the polynomial $j(x) = \alpha^4x^4+\alpha^3x^2+\alpha^9x$, where for the time being we remove the constant factor. A polynomial where the only nonzero terms have powers of x that are powers of 2 is called 'linearized'. It has the property (which is easily verified directly) that $j(a+b) = j(a)+j(b)$. This also means that the set of roots form a linear subspace of the field. We can find the roots by evaluating j in $1, \alpha, \alpha^2, \alpha^3$ and solving the homogeneous binary equations

$$\begin{bmatrix} 1 & 0 & 1 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 1 & 0 & 1 & 0 \end{bmatrix} x = 0,$$

where the columns of the coefficient matrix are the values of j. Thus we find the solutions $[0,0,0,0], [0,1,0,0], [0,0,0,1], [0,1,0,1]$. Similarly, we find the roots of the polynomial with the constant term included by using this term on the right side, and we get the solutions $[1,0,0,0], [1,1,0,0], [1,0,0,1], 1,1,0,1$, or $1, \alpha^4, \alpha^{14}, \alpha^7$. Here α^{14} was introduced to facilitate the factorization, and the remaining values are the error positions we found earlier.

9.5 Calculation of error values

We recall the expression for the error values 4.10. We derive a simpler formula (due to Forney). Before doing that we will introduce a useful concept.

Definition 9.5.1. *Let $a(x) = a_nx^n+\cdots+a_1x+a_0$ be a polynomial in $\mathbb{F}_q[x]$.*

Then $a'(x) = na_nx^{n-1}+\cdots+2a_2x+a_1$, where the integers i in the coefficients are the sum of i 1s in the field.

With this definition it is easy to see that $(a(x)+b(x))' = a'(x)+b'(x)$ and $(a(x)b(x))' = a'(x)b(x)+a(x)b'(x)$.

Let the error positions be $x_{i_1}, x_{i_2}, \dots, x_{i_t}$ with the corresponding error values $e_{i_1}, e_{i_2}, \dots, e_{i_t}$.

Using the notation from Theorem 9.3.1 we will prove

Theorem 9.5.1.

$$e_{i_S} = -x_{i_S}^{-(2t+1)} \frac{r_j\left(x_{i_S}\right)}{g_j'\left(x_{i_S}\right)}. \tag{9.2}$$

Proof. From (4.10) we have that

$$e_{i_S} = \frac{\sum_{r=1}^{t} P_r^{(S)} S_r}{P^{(S)}(x_{i_S})},$$

where

$$P^{(S)}(x) = \sum_{r=1}^{t} P_r^{(S)} x^r = (-1)^{t-S} x \prod_{\substack{l=1 \\ l\neq S}}^{t} \left(x - x_{i_l}\right)$$

and

$$S_r = \sum_{j=1}^{t} e_{i_j} x_r^{i_j}.$$

From Theorem 9.3.1, we get

$$Q(x) = g_j(x) = \prod_{l=1}^{t} \left(x - x_{i_l}\right),$$

so

$$g_j'(x) = \sum_{S=1}^{t} \prod_{\substack{l=1 \\ l\neq S}}^{t} \left(x - x_{i_l}\right)$$

and hence

$$g_j'\left(x_{i_S}\right) = \prod_{\substack{l=1 \\ l\neq S}}^{t} \left(x - x_{i_l}\right) = x_{i_S}^{-1}(-1)^{(t-S)} P^{(S)}\left(x_{i_S}\right).$$

From Theorem 9.3.1 we see that

$$r_j(x) = g_j(x)S(x) \mod x^{2t},$$

and therefore

$$\begin{aligned}
r_j(x) &= \prod_{l=1}^{t}\left(x - x_{i_l}\right)\sum_{i=1}^{2t} S_i x^{2t-i} \mod x^{2t} \\
&= (-1)^{t-S} P^{(S)}(x) x^{-1}\left(x - x_{i_S}\right)\sum_{i=1}^{2t} S_i x^{2t-i} \mod x^{2t} \\
&= (-1)^{t-S}\left(x - x_{i_S}\right)\sum_{i=1}^{2t}\sum_{r=1}^{t} P_r^{(S)} x^{r-1} S_i x^{2t-i} \mod x^{2t}.
\end{aligned}$$

Therefore,

$$r_j\left(x_{i_S}\right) = -(-1)^{(t-S)}\sum_{r=1}^{t} x_{i_S}^{2t} P_r^{(S)} S_r$$

and the result follows. □

Example 9.5.1. The $(15, 9, 7)$ Reed–Solomon code over $\mathbb{F}_{16}$.
To find the error values one can solve the system of equations:

$$\begin{aligned}
S_1 &= 1 = e(\alpha) = e_1 + e_2\alpha^4 + e_3\alpha^7, \\
S_2 &= 1 = e(\alpha^2) = e_1 + e_2\alpha^8 + e_3\alpha^{14}, \\
S_3 &= \alpha^3 = e(\alpha^3) = e_1 + e_2\alpha^{12} + e_3\alpha^6.
\end{aligned}$$

The solution is $e_1 = \alpha, e_2 = \alpha^6$ and $e_3 = \alpha^{10}$, so we get $e(x) = \alpha^{10}x^7 + \alpha^6x^4 + \alpha$ and therefore $c(x) = r(x) + e(x) = x^8 + \alpha^{10}x^7 + \alpha^{14}x^6 + \alpha^4x^5 + \alpha^6x^4 + \alpha^9x^3 + \alpha^6x^2 (= x^2g(x))$.
We had $r_3(x) = \alpha^7x^2 + \alpha x + \alpha^3$ and $g_3(x) = \alpha^4x^3 + \alpha^3x^2 + \alpha^9x + 1$, so $g_3'(x) = \alpha^4x^2 + \alpha^9$.
We use Formula (9.2) and get:

$$\begin{aligned}
e_1 &= (\alpha)^{-7}\frac{\alpha^7 + \alpha + \alpha^3}{\alpha^4 + \alpha^9}\frac{1}{\alpha^{14}} = \alpha, \\
e_2 &= \left(\alpha^4\right)^{-7}\frac{1 + \alpha^5 + \alpha^3}{\alpha^{12} + \alpha^9} = \alpha^2 \cdot \frac{\alpha^12}{\alpha^8} = \alpha^6, \\
e_3 &= \left(\alpha^7\right)^{-7}\frac{\alpha^6 + \alpha^8 + \alpha^3}{\alpha^3 + \alpha^9} = \alpha^{11} \cdot \frac{1}{\alpha} = \alpha^{10},
\end{aligned}$$

in accordance with the previous result.

9.6 Problems

Problem 9.6.1. Let α be a primitive element of $\mathbb{F}_{16}$ with $\alpha^4 + \alpha + 1 = 0$.
Let C be the $(15, 9)$ Reed–Solomon code over $\mathbb{F}_{16}$ obtained by evaluating polynomials from $\mathbb{F}_{16}[x]$ of degree at most 8 in $\alpha^0, \alpha, \alpha^2, \ldots, \alpha^{14}$. The generator polynomial of this code is $g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^6)$.
Decode the received words below using the Euclidean algorithm and Formula (9.2).

(1) $r(x) = x^3 + \alpha^2 + \alpha x$.

(2) $r(x) = \alpha^2 x^3 + \alpha^7 x^2 + \alpha^{11} x + \alpha^6$.

Problem 9.6.2. We use the code C_{sub} from above and receive

(1) $x^8 + x^5 + x^2 + 1$

and

(2) $x^{11} + x^7 + x^3 + x^2 + x + 1$.

Decode these two words using the Euclidean Algorithm.

Problem 9.6.3. Let α be a primitive element of $\mathbb{F}_{16}$ with $\alpha^4 + \alpha + 1 = 0$.
Let C be the $(15, 7)$ Reed–Solomon code over $\mathbb{F}_{16}$ obtained by evaluating polynomials from $\mathbb{F}_{16}[x]$ of degree at most 6 in $\alpha^0, \alpha, \alpha^2, \ldots, \alpha^{14}$. The generator polynomial of this code is $g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^8)$.
Decode the received word below using the Euclidean algorithm and Formula (9.2).

$$\begin{aligned} r(x) = {} & x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + \alpha^3 x^9 + x^8 \\ & + x^7 + x^6 + x^5 + x^4 + x^3 + \alpha^2 x^2 + \alpha x + 1. \end{aligned}$$

Chapter 10

Iterative Decoding

We discuss decoding of composite codes by iterating local processing. The first sections are based on processing a symbol using only a few parity checks, whereas the last two sections discuss methods where we iterate the decoding of component codes.

10.1 Low density parity check codes

As discussed in Chapter 1, a block code is defined by its parity check matrix, H, and every row in H indicates a parity check that all words satisfy. A code has many equivalent parity check matrices, and it can have more than $n-k$ rows. Thus in this chapter we shall not require that the rows of H are linearly independent.

Definition 10.1.1. *A* low density parity check (LDPC) code *with parameters* (n, i, j) *has block length n and a sparse parity check matrix* $H = [h_{uv}]$. *Let* I_v *indicate the set of row indices in column* v *such that* $h_{uv} = 1$, *and* J_u *similarly the set of column indices such that* $h_{uv} = 1$. *Here* $|I_v| = i, |J_u| = j$, *and* $|J_{u'} \cap J_{u''}| \leq 1$.

The name suggests that there are few ones in H, i.e., n is much larger than i, j,

Lemma 10.1.1. *The rate of an* (n, i, j) LDPC code *satisfies*

$$R \geq 1 - \frac{i}{j}.$$

Proof. If H has b rows, the total number of 1s is $bj = ni$. These rows may not all be linearly independent, and thus the dimension k of the code satisfies

$$k \geq n - b = n - \frac{ni}{j}. \qquad \square$$

We adopt a geometric terminology, referring to the positions in the code as points and the rows of H as lines. Thus J_u is the set of points on line u. The definition requires that two different lines intersect in at most one point, and it follows that there is at most one line connecting two different points. As in our earlier discussion of geometries, we can represent the connections as a bipartite graph (the code graph), where n symbol nodes on the left represent points and have degree i, while b parity nodes on the right represent lines and have degree j.

Example 10.1.1. Decoding erasures with LDPC codes.
One may get a first impression of iterative decoding by considering decoding of erasures only (thus the assumption is that no actual errors occur). Decoding simply consists in solving the system of linear equations $Hr^T = 0$. We consider one parity check equation at a time. If there are more than one erasure in a line, we leave the positions unchanged. However, if there is only one erasure, we can fill in the symbol value. We repeat this process until all erasures are corrected or all remaining equations contain more than one erasure. The last situation causes a decoding failure, even though the remaining system of equations might have a unique solution.

We can decode an LDPC code in the following way: Consider the i parity checks in I_v. Each of these involves $j-1$ other received symbols, and the sum provides an estimate of the symbol at position v. From our assumptions, the $i(j-1)$ positions involved in these estimates are distinct. We take the decoded value of the symbol to equal the majority of the i estimates and the received symbol r_v. This approach is called *majority decoding*.

Lemma 10.1.2. *If at most $\frac{i}{2}$ errors occur among the $i(j-1)+1$ symbols involved in the majority decision, the position is correctly decoded.*

Proof. Since the symbols in the estimates are distinct, at most $i/2$ of $i+1$ can be in error, while at least $i/2+1$ are correct. □

In particular, we have

Lemma 10.1.3. *For a (n, i, j) LDPC code the minimum distance satisfies $d \geq i+1$.*

10.2 Bit flipping

In the previous section majority decoding was defined as a decision about each code symbol as a function of the received vector. Thus the decoding can be performed in parallel for all positions in a single step. In this section we modify this approach by applying a majority decision to a single position in each step. The decoded value is then substituted into the received vector before the next step is performed.

10.2.1 Generalized syndromes

As suggested earlier, it can be useful for iterative decoding to use a parity check matrix with more than $n - k$ rows, and we assume that the rows have been selected in such a way that all rows have weight j and all columns weight i. For a received word, r, we refer to Hr^T as the generalized syndrome. The generalized syndromes form a linear subspace of dimension $n - k$. Any single error gives a generalized syndrome of weight i, and a double error has a generalized syndrome of weight $2i$ or $2i - 1$. Thus for sufficiently small error weights, we can expect the weight of the generalized syndrome to increase with increasing weight of the error pattern.

This observation leads to an iterative approach called *bit flipping*. In general we change a bit in the received vector if the resulting vector has a lower generalized syndrome weight. The rationale for this approach is that we expect to reduce the weight of the error patterns, i.e., the bit that was flipped was in error. The process is repeated until either the syndrome weight is 0, i.e., we have obtained a codeword, or it is no possible to reduce the syndrome weight by changing a single symbol, in which case we declare a decoding failure.

There are a number of variations and extensions on the basic algorithm. One may flip a single bit in each step or change several bits at a time, and the number of parity failures for each bit can be used to select the bits that should be changed. We consider only one version, which is the one that can be analyzed most easily.

10.2.2 A bit-flipping algorithm

We base our discussion on the following algorithm:

Algorithm 10.2.1. *Bit flipping*
Input: *The received vector r.*

1. *Set $y = r$.*
2. *Calculate $s = Hy^T$.*
3. *Select a u such that $\sum_{u \in I_v} s_u > \frac{i}{2}$, set $y_v = y_v + 1 \mod 2$.*
4. *Repeat from 2 until no u satisfies the condition.*

Output: *If $s = 0$, the decoded word, y, else declare failure.*

Lemma 10.2.1. *The weight of the generalized syndrome is decreasing when the bit flipping algorithm is applied.*

Proof. A symbol is changed exactly when the number of 1s in the syndrome can be reduced. □

Theorem 10.2.1. *The iterative algorithm stops after a finite number of iterations with either a correct codeword, or a vector where most of the parity checks are satisfied for all symbols.*

Proof. The theorem follows from Lemma 10.2.1. □

If more than one value of u satisfies the condition for flipping, different choices may lead to the same decoded word. However, it is also possible that only some of the choices work, or that different choices lead to different decoded words. In the last case, flipping several bits at the same time may lead to a decoding failure, while the algorithm succeeds when we take one bit at a time.

10.2.3 Decoding of projective geometry codes

Cyclic codes derived from projective planes were described in Section 5.4. The parameters of these block codes are $n = 2^{2s} + s^s + 1$, $k = n - 3^s - 1$, $d = s^2 + 2$. The parity check lines are the n lines in the plane. Since there are $2^s + 1$ parity checks on each symbol, all n symbols are involved in a majority decision. In each step of the bit flipping algorithm we usually prefer to change a symbol with the largest number of parity failures, thus reducing the syndrome weight as much as possible. There are often several syndromes with this property.

The geometric description allows us in turn to describe the action of the bit flipping algorithm on different error patterns.

There are $2^s + 1$ points on each line and $2^s + 1$ lines through each point. A bit in the generalized syndrome is 0 if the number of errors on the corresponding line is even, 1 if it is odd.

Theorem 10.2.2. *For projective geometry codes the generalized syndrome is a codeword in the dual code.*

Proof. A single error causes the generalized syndrome to equal a column in H, which is the transpose of a row and thus a codeword in the dual code. All other generalized syndromes are linear combinations of these columns. □

In particular we notice that the syndromes for single errors have weight $2^s + 1$, which is the minimum weight of the dual code. Errors of weight 2 always have syndrome weight $2^{s+1} - 1$, since two lines always intersect in a point. We can now go on to classify error patterns of higher weight:

There are two types of errors of weight 3: either the errors are on a line, and the syndrome weight is $3 \cdot 2^s - 1$, or they are in a triangle, in which case the syndrome weight is $2^s - 3$.

4 errors can be on a line, there can be 3 errors on a line and one outside, or the errors are on a quadrangle. Again the syndrome weights follow.

In general, the lowest syndrome weight is obtained when no 3 points are on a line, and in particular a minimum weight codeword has this property. Clearly for a subset of points on a weight $2^2 + 2$ codeword, the syndrome weight decreases once the weight of the pattern exceeds $d/2$.

Example 10.2.1. Decoding of the $(73, 45, 10)$ code.
Here we give a few more details of bit flipping for this code. Since $i = 9$, majority decoding gives the unique closest codeword for any error pattern of weight $< d/2$. If 4 errors occur, and 3 are on a line, 8 of the parity checks involving an error on the line will fail, while only 6 fail on the fourth error. Thus one of the 3 errors will be flipped, first leaving 3 errors on a triangle. It makes no difference in what order the remaining error positions are flipped. There are 5 cases for errors of weight 5: all errors on a line, 4 errors on a line, 3 errors on each of two lines such that one is on the intersection, 3 errors on a line and 2 outside, and no 3 errors on a line. Only the last case is part of a weight 10 word, and the other patterns are correctly decoded by the bit flipping algorithm. In the last case each of the error positions has 5 parity failures, and it is thus a candidate for bit flipping. After the first error is corrected, the remaining 4 positions are uniquely determined. However, since the error positions are part of the support of a weight 10 codeword, there are other positions with 5 parity failures. It turns out that there are always 3 possible error patterns of this type in the same coset. Nevertheless most error patterns of weight 5 are corrected. The bit flipping algorithm also corrects some unique error patterns of weight 6 and 7, but in most cases cosets of these weights have several patterns of the same weight.

10.3 Decoding by message passing

The majority decisions described in the previous section are based on a local subset of the code graph, and in general repeated application of this rule does not lead to the best decision. Here we give an analysis of an algorithm that propagates information about symbols in a larger subset of the graph. The situation is particularly simple if the graph is a *tree*, i.e., it is connected, but if any one edge is removed, the graph is no longer connected, in particular a tree has no circuits. Thus for each edge we can talk about the subgraph on either side of it.

Definition 10.3.1. *In an* LDPC *code, a code defined by a subgraph consisting of Y of the parity nodes and all symbol nodes connected to them is called a* tree code *if the corresponding graph is a tree.*

Lemma 10.3.1. *In an (N, i, j)* LDPC *code with $j > 2$, a tree code is a linear $(Yj - Y + 1, Yj - 2Y + 1, 2)$ code.*

Proof. If there are Y parity checks in the code, there are Yj branches connected to symbol nodes. However $Y - 1$ of these symbols are shared between two parity checks when they are connected to form a tree. In a tree some nodes called leaves are connected to only one other node. In this case the leaves are symbol nodes.

For $j > 2$, there is at least one parity node that is connected to at least two leaves. Thus all parity checks are satisfied if these are the only nonzero symbols. □

Thus a tree code in itself has poor minimum distance, and in particular the leaves are poorly protected.

Example 10.3.1. Tree codes in LDPC codes.
In the code considered in Example 10.2.1, $i = j = 9$. The nine parity checks on a particular symbol define a tree code consisting of all 73 symbols. By making a majority decision about each symbol, 4 errors can be corrected.

In order to analyze the following algorithm, we state the decoding problem in a more general way: Assume that there is an integer weight $a_v(c_v)$ associated with the symbol v, i.e., the weight has value $a_v(0)$ if $c_v = 0$ and $a_v(1)$ if $c_v = 1$. For a codeword c we get the weight

$$A(c) = \sum_v a_v(c_v).$$

Find a codeword c such that the weight is minimized:

$$A = \min_{c \in C} A(c).$$

Thus, in particular, the usual maximum likelihood decoding problem is obtained by letting $a_v(x) = r_v + x \mod 2$.

For each edge in the tree connecting symbol v and parity check u, we define a *message* being passed in each direction, each in the form of a pair of integers.

Definition 10.3.2. *The* message $m_s(v, u, x)$ *from symbol v to parity check u indicates the minimum value of the weight function over codewords in the subtree that includes the symbol, but not the parity check, conditioned on $c_v = x$. Similarly, the message $m_p(v, u, x)$ from parity check u to symbol v indicates the minimum value of the weight function over codewords in the subtree that includes the parity check, but not the symbol, conditioned on $c_v = x$.*

Lemma 10.3.2. *The minimal weight may be found from the messages on any branch as*

$$A = \min_x [m_s(v, u, x) + m_p(v, u, x)].$$

Proof. Each edge separates the tree into two subtrees, and the minimal weight is obtained by adding the contributions for the subtrees with one of the choices for $c_v = x$. □

Theorem 10.3.1. *The messages defined in Definition 10.3.2 can be calculated recursively starting from the leaves of the tree. When all incoming messages to a node are known, the outgoing messages are calculated as*

1. *Symbol node v:*

$$m_{\mathrm{s}}(v, u', x) = a_v(x) + \sum_{u \in I_v, u \neq u'} m_{\mathrm{p}}(v, u, x).$$

2. *Parity node u:*

$$m_{\mathrm{p}}(v', u, x') = \sum_{v \in J_u, v \neq v'} \min_x m_{\mathrm{s}}(v, u, x)$$

when x' satisfies the parity check with the input symbols that give minimal message values. For the other value of x', change one of the m_{s} such that the increase is minimal.

Proof. For a symbol node, the minimal value is the sum of the incoming values from subtrees connected to it plus the weight of the symbol itself. All of these values are conditioned on the same symbol value. For a parity node we first find the sum of the minimal incoming messages. The associated symbol values determine the possible value on the output edge. In order to get the message for the other symbol value, we have to change at least one input symbol. The smallest increase in the weight is obtained by changing only one message. In the first step we can calculate the messages from those symbol nodes that are leaves in the tree, since the only input here are the symbol weights $a_v(c_v)$. After each step, the new set of nodes that have all input messages defined may be obtained as the leaves of the tree that remain when the leaves from the previous stage have been removed. □

The minimal weight calculated from the messages is a lower bound on the number of errors in the tree. Even though not all errors can be corrected with certainty, decoding the tree code serves as a way of estimating the number of errors.

The messages can be simplified by always taking the difference between the two values rather than transmitting a pair of integers. This change has no effect on the result of the decoding, but we lose information about the number of errors that are corrected. In this case the initial values of the weights are $a_v = a_v(1) - a_v(0)$, i.e., ± 1. The numerical value of the message difference indicates the reliability of the symbol. If all symbol weights are initiated to $a_v(c_v)$ and the outgoing messages are initially set to this value, we can use the updating from Theorem 10.3.1. By iterating this calculation a finite number of times, we arrive at the same result. The reason is that the message from a subtree does not depend on the incoming message, and thus the correct value will propagate from the leaves as before.

If the symbol weights are initiated in the same way in a general graph, and the same updating rules are used, the messages in a given edge after s steps will depend only on nodes that can be reached in s steps from this branch. If this part of the graph is a tree, the messages may be interpreted as for the tree code.

Algorithm 10.3.1. *Iterative decoding by message-passing.*
Input: *The received vector r.*

1. *Initially the received symbols are assigned weights $a_v(c_v) = r_v + 1 \mod 2$.*
2. *The messages are calculated as in Theorem 10.3.1.*
3. *The algorithm continues until the weights are unchanged or a preset limit on the number of iterations has been reached.*
4. *For each symbol, y_v choose the value that gives the minimum of*

$$a_v(x) + \sum_{u \in I_v} m_{\mathrm{p}}(v, u, x).$$

Output: *The decoded word, y.*

The application of Algorithm 10.3.1 is only justified even in some approximate sense as long as the number of nodes that can be reached from a given starting node is less than n. If the difference messages a_v are used, they may be interpreted as reliabilities of the symbols and used in additional iterations of the algorithm. It can be observed experimentally that the performance of the decoding is improved, but an exact analysis of this form of iterative decoding is difficult. However the following result indicates that it is sometimes possible to combine the local results of decoding the tree codes into a global result:

Theorem 10.3.2. *Assume that several tree codes in an* LDPC *code are decoded, and that at least $n - k$ independent parity checks are included in the codes. If T is the largest number of errors detected in the decoding of the tree codes, any error patterns of weight T which is consistent with the decoding results of the tree codes is an* ML *decision.*

Proof. As discussed earlier, the minimal weight obtained by decoding a tree code is a lower bound on the number of errors that has occurred, but there may be several possible error patterns of the same weight. After decoding each code, we can make a list of these patterns. Since the tree codes include all parity checks, an error pattern which is shared between all lists has the right syndrome. Since each tree code is decoded ML, the number of errors cannot be less than T. Thus any error pattern with T errors is ML, but there may not be such an error pattern. □

Example 10.3.2. Message passing in an LDPC code.
The code considered in this example is a $(15, 3, 3)$ LDPC code with the following parity check matrix:

$$\begin{bmatrix}
1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&0&1&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&1&0&0&1\\
0&0&1&0&0&0&0&0&0&0&0&0&1&1&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
0&0&0&0&1&0&0&0&0&0&1&0&0&1&0\\
0&0&0&0&0&1&0&1&0&0&0&0&0&1&0\\
0&0&0&0&0&1&0&0&1&0&0&0&0&0&1\\
0&0&0&0&0&0&1&0&0&1&0&0&1&0&0\\
0&0&0&0&0&0&1&0&0&0&1&1&0&0&0
\end{bmatrix}.$$

The code has dimension 5 and minimum distance 6, and thus the rate and distance are a little lower than the parameters of the equidistant code. The structure of the code makes it suitable as a demonstration of message passing. Using the algorithm described in this section, we initially set a message for each symbol, (0, 1) where the received vector has a 0, and $(1, 0)$ otherwise. At each step of the decoding a message is generated for each branch of the code graph (corresponding to a 1 in the parity check matrix). Each position contains alternatively a message from a symbol to a parity check and a message in the opposite direction.
Let the received word be $(1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0)$. In the second step we calculate messages from the parity checks to the symbols, and again the possible messages are just (0, 1) and $(1, 0)$. The first of these messages is chosen when the sum of the other two symbols in the parity check sum to 0. In the third step we calculate messages reflecting the number of errors in subtrees consisting of a symbol, 2 of the parity checks containing the symbol, and the remaining 4 symbols in those parity checks. Each such message is obtained by adding the messages from the parity checks to the original message from the symbol. Thus the possible values are $(0, 3)$, $(1, 2)$, $(2, 1)$, $(3, 0)$. In the next step we consider the trees consisting of a central parity check, 6 neighboring parity nodes, and all 15 received symbols. In the following step each symbol receives messages about 3 subtrees with 5 parity nodes and 10 symbol nodes. Thus all parity checks are active, but some symbols are counted twice. If at this point we add the original symbol weight and the 3 messages, we can correctly decide the first 5 and the last 4 symbols. The re-

maining 6 give a tie between the two values. We could use the approach from Example 10.1.1 to get the the codeword $(1,1,0,1,0,1,0,1,1,0,0,0,0,0,0)$, correcting 3 errors.

The parity check matrix in the example is a structure where all symbols are reached with a subtree containing two depths of parity nodes (compared to the projective geometry codes where all symbols are reached in a single step). This particular structure is known as a *generalized quadrangle*, and as the cube in real space, the lines form quadrangles, but never triangles. Using a projective space over q symbols, such a structure can be constructed with (q^3+q^2+q+1) points and lines. We should note, however, that LDPC codes constructed from good graphs do not necessarily have particularly good performance. It should be noted that the LDPC codes that have been used in applications have lengths of several thousand bits.

10.4 Decoding product codes by iterated (serial) decoding

As discussed in a previous chapter, we use a graph description of the codes where each node represents the constraints of a component code. The graph reflects the structure of the decoding algorithm, where in each iteration all component codes on one side of the bipartite graph are decoded up to half the minimum distance.

To make the transition from the error pattern to a graph we start out with the graph description of the code. Here each symbol is associated with an edge, and we get a graph representation of the error pattern, the error graph, by keeping all edges that correspond to error positions and removing the remaining edges (and any nodes that do not have incoming edges). Since the error positions are assumed to be mutually independent, the error graph has the properties of a randomly chosen (bipartite) graph.

In each step any combination of at most t errors in the component code is decoded, and for the time being we assume that a component code that contains more errors is left unchanged. Decoding succeeds if all errors are eventually removed when the decoding of component codes is repeated. In terms of the error graph, we remove nodes that have degree less than t and all the edges connected to such nodes. Decoding fails if and only if there is a subset of nodes in the error graph for which all nodes are connected to other nodes in the subset by more than t branches. In graph theory terminology, such a subset is called a $(t+1)$-*core*. Clearly, if all other errors are corrected, the decoding stops with this subset of the error graph uncorrected, and as long as this situation is not reached, the decoding can proceed in at least one node. The result of the decoding process does not depend on the order in which the component codes are decoded.

The analysis of product codes becomes more manageable if all row codes or all column codes are decoded in one step, which is possible since the subsets of symbols are disjoint. This approach allows us to trace the progress of the decoding in each iteration in the form of a degree distribution on the error graph. We use the approximation that the errors corrected in rows are randomly distributed among the columns and vice versa. The actual distribution is slightly more favorable, since errors corrected in a row must be located in different columns, which makes them easier to correct in the next step.

If W errors in the frame are randomly distributed, the number of errors in each row follows a binomial distribution. When n^2 is large compared to W, the number of errors in each row follows a Poisson distribution with mean $M = W/n$. In the following derivation we consider the more general case where t_1 errors are corrected in the rows and t_2 in the columns. The probability that a row contains at least t_1 errors is

$$\pi_{t_1}(m) = \sum_{x \geq t_1} e^{-m} m^x / x!. \tag{10.1}$$

The average number of errors left in a column when all row codes are decoded follows from the Poisson distribution with mean m as

$$\sum_{x > t_1} x e^{-m} m^x / x! = m \sum_{x \geq t_1} e^{-m} m^x / x! = m \pi_{t_1}(m), \tag{10.2}$$

where we have used a well-known identity for the Poisson distribution.

We now introduce the simplifying assumption that the decoded positions are randomly distributed among the columns. Clearly this is not exactly the case for finite n, since t_1 errors in a particular row must be located in different columns. However, the effect vanishes asymptotically, and as long as the total number of errors is large, the approximation is extremely close. Thus, the first decoding of the column codes operates on a Poisson distribution with a reduced mean value

$$m_1 = M \pi_{t_1}(M).$$

We shall prove by induction that after decoding step j, the degree distribution follows a truncated Poisson distribution

$$P[\delta = x] = m_j^x / x!, \quad \text{if } x > t, \tag{10.3}$$

and 0 otherwise. The parameter m_j, which we refer to as the Poisson parameter, is clearly not the mean value of the truncated distribution, but for a given m the average number of remaining errors per row or column follows from (10.2). The Poisson parameter is found from the simple recursion

$$m_{j+1} = M \pi_t(m_j) \tag{10.4}$$

and we alternate between $t = t_1$ for the rows and $t = t_2$ for the columns. We have seen that the first two values $m_0 = M$ and m_1 satisfy (10.4), and the recursion holds for $j = 1$.

If the degrees of the nodes follow a truncated Poisson distribution, and a randomly chosen subset of the branches is removed, nodes that have degree more than t after the decoding of the rows are obtained from columns which had more than t errors before the decoding, and thus their distribution is Poisson with a smaller parameter. In the following decoding all resulting light vertices are removed, and the degree distribution is again a truncated Poisson distribution.

In general the recursion follows by induction, since in each step the number of errors is reduced by a factor of

$$m_j \pi_t(m_j)/m_{j-1}\pi_t(m_{j-1}) = M\pi_t(m_j)/m_{j-1},$$

where we have used (10.2) and the induction hypothesis (10.4). The Poisson parameter is reduced by the same fraction, and we find the new parameter to be

$$m_{j+1} = M\pi_t(m_j).$$

If the initial value, M, is less than some threshold value, m_j converges to zero, while for larger values of M, the iteration stops with a pair of stationary values for m. If we take $t_1 = t_2$, m converges to the largest value such that $m' = M\pi(m')$. We can find the limiting value of M as the smallest value for which there is a nonzero solution for m'. Numerical calculations give $M = 5.15$ for $t = 3$ and $M = 8.36$ for $t = 5$.

Example 10.4.1. If iterative decoding of a product code with $t_1 = 4$, $t_2 = 3$ is started with $M = 5$, we can find the average number of errors that remain in each row and column from (10.4). A calculation gives

$$m_j = 3.674, 3.551, 2.372, 2.115, 0.821, 0.252, 0.0007, 0.$$

We expect the decoding to finish in 8 iterations.

If no decoding errors occur in the component codes, the iterative decoding algorithm terminates, with high probability, with the correct codeword or with a large $(t+1)$ core. Once most of the errors are corrected, it is highly likely that the decoding proceeds almost to completion, but the analysis does not directly cover the final stage. For small values of the channel error probability p, the probability of decoding failure depends on the probability that the error pattern contains a minimal core, i.e., $(t_1+1)(t_2+1)$ errors located in t_1+1 rows and t_2+1 columns. The probability of decoding failure can be approximated as

$$P_{\text{fail}} \approx \binom{n}{t_1+1}\binom{n}{t_2+1} p^{(t_1+1)(t_2+1)}, \tag{10.5}$$

since for each choice of the rows and columns, there can be a minimal core of errors.

Since we are mostly interested in fairly long component codes of high rates, we consider the limit as n increases and t is constant. Actually, the performance for small t_1 and t_2 is improved by taking component codes with different distances. For larger values of t_1, almost all errors should be corrected in the first step, and t_2 can be chosen quite small. This situation takes us back to the serial decoding considered earlier.

For RS codes, the probability that a received vector with more than t errors is decoded is close to $1/t!$ (Theorem 8.2.3). It follows from a similar argument that the probability of decoding a high rate BCH code when more than t errors are present is given by the same expression. Thus we can neglect the influence of decoding errors for moderate values of t. For $1 < t < 5$, decoding errors will occur, and it is often preferable to reduce the probability of these events by using binary codes of even weight. The loss associated with decoding errors can be reduced by a factor of n in a binary BCH code if only $t-1$ errors are corrected, and an extra parity symbol in a RS code gives a similar factor. Thus in these cases the probability of component decoding error vanishes.

The consequence of this analysis is that if at least $t = 2$ errors are corrected in each row and column, a very small probability of decoding failure can be obtained, even when the average number of errors in each row is slightly greater than t (for $t = 1$ there is always a significant probability of a small number of residual errors). When the decoding fails on a marginal channel, there is typically a large number of errors left. For better input bit error probabilities, the probability of decoding failure is very small, but the probability of a small core can be calculated as above.

10.5 Decoding of graph codes

Iterative decoding of product codes can readily be extended to decoding of graph codes on bipartite graphs as described in Chapter 8. If a graph is chosen randomly such that all nodes have degree n, the error graph is the same as in the previous section, and the analysis gives the same performance. Graphs derived from finite planes (as discussed in Section 2.5) give similar decoding thresholds.

For component codes of length n correcting t errors, the fraction of errors corrected is therefore M/n, but by increasing the size of the graph we get the improved performance of a longer code, while the rate of the code and the fractional number of errors corrected remain constant. The amount of computation increases only linearly with the size of the graph.

There are known constructions for large graphs with desirable properties, but since it is usually desirable to have a fairly high code rate, the component codes cannot be short. Even with moderate code lengths, $64 - 256$, the graphs from finite planes are probably large enough for most cases.

10.6 Parallel decoding

Parallel decoding relies on iterating the decoding of the component codes and exchanging information about the reliability of the decoded symbols in a way similar to what we discussed in a previous section. Just as message passing in LDPC codes, parallel decoding is usually applied when reliability information is available about the received symbols. However, as in the earlier section we shall start from binary received data in order to get an interpretation of the messages. The extension to soft decision decoding is discussed in Appendix B.

10.6.1 Message passing in graph codes

We have defined a graph code in terms of a graph where the nodes represent component codes and the symbols are associated with the branches. Here we modify this description to resemble the definition of LDPC codes: the code is represented by a bipartite graph where one set of nodes represent the symbols and the other the component codes. Thus each symbol node is connected to those code nodes that represent constraints which include the symbol.

With this description we can modify the message passing algorithm for LDPC codes to work on graph codes. Assume that the code graph is a tree. As before, there are going to be weights associated with the symbol nodes, and they are initialized as before. For each edge in the tree connecting symbol v and component code u, we define a *message* being passed in each direction, each in the form of a pair of integers.

Definition 10.6.1. *The* message $m_{\mathrm{s}}(v,u,x)$ *from symbol v to component code u indicates the minimum value of the weight function over codewords in the subtree that includes the symbol, but not the component code, conditioned on $c_v = x$. Similarly, the message $m_{\mathrm{p}}(v,u,x)$ from component code u to symbol v indicates the of the minimal value of the weight of weight of codewords in the subtree that includes the component code, but not the symbol, conditioned on $c_v = x$.*

Instead of two messages we can exchange the difference. This value reflects the reliability of the symbol, so the result of the decoding is the same. Clearly, count of the total number of errors is lost.

Let I_v indicate that set of codes that include symbols v, and similarly J_u the set of symbols in code u. The messages can be calculated recursively as for LDPC codes starting from the leaves.

1. Symbol node v:

$$m_{\mathrm{s}}(v,u',x) = a_v(x) + \sum_{u \in I_v, u \neq u'} m_{\mathrm{p}}(v,u,x)$$

2. Component code u:

$$m_{\mathrm{p}}(v', u, x) = \min_{c \in C, c_{v'} = x} \sum_{v \in J_u, v \neq v'} m_{\mathrm{s}}(v, u, c_v)$$

   Thus the minimum is taken over codewords in the component code conditioned on the value of v.

In order to apply this algorithm, we need a decoding algorithm for the component codes that includes the symbol weights.

10.6.2 Parallel encoding and decoding of product codes

In *parallel decoding* the same information symbols are encoded by two (or more) systematic encoders, and the transmitted frame consists of the information symbols and two sets of parity symbols. The information symbols are permuted in some way before the second encoding, and the permutation is referred to as *interleaving*.

Product codes may be seen as a type of parallel encoding if the part of the word representing parity checks on parity symbols is removed. Leaving out these symbols reduces the minimum distance, but if the component codes have moderate rate, the overall rate is increased. Thus the transmitted frame consists of a k by k block of information symbols and two k by $n-k$ blocks of parity symbols, one relating to the row codes and one to the column codes. The symbols are not changed during the decoding, but they are modified by messages passed between the row and column decoders.

A product code can be decoded by the algorithm for graph codes. The largest trees in the graph consist of a single row code and all column codes, and thus the assumption of a tree code is valid only in the first iteration. However, the algorithm is often continued for several more iterations in order to get a single solution that satisfies all constraints. For the later iterations we cannot give a precise analysis.

Example 10.6.1. Parallel product decoding.
The information in this small example is 4 by 4 bits. Each row and column is encoded using a $(8, 4, 4)$ code, and thus the total code is a $(48, 16)$ block code. We get a minimum weight codeword by considering a single nonzero information symbol. In this case there are 3 nonzero parity symbols in each block, and the weight is 7. Let the received block be

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

In the decoding of the first row the received (11000000) is equally close to (11001100) and the zero word. Thus the messages from the first two symbols are $+1$ and $+2$ for symbol values 0 and 1. For the zero vectors, all information symbols have messages 0 and $+3$. In the next step, the same two codewords are closest. The total weight of the zero word is 4, 2 from the errors, and 2 from the messages in the first two positions. For the alternative (11001100) we get weight 2 from the errors, but 4 from the messages. Thus we decide for the zero word, which is the correct decision. It may be noted that in both cases we overestimate the distances to the codewords as 8 and 12, where the actual distances are 4 and 8. The reason for this difference is that the messages from the information symbols are not independent.

As we mentioned earlier, the minimum distance of a product code can be increased by permuting the symbols in the component codes. Such a modification usually comes at the expense of having different 'checks on checks' symbols for rows and columns. Since these parity symbols are not used here, such a permutation will often produce a better code. A pseudo-random permutation can also reduce the dependence between messages, but there is no analysis of these effects for specific permutations.

10.6.3 Parallel encoding and decoding of convolutional codes

Here we modify the product code construction to let all information symbols be encoded by two (often identical) systematic convolutional codes. (This method is sometimes referred to as 'parallel concatenation' or 'turbo' coding in the literature). We simplify the presentation by letting the information symbols be arranged as an k by k bit array, which encoded row-wise by one encoder and column-wise by the other. As mentioned earlier, the performance is improved by using a pseudo-random permutation of the symbols before the second encoding.

The message associated with a particular information symbol is obtained by comparing the closest codeword to one that has the opposite value of that symbol. The two sequences differ by a codeword, which is usually quite short. In principle, the alternative sequence could be found by repeating the Viterbi algorithm while forcing a particular value in that particular position, however, we shall discuss a more efficient algorithm shortly. As before, we subtract the contribution from the information symbol from the message (so that it is not counted twice). The decoding of the two component codes is continued for a prescribed number of iterations with messages exchanged for the information symbols. The decoding is considered successful if the two results agree. If the memory of the convolutional code is short compared to the size of the rows and columns, the decoding is similar to parallel product decoding. As the number of iterations increases, more dependency occurs between the messages, and the analysis is further complicated by the variable-length of convolutional codewords.

The decoder uses a modified Viterbi decoding algorithm which accepts weights for the input symbols and produces a message for each decoded information symbol. The first step is to allow more general cost functions. The state transitions and the corresponding transmitted blocks are given by the matrix Φ as in Chapter 7. In Chapter 7 the costs were simply the number of differences between the received block, r_j, and $\phi_{i'i}$, but the algorithm can be used more generally with $e_{i'i}$ being a function of j, the state transition (i', i), and the received symbols r_j. As long as the costs are added and do not depend on the past segments, the optimality of the solution is proved as in Section 7.3.

As discussed in Section 10.3, the weight of each information symbol can be, $a_j = a_j(1) - a_j(0)$, the difference between the number of errors corrected when information symbol j is assumed to be a 1 and a 0. We find these values by combining the result of the modified Viterbi decoding with a similar decoding performed in the reverse direction through the sequence. Since the reverse algorithm is also optimal, the sequences will eventually have the same accumulated costs, but the intermediate values are different. Hence, at time j' the forward algorithm has calculated the minima of the accumulated costs for sequences starting at time 0 and reaching each of the states at time j', and the reverse algorithm has calculated the accumulated costs from time N back to time j'. Thus we can find the cost of the alternative sequence as the minimum of the sum over the states that have the required information symbol.

The message from code 1 about the information symbol u_j, $m_1(j)$ is now calculated as $a_j - m_2(j)$, where m_2 is the message received from code 2.

Algorithm 10.6.1. *Parallel decoding of convolutional codes.*
Input: *The received sequence.*

1. *Decode code* 1 *using the modified Viterbi algorithm with weights*

$$e_{i'i}(j) = W(\phi_{i'i} + r_j)$$

if the information symbol $u_{i'i} = 0$, *and*

$$e_{i'i}(j) = W(\phi_{i'i} + r_j) + m_2(j)$$

if the information symbol $u_{i'i} = 1$.

2. *Repeat the decoding in the reverse direction to obtain the accumulated weights* $\mu_i'(j)$.
3. *For each* j', *find the new weight of the information symbol at time* j', $a_{j'}$ *as the difference between the weight of the ML sequence and the best sequence with the opposite value of the information symbol at time* j'.
4. *Generate a message for each information symbol as* $m_1(j') = a_{j'} - m_2(j')$.
5. *Permute the information symbols and decode code* 2 *using these messages.*
6. *Iterate between the two decoding steps a prescribed number of times.*

Output: *The decoded information sequence.*

Example 10.6.2. Parallel decoding of convolutional codes.
In this example we illustrate the decoding principle by considering a very small parallel encoding. A systematic convolutional code of rate $\frac{1}{2}$ and memory 1 is generated by $g = (11.01)$. The information bits are assumed to be placed in a 4 by 4 array, and the parity bits are stored as two similar blocks. The zero word with 4 errors in the information bits (as in an earlier example) could be

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Decoding the first two rows or columns, (10.10.00.00.10.10.00.00) gives 5 equally good sequences: the zero sequence and

$$\begin{gathered} (11.10.01.00.00.00.00.00), \\ (00.00.00.00.11.10.01.00), \\ (11.10.01.00.11.10.01.00), \\ (11.10.10.10.10.10.01.00). \end{gathered}$$

As a result, the first 6 information symbols have weight 0. For the 4 positions that are 1, this contribution is subtracted from the message. In this way the zero word is preferred in the second decoding.

The typical construction uses a frame of about 10000 information symbols and a pseudo-random interleaver. The two sets of parity symbols are produced by 16-state encoders with systematic rational encoding function $(1 + x^4)/(1 + x + x^2 + x^3 + x^4)$ as discussed briefly in Section 7.1. The performance of parallel decoding does not improve if the memory of the component codes are increased.

10.7 Problems

Problem 10.7.1.

(1) Perform a systematic encoding of the code from Example 5.1.3 when the information symbols are (11011011000). Use the parity polynomial to determine the first parity symbol and proceed to find the others one at a time. Compare this process to the erasure correction described in Example 10.1.1.

(2) Decode the received vector (010110100110110110110) using majority decisions.

Problem 10.7.2.

(1) Use the structure of a cube to construct a parity check matrix in the following way: Associate a binary symbol with each edge, and let each corner represent a parity check on the three adjoining edges. Write out the matrix.

(2) Find the parameters of the code.

(3) Find a tree code with as many symbols as possible.

(4) Show how an error can be corrected by combining two tree codes.

Problem 10.7.3.

(1) Decode the code from Example 5.1.3 using bit flipping when the received vector is (0101101001101111110110).

(2) How many errors are corrected?

(3) Is the result the same when the symbols are decoded in a different order?

(4) Repeat the decoding using message passing.

Problem 10.7.4.

(1) Use the parity check matrix from Example 5.1.3 to decode the vector (111010001110101) using message passing in a tree code.

(2) Continue the decoding for two more iterations. Is the decision ML?

Problem 10.7.5.

(1) Consider a product code where the component codes have generator and parity check matrices

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

What are the parameters of the code?

(2) Decode the received word

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Problem 10.7.6.

(1) In a large product code there are on the average 3 errors in each row and column. Assume that the errors are randomly distributed and that the component codes are long enough for the number of errors to be well approximated by the Poisson distribution. What is the average number of errors in a column after the first decoding of the row codes?

(2) What fraction of the remaining errors is decoded when the column codes are decoded?

(3) After column decoding, what is the probability that there are 4 errors in a row?

Chapter 11

Algebraic Geometry Codes

In this chapter we briefly describe a class of algebraic block codes, which are a recent generalization of the Reed–Solomon codes.

Where Reed–Solomon codes over $\mathbb{F}_q$ can have length at most q, these so-called *algebraic geometry codes* can be significantly longer. We will not give the general theory here; rather, we present a special class of codes that seems to have the best chance to eventually be used in practice.

11.1 Hermitian codes

A codeword in an (n, k) Reed–Solomon code is obtained by evaluating certain functions (polynomials from $\mathbb{F}_q[x]$ of degree smaller than k) in certain elements $(x_1, x_2, \ldots, x_n)$ from $\mathbb{F}_q$. In particular, we can get a basis for the code by evaluating the monomials $1, x, \ldots, x^{k-1}$ in the n elements.

The algebraic geometry codes are obtained by generalizing this idea in the sense that one chooses points from some geometric object (i.e., curves or higher dimensional surfaces) and then chooses a suitable set of functions. The evaluation of any of these functions in the chosen points gives a codeword. It turns out that with the proper choice of points and functions one gets codes (for large q and n) that are better than previously known codes.

In order to describe the codes, we need some facts on polynomials in two variables. If F is a field, then $F[x, y]$ denotes the set of polynomials in the variables x and y with coefficients from F, i.e., expressions of the form

$$a_{m,n}x^m y^n + \cdots + a_{i,j}x^i y^j + \cdots + a_{0,1}y + a_{1,0}x + a_{0,0},$$

where $a_{i,j} \in F$. The *degree* is the maximum of the numbers $i + j$ with $a_{i,j} \neq 0$. Like polynomials in one variable, a polynomial in two variables is *irreducible* if it cannot be written as a product of polynomials of positive degree.

We state without proof a theorem that plays the same role as Theorem 2.2.2 did for Reed–Solomon codes.

Theorem 11.1.1. (*Bézout*) *Let $f(x, y), g(x, y) \in F[x, y]$ where $f(x, y)$ is irreducible and has degree m, and $g(x, y)$ has degree n. If $g(x, y)$ is not a multiple of $f(x, y)$, then the two polynomials have at most mn common zeroes in F^2.*

In the following we will describe some algebraic geometric codes, the *Hermitian codes*. The construction consists of choosing points on a specific *curve* and then choosing a suitable set of functions whose values in these points give a codeword. This illustrates the general construction.

We choose points $P_1 = (x_1, y_1), P_2 = (x_2, y_2), \dots, P_n = (x_n, y_n)$ as all the different elements of $\mathbb{F}_{q^2}^2$ that satisfy the equation

$$x^{q+1} - y^q - y = 0.$$

It can be shown (see Problem 11.3.3) that the polynomial $x^{q+1} - y^q - y$ is irreducible.

Note that the points have coordinates in $\mathbb{F}_{q^2}$. With the above notation, we have:

Lemma 11.1.1. $n = q^3$.
The points are $(0, \beta)$ *with* $\beta \in \mathbb{F}_{q^2}, \beta^q + \beta = 0$ *and*

$$\left(\alpha^{i+j(q-1)}, \beta_i\right), \quad i = 0, 1, \dots, q-2, \ j = 0, 1, \dots, q,$$

where ${\beta_i}^q + \beta_i = \alpha^{i(q+1)}, \beta_i \in \mathbb{F}_{q^2}$*, and* α *is a primitive element of* $\mathbb{F}_{q^2}$.

Proof. We first note that if $\beta \in \mathbb{F}_{q^2}$, then $\beta^q + \beta \in \mathbb{F}_q$, this follows from Problem 2.6.12. Also $\beta_1^q + \beta_1 = \beta_2^q + \beta_2 \Rightarrow \beta_1^q - \beta_2^q + \beta_1 - \beta_2 = 0$ and therefore $(\beta_1 - \beta_2)^q - (\beta_1 - \beta_2) = 0$. The polynomial $x^q + x$ has exactly q zeroes in $\mathbb{F}_{q^2}$ (see Problem 2.6.12), so $\beta^q + \beta = 0$ in q cases and for each $i \in \{0, 1, \dots, q-2\}$, $\beta^q + \beta = \alpha^{i(q+1)}$ in q cases, since α^{q+1} is a primitive element of $\mathbb{F}_q$. In the first case we get $x^{q+1} = 0$ and hence $x = 0$. In the second case we get $x^{q+1} = \alpha^{i(q+1)}$ with the $q+1$ solutions $\alpha^{i+j(q-1)}$, $j = 0, 1, \dots, q$. Therefore the number of points $n = q + (q^2 - q)(q+1) = q^3$. □

In order to describe the set of functions we will use in the construction of the codes, we introduce the following:

Definition 11.1.1. *For* $x^a y^b \in \mathbb{F}_{q^2}[x, y]$ *the* order function $\rho\colon \mathbb{F}_{q^2}[x, y] \to \mathbb{N}_0$ *is defined as*

$$\rho(x^a y^b) = aq + b(q+1),$$

and for $f(x, y) = \sum_{i,j} f_{i,j} x^i y^j \in \mathbb{F}_{q^2}[x, y]$,

$$\rho\left(f(x, y)\right) = \max_{f_{i,j} \neq 0} \rho\left(x^i y^j\right)$$

We note that $\rho(f(x, y) g(x, y)) = \rho(f(x, y)) + \rho(g(x, y))$, $\rho(x) = q$ and $\rho(y) = q + 1$.

With this we define the Hermitian codes.

Definition 11.1.2. *Let s be a natural number such that $s < n-q^2+q$. A Hermitian code $H(s)$ over $\mathbb{F}_{q^2}$ consists of the codewords*

$$(f(P_1), f(P_2), \dots, f(P_n)),$$

where $f(x, y) \in \mathbb{F}_{q^2}[x, y]$ with $\rho(f(x, y)) \le s$ and $\deg_x(f(x, y)) < q + 1$.

From the construction it is obvious that the length of the code is n, and it is also clear that the code is linear.

To determine the dimension we first note that the evaluation of the monomials

$$M(s) = \left\{x^a y^b \text{ where } 0 \le \rho\big(x^a y^b\big) \le s, 0 \le a < q + 1\right\}$$

in $P_1, P_2, \dots, P_n$ gives a basis for the code. To see this, we first prove

Lemma 11.1.2. *The elements of $M(s)$ have different orders.*

Proof. If

$$\rho\left(x^{a_1} y^{b_1}\right) = \rho\left(x^{a_2} y^{b_2}\right),$$

then $a_1 q + b_1(q+1) = a_2 q + b_2(q+1)$, so $(a_1-a_2+b_1-b_2)(q+1) = (a_1-a_2)$. But the absolute value of the right side is smaller than $q + 1$ and therefore we must have $a_1 = a_2$ and hence $b_1 = b_2$. □

This implies that we have a natural ordering of these monomials, namely, by their ρ value. Let this be $\phi_0, \phi_1, \dots$.

Example 11.1.1. Let $q = 4$ The first 10 polynomials and there orders are

polynomial	1	x	y	x^2	xy	y^2	x^3	x^2y	xy^2	y^3
order	0	4	5	8	9	10	12	13	14	15

Lemma 11.1.3. *The vectors obtained by evaluating the monomials in $M(s)$ in the points $P_1, P_2, \dots, P_n$ are linearly independent.*

Proof. Suppose that $\lambda_0\phi_0(P_j) + \cdots + \lambda_v\phi_v(P_j) = 0$ for $j = 1, 2, \dots, n$.

This implies that the polynomial $\phi(x, y) = \lambda_0\phi_0(x, y) + \cdots + \lambda_v\phi_v(x, y)$ has n zeroes. Since $\deg_x(\phi(x, y)) < q + 1$, it is not a multiple of $x^{q+1} - y^q - y$ and the degree of $\phi(x, y)$ is at most $\frac{s+q}{q+1}$, and since $P_1, P_2, \dots, P_n$ are points on the curve with the equation $x^{q+1} - y^q - y = 0$ which has degree $q + 1$, it follows from Theorem 11.1.1 that the number of zeroes is at most $s + q$. But $s + q < n$ and therefore $\lambda_0\phi_0(x, y) + \cdots + \lambda_v\phi_v(x, y)$ must be the zero polynomial, so $\lambda_0 = \lambda_1 = \cdots = \lambda_v = 0$. □

Let $\mu(s)$ denote the number of elements in $M(s)$.

We then have

Theorem 11.1.2. *If $s > q^2-q-2$, then $\mu(s) = s+1-\frac{q^2-q}{2}$, and for $s > q^2-q-1$ there exists (a,b) with $0 \le a \le q, 0 \le b$ such that $aq + b(q+1) = s$.*

Proof. By induction on s. If $s = q^2 - q - 1$, then the number of solutions to the inequality that has $a = j$ is $q-1-j$, so the total number is $\frac{q^2-q}{2}$, corresponding to the statement of the theorem. If $s = q^2 - q$, then $(q-1)q = s$. Suppose we have proven the statement for all s^* where $q^2 - q \le s^* < s$. Then if $a_1q + b_1(q+1) = s-1$, we get $(a_1-1)q + (b_1+1)(q+1) = s$. If $a_1 = 0$, we get $s = q \cdot q + (b_1 + 1 - q)$. $\square$

This combined with the observation above immediately gives

Theorem 11.1.3. *The dimension $k(s)$ of the code $H(s)$ is*

$$k(r) = \begin{cases} \mu(s), & \text{if } 0 \le s \le q^2-q-2, \\ s+1-\frac{q^2-q}{2}, & \text{if } q^2-q-2 < s < n-q^2+q. \end{cases}$$

The number $g = \frac{1}{2}(q^2-q)$ is called the *genus* of the curve $x^{q+1}-y^q-y = 0$. It can be proved that this is the number of natural numbers that do not appear as orders.

Corollary 11.1.1. *It can be seen from the above that $\rho(\phi_r) = r+g-1$ if $r \ge g+1$ and that $\rho(\phi_r) < r+g-1$ if $r < g+1$.*

We also have

Theorem 11.1.4. $H(s)^\perp = H(n+q^2-q-2-s)$.

Proof. We will only prove the theorem in the case where $s > q^2-q-2$. From the above it is easy to see that $k(s)+k(n+q^2-q-2-s) = n$, so it suffices to show that any element of $H(s)$ is orthogonal to any element of $H(n+q^2-q-2-s)$.

This is the same as showing that if (a_1,b_1) and (a_2,b_2) satisfy $a_1q+b_1(q+1) \le s, a_1 < q+1$ and $a_2q + b_2(q+1) \le n+q^2-q-2-s, a_2 < q+1$, then

$$\sum_{i=0}^{n} x^{a_1+a_2}y^{b_1+b_2}(P_i) = 0.$$

By Lemma 11.1.1, the left side of this equation is the same as

$$\sum_{\beta^q+\beta=0} 0^{a_1+a_2}\beta^{b_1+b_2} + \sum_{i=0}^{q-2} \sum_{\beta^q+\beta=\alpha^{i(q+1)}} \beta^{b_1+b_2} \sum_{j=0}^{q} (\alpha^{i+j(q-1)})^{a_1+a_2}.$$

If $a_1 + a_2 = 0$ this becomes

$$\sum \beta^{b_1+b_2} = 0.$$

If $a_1 + a_2 \neq 0$, we get

$$\sum_{i=0}^{q-2} \alpha^{i(a_1+a_2)} \sum_{\beta^q+\beta=\alpha^{i(q+1)}} \beta^{b_1+b_2} \sum_{j=0}^{q} \alpha^{j(q-1)(a_1+a_2)}.$$

Since

$$\sum_{j=0}^{q} \alpha^{j(q-1)(a_1+a_2)} = \frac{\alpha^{(q^2-1)(a_1+a_2)} - 1}{\alpha^{(q-1)(a_1+a_2)} - 1},$$

the sum is 0 if $\alpha^{(q-1)(a_1+a_2)} \neq 1$.

The case $\alpha^{(q-1)(a_1+a_2)} = 1$ is treated in Problem 11.3.4.

Note that it is now easy to get a parity check matrix for the code $H(s)$. The rows of this matrix are the evaluations of the monomials in $M(n + q^2 - q - 2 - s)$ in the points $P_1, P_2, \ldots, P_n$.

Our final task is to estimate the minimum distance of the Hermitian codes. Here we have the following

Theorem 11.1.5. *Let $d(s)$ denote the minimum distance of the code $H(s)$. If $q^2 - q - 2 < s < n - q^2 + q$, then $d(r) \geq n - s$.*

We will only prove this in the case where $s = h(q + 1)$ for some nonnegative integer h. We can show that a codeword $(f(P_1), \ldots, f(P_n))$ in $H(s)$ has weight at least $n - s$ by showing that the polynomial $f(x, y)$ has at most s zeroes among the points $P_1, \ldots, P_n$. Since $aq + b(q + 1) \leq h(q + 1)$ we get $(a + b)(q + 1) \leq h(q + 1) + q$ and hence $a + b \leq h + \frac{q}{q+1}$. Therefore $a + b \leq h$, so the degree of $f(x, y)$ is at most h. We also have that $\deg_x(f(x, y)) < q + 1$ and therefore, by Theorem 11.1.1, $f(x, y)$ has at most $h(q + 1) = s$ zeroes, since $P_1, P_2, \ldots, P_n$ are on a curve of degree $q + 1$. □

A proof of the theorem for other values of s is more difficult, actually one can prove that $d(r) = n - s$.

Combining Theorem 11.1.3 and Theorem 11.1.5 we get

Corollary 11.1.2. *$d(s) \geq n - k(s) + 1 - g$ for the codes $H(s)$.*

Recall that for Reed–Solomon codes we have $d = n - k + 1$, so the algebraic geometry codes have smaller minimum distance, but are longer.

Example 11.1.2. Hermitian codes over $\mathbb{F}_{16}$.
With $q = 4$ we get $(64, s-5, 64-s)$ codes over $\mathbb{F}_{16}$ for $s = 11, 12, \ldots, 51$. If we take the $(64, 32, 27)$ code over $\mathbb{F}_{16}$ and concatenate it with the $(8, 4, 4)$ extended Hamming code, we get a $(512, 128, 108)$ binary code.

11.2 Decoding Hermitian codes

In this section we will present a modification of the method from Section 4.2 to give a decoding algorithm for Hermitian codes.

We consider the code $H(s)$ as presented in Section 11.1. Suppose we receive a word $r = (r_1, r_2, \ldots, r_n)$ which is the sum of a codeword $c = (f(P_1), f(P_2), \ldots, f(P_n))$ and an error vector e of weight τ. The idea is to determine a polynomial in three variables

$$Q(x, y, z) = Q_0(x, y) + zQ_1(x, y) \in \mathbb{F}_{q^2}[x, y, z]\backslash\{0\}$$

1. such that $Q(x_i, y_i, r_i) = 0,\ i = 1, \ldots, n$.
2. $\rho(Q_0) \le s + \tau + g$.
3. $\rho(Q_1) \le \tau + g$.

Theorem 11.2.1. *If τ errors occurred, then there is at least one nonzero polynomial $Q(x, y, z)$ which satisfies the three conditions.*

Proof. If there are τ errors, then there exists a polynomial $Q_1(x, y) = \sum_{i=0}^{\tau} \lambda_i \phi_i$ with the error positions as zeroes, since we get a system of τ homogenous linear equations in $\tau + 1$ unknowns. If $f(x, y)$ is the polynomial that gives the transmitted codeword and $r(x, y)$ is the polynomial corresponding to the received word, we then have $f(P_j)Q_1(P_j) = r(P_j)Q_1(P_j)$ for $j = 1, 2, \ldots, n$, since $f(P_j) = r(P_j)$ if P_j is not an error position and $Q_1(P_j) = 0$ if P_j is an error position. This means that $Q(x, y, z) = f(x, y)Q_1(x, y) - zQ_1(x, y)$ satisfies all three conditions. $\square$

Theorem 11.2.2. *If the number of errors is less than $\frac{d-g}{2}$, then the error postions are zeroes of $Q_1(x, y)$.*

Proof. If $\tau < \frac{d-g}{2} = \frac{n-s-g}{2}$ we get $s + \tau + g < n - \tau$. The polynomial $Q(x, y, f(x, y))$, where the transmitted codeword is generated by $f(x, y)$, has at least $n - \tau$ zeroes among the points $P_1, P_2, \ldots, P_n$. The degree of $Q(x, y, f(x, y))$ is smaller than or equal to $\frac{s+\tau+g}{q+1}$, so by Theorem 14.1.1 it must either be the zero polynomial, or a multiple of the polynomial $x^{q+1} - y^q - y$. In both cases we get $Q_0(P_j) + f(P_j)Q_1(P_j) = 0$ for $j = 1, 2, \ldots, n$. Since we have $Q_0(P_j) + r(P_j)Q_1(P_j) = 0$ for $j = 1, 2, \ldots, n$, we get by subtraction $(r(P_j) - f(P_j))Q_1(P_j) = 0$ for $j = 1, 2, \ldots, n$ and hence $Q_1(P_j) = 0$ if P_j is an error position. $\square$

From the above we can now give a decoding algorithm for the codes $H(s)$. We define $l_0 = s + \lfloor \frac{d-g}{2} \rfloor$ and $l_1 = \lfloor \frac{d-g}{2} \rfloor$.

The algorithm can be presented as follows:

Algorithm 11.2.1.
Input: *A received word* $r = (r_1, r_2, \ldots, r_n)$.

1. *Solve the system of linear equations*

$$\begin{bmatrix} \phi_0(P_1) & \phi_1(P_1) & \ldots & \phi_{l_0}(P_1) & r_1\phi_0(P_1) & r_1\phi_1(P_1) & \ldots & r_1\phi_{l_1}(P_1) \\ \phi_0(P_2) & \phi_1(P_2) & \ldots & \phi_{l_0}(P_2) & r_2\phi_0(P_2) & r_2\phi_1(P_2) & \ldots & r_2\phi_{l_1}(P_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_0(P_n) & \phi_1(P_n) & \ldots & \phi_{l_0}(P_n) & r_n\phi_0(P_n) & r_n\phi_1(P_n) & \ldots & r_n\phi_{l_1}(P_n) \end{bmatrix} \begin{bmatrix} Q_{0,0} \\ Q_{0,1} \\ Q_{0,2} \\ \vdots \\ Q_{0,l_0} \\ Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,l_1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

2. *Put* $Q_1(x, y) = \sum_{j=0}^{l_1} Q_{1,j}\phi_j$.

3. *Find the zeroes of* $Q_1(x, y)$ *among the points* $P_1, P_2, \ldots, P_n$.

4. *Calculate the first* $l_1 + g - 1$ *syndromes using the parity check matrix of the code and the received word.*

5. *Solve the system of linear equations using the parity check matrix and the (possible) error positions and the calculated syndromes to find the error values.*

Output: *The error vector.*

Notice in the system above that each row of the matrix corresponds to a triple (x_i, y_i, r_i). We have already seen that if the number of errors is smaller than $\frac{d-g}{2}$, then the output of the algorithm is the sent word.

Example 11.2.1. Decoding of the code $H(27)$ over $\mathbb{F}_{16}$.
The code has parameters $(64, 22, 37)$ and the above algorithm corrects 15 errors. If we consider a special situation where $f(x, y) = x^6$, which has order 24, and suppose the errors occur in points (x, y) where $x^5 = 1$, then we can use the polynomial x^5-1 as $Q_1(x, y)$ (it has order $20 \leq 15+6$) and in the correct position we have $Q_0(x, y) = f(x, y)Q_1(x, y)$, so we can use $Q_0(x, y) = x^{11} + x^6$. Finally, we have to solve 20 equations to get the errors values, so in this case we can correct more than 15 errors. However, there may be other solutions for $Q_0(x, y)$ and $Q_1(x, y)$.

It is possible to present another version of the above algorithm as we did in Section 4.4, and generalize it so that all error patterns of weight smaller than half the minimum distance can be decoded; we refer to the specialized literature for that.

11.3 Problems

Problem 11.3.1.

(1) With $s = 27$, what are the parameters of the code $H(s)$ over $\mathbb{F}_{16}$?

(2) Find a generator matrix for the code.

(3) What are the parameters of the dual code?

(4) How can you find a parity check matrix of the code?

Problem 11.3.2. *Project.*
Write a program that implements the decoding algorithm for the codes $H(s)$ over $\mathbb{F}_{16}$.
Try to decode random error patterns of weight up to half the minimum distance.

Problem 11.3.3. Show that the polynomial $x^{q+1} - y^q - y$ is irreducible in $\mathbb{F}_{q^2}[x, y]$.
Hint: write $x^{q+1} - y^q - y$ as $(a(x) + yb(x, y)) \cdot (c(x) + yd(x, y))$.

Problem 11.3.4. Complete the proof of Theorem 11.1.4 by considering the case when $\alpha^{(q-1)(a_1+a_2)} = 1$.

Appendix A

Some Results from Linear Algebra

A.1 Vandermonde matrices

In this section we give some results for matrices with a special structure. These matrices play a significant role in the text, in particular in Chapter 4.

Lemma A.1.1. *Let $\beta \in \mathbb{F}_q$ be an element of order n, i.e. $\beta^n = 1$ and $\beta^i \neq 1$ for $0 < i < n$ and let $x_j = \beta^{j-1}$, $j = 1, 2, \ldots, n$.*

$$\text{Let } A = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ \vdots & \vdots & \ldots & \vdots \\ x_1^a & x_2^a & \ldots & x_n^a \end{bmatrix}$$

and

$$B = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & \ldots & \vdots \\ x_1^s & x_2^s & \ldots & x_n^s \end{bmatrix},$$

where

$$s + a + 1 \leq n.$$

Then

$$BA^T = 0.$$

Before we prove the claim we note that it also follows that $AB^T = 0$.

Proof. Let $C = BA^T$. Since $b_{ij} = x_j^i$ and $a_{rj} = x_j^{r-1}$, we get

$$c_{ir} = \sum_{j=1}^{n} x_j^i x_j^{r-1} = \sum_{j=1}^{n} x_j^{i+r-1} = \sum_{j=1}^{n} \left(\beta^{i+r-1}\right)^{j-1}.$$

Since

$$i + r - 1 \leq s + a \leq n - 1,$$

so

$$\beta^{i+r-1} \neq 1,$$

we get

$$c_{ir} = \frac{\left(\beta^{i+r-1}\right)^n - 1}{\beta^{i+r-1} - 1}$$

but

$$\left(\beta^n\right)^{i+r-1} = 1$$

and the result follows. □

If $a = n - 1$, the matrix A is a so-called Vandermonde matrix, and its determinant is readily calculated.

Theorem A.1.1. *Let*

$$D_n = \begin{vmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{vmatrix},$$

where $x_i \in \mathbb{F}_q$.

Then

$$D_n = \prod_{i>j}^{n} \left(x_i - x_j\right).$$

Proof. By induction on n. We have $D_2 = x_2 - x_1$. D_n can be considered as a polynomial of degree $n - 1$ in the variable x_n. As such it has zeroes $x_1, x_2, \dots x_{n-1}$, and since the coefficient to x_n^{n-1} is D_{n-1}, we get

$$D_n = \prod_{i=1}^{n-1} (x_n - x_i)\, D_{n-1}$$

and the result follows. □

Corollary A.1.1.

$$\begin{vmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_t} \\ x_{i_1}^2 & x_{i_2}^2 & \dots & x_{i_t}^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_{i_1}^t & x_{i_2}^t & \dots & x_{i_t}^t \end{vmatrix} = x_{i_1} \cdots x_{i_t} \prod_{1 \le l < j \le t} \left(x_{i_j} - x_{i_l}\right)$$

For later use we have also

Corollary A.1.2.

$$\begin{vmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_{S-1}} & S_1 & x_{i_{S+1}} & \dots & x_{i_t} \\ x_{i_1}^2 & x_{i_2}^2 & \dots & x_{i_{S-1}}^2 & S_2 & x_{i_{S+1}}^2 & \dots & x_{i_t}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i_1}^t & x_{i_2}^t & \dots & x_{i_{S-1}}^t & S_t & x_{i_{S+1}}^t & \dots & x_{i_t}^t \end{vmatrix}$$

$$= x_{i_1} \cdots x_{i_{s-1}} \cdot x_{i_{s+1}} \cdots x_{i_t} \prod_{1 \le l < S < j \le t} \left(x_{i_j} - x_{i_l} \right) \sum_{r=1}^{t} p_r^{(S)} S_r ,$$

where $p_r^{(S)}$ are the coefficients of the polynomial.

$$P^{(S)}(x) = \sum_{r=1}^{t} p_r^{(S)} x^r = (-1)^{t-s} x \prod_{\substack{m=1 \\ m \ne s}}^{t} (x - x_{i_m}) .$$

Proof.

$$d = \begin{vmatrix} x_{i_1} & x_{i_2} & \dots & x_{i_{s-1}} & x & x_{i_{s+1}} & \dots & x_{i_t} \\ x_{i_1}^2 & x_{i_2}^2 & \dots & x_{i_{s-1}}^2 & x^2 & x_{i_{s+1}}^2 & \dots & x_{i_t}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i_1}^t & x_{i_2}^t & \dots & x_{i_{s-1}}^t & x^t & x_{i_{s+1}}^t & \dots & x_{i_t}^t \end{vmatrix}$$

$$= x_{i_1} \cdots x_{i_{s-1}} \cdot x_{i_{s+1}} \cdots x_{i_t} \prod_{1 \le l < s < j \le t} \left(x_{i_j} - x_{i_l} \right) (-1)^{t-s} x \prod_{\substack{m=1 \\ m \ne s}}^{t} (x - x_{i_m})$$

by Corollary A.1.1, and hence

$$d = x_{i_1} \cdots x_{i_{s-1}} \cdot x_{i_{s+1}} \cdots x_{i_t} \prod_{1 \le l < s < j \le t} \left(x_{i_j} - x_{i_l} \right) P^{(s)}(x),$$

so by replacing the x^j with S_j the claim follows. □

We return to the matrix A and write its elements as powers of β:

$$A = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \beta & \dots & \beta^{n-1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \beta^a & \dots & \beta^{(n-1)a} \end{bmatrix}$$

In the special case where A is a square matrix, i.e., $a = n - 1$, and g is the vector of coefficients of a polynomial $g(x)$ of degree $< n$, then

$$G = Ag^T$$

is a vector of values of $g(x)$. It is easy to see that in this case

$$A^{-1} = n^{-1} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & \gamma & \dots & \gamma^{n-1} \\ \vdots & \vdots & \dots & \vdots \\ 1 & \gamma^{n-1} & \dots & \gamma^{(n-1)(n-1)} \end{bmatrix},$$

where $\gamma = \beta^{-1}$, and then

$$g = A^{-1}G^T.$$

(The factor n^{-1} makes sense since $n | q - 1$ so $q - 1 = an$ and therefore $n^{-1} \equiv a \mod p$).
This means that the encoding of Reed–Solomon codes an be interpreted as a finite field transform (sometimes called a finite field Fourier Transform), and similarly the syndrome can be described as the transform of the received word.

A.2 A useful theorem

We present a theorem that is useful in the study of graph codes.

Theorem A.2.1. *Let A be a real symmetric $m \times m$ matrix* (where $m \geq 2$) *with eigenvalues $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_m$ and x_1 an eigenvector for λ_1. If $y \in \mathbb{R}^m$ satisfies $x_1 \cdot y = 0$ then*

$$y^T Ay \leq \lambda_2 |y|^2$$

Proof. Without loss of generality we can assume that $|x_1| = 1$. For $i = 2, \dots, m$, let x_i be eigenvectors for λ_i respectively such that $\{x_1, x_2, \dots, x_m\}$ is an orthonormal basis for $\mathbb{R}^m$. Let $y = \sum_{i=1}^m \alpha_i x_i$, where $\alpha_i = x_1 \cdot y$ so $\alpha_1 = 0$. Now

$$Ay = \sum_{i=2}^m \alpha_i A x_i = \sum_{i=2}^m \alpha_i \lambda_i x_i,$$

and therefore

$$y^T Ay = \sum_{i=2}^m \alpha_i \lambda_i y^T x_i = \sum_{i=2}^m \sum_{j=2}^m \alpha_i \lambda_i \alpha_j {x_j}^T x_i$$

$$= \sum_{i=2}^m {\alpha_i}^2 \lambda_i \leq \lambda_2 \sum_{i=2}^m {\alpha_i}^2 = \lambda_2 |y|^2. \qquad \square$$

Appendix B

Communication Channels

In some communication channels the output is accurately described as the sum of (a scaled version of) the input and an independent noise term, which follows a Gaussian distribution. This description applies to deep space communication where several coding techniques were first applied. This appendix describes how the codes studied in the book are applied to such channels, and how some of the decoding techniques can be modified to work in this context.

B.1 Gaussian channels

The concept of mutual information as introduced in Chapter 3 can be extended to real variables. We define the mutual information by a modification of (3.2):

$$I(X;Y) = E\left[\log \frac{p(y|x)}{p(y)}\right], \tag{B.1}$$

where p is the density function. Note that for a ratio between two probabilities, we may use the ratio of the density functions. Again the capacity is the maximum of this quantity with respect to an acceptable input distribution. We shall consider only the following special case that is of practical importance:

Definition B.1.1. *A* discrete time memoryless additive Gaussian channel *is defined by variables X and Y, where X has mean zero and standard deviation σ, and $Y = X + N$, where the noise variable N is normal with mean zero and standard deviation ρ and independent of X.*

We note without proof that the mutual information is maximized by choosing Y to be Gaussian, and that this implies X Gaussian. With this choice the capacity may be calculated as follows.

Theorem B.1.1. *The capacity of the discrete memoryless Gaussian channel is*

$$C = \frac{1}{2}\log\left(1 + \frac{\sigma^2}{\rho^2}\right). \tag{B.2}$$

Proof. For a normal variable with standard deviation α,

$$E\left[\log(p(x))\right] = -\frac{1}{2}\log\left(2\pi\alpha^2\right) + E\left[-\frac{x^2}{2\alpha^2}\log e\right].$$

It follows from the definition of the variance that the last term is $-\frac{1}{2}\log e$. Applying this expression to the output, Y, and Y conditioned on X, which is the noise distribution, gives the theorem. □

In many communication systems the modulation system gives rise to two identical and mutually independent channels (quadrature modulation). It is sometimes convenient to represent these as a single complex channel where the noise is a complex Gaussian variable. We may find the capacity as twice the value given by (B.2)

B.2 Gaussian channels with quantized input and output

As we mentioned in the previous section, the mutual information of the Gaussian channel is maximized by using a normal distribution as the input. However, for practical communication it is convenient to use a discrete input distribution with the smallest possible number of levels. Clearly, we have to use more than A input symbols to reach a capacity of $\log A$. Similarly, the receiver is simplified if we can work with a quantized output. The easiest situation would be to distinguish the same number of levels as used for the input, but better performance is obtained with finer output quantization.

Example B.2.1. A quantized Gaussian channel.
The Gaussian channel may be converted to a binary symmetric channel by using inputs ± 1 and observing the sign of the output. Introducing the function $Q(\alpha)$ as the probability that a normal variable with mean 0 and variance 1 exceeds α, we have $p = Q(1/\rho)$ for the discrete channel. For the Gaussian channel, $\sigma = \rho = 1$ gives $C = \frac{1}{2}$. However, $Q(1) = 0.1587$, and the capacity of a BSC with $p = 0.1587$ is only 0.3689. Thus the quantization introduces a significant degradation. However, most of the degradation is due to the output quantization.

Let the input alphabet be real numbers $\{x_1, x_2, \ldots, x_r\}$, and let the output be compared to a set of thresholds $\{s_1, s_2, \ldots, s_m\}$. Thus the output alphabet is $\{y_1, y_2, \ldots, y_{m+1}\}$, where the discrete output y_j indicates that the real output, $x+n$, satisfies $s_{j-1} < x+n < s_j$, $1 < j < m+1$, y_1 corresponds to $x+n < s_1$, and y_{m+1} to $x+n > s_{m+1}$. The transition probabilities may be found as

$$P\left[y_j|x_i\right] = Q\left(\frac{x_i - s_j}{\rho}\right) - Q\left(\frac{x_i - s_{j+1}}{\rho}\right), \tag{B.3}$$

where ρ is the standard deviation of the noise.

Example B.2.2. (Example continued).
With the same channel as before, we keep the binary input, but use eight output symbols defined by thresholds at $0, \pm\frac{1}{2}, \pm 1$, and $\pm\frac{3}{2}$. The transition matrix becomes

$$Q = \begin{pmatrix} 0.3085 & 0.1915 & 0.1915 & 0.1499 & 0.0918 & 0.0441 & 0.0165 & 0.0062 \\ 0.0062 & 0.0165 & 0.0441 & 0.0918 & 0.1499 & 0.1915 & 0.1915 & 0.3085 \end{pmatrix}.$$

As illustrated in the example, a Gaussian channel can be converted to a discrete channel by using a simple discrete input alphabet and performing a suitable quantization of the output. The capacity of the discrete channel is upper bounded by the value (B.2) for the Gaussian channel with the same input variance and noise. In particular binary codes can be used on channels with capacity less than 1, but to get the best performance $4-16$ output levels are often used (*soft decision*).

Example B.2.3. A complex Gaussian channel.
A commonly used system of transmitted points for a complex Gaussian channel is the set $\{\pm 1 \pm i, \pm 1 \pm 3i, \pm 3 \pm i, \pm 3 \pm 3i\}$ which is known as 16 QAM (quadrature amplitude modulation). At the output the plane may be divided into decision regions by using the thresholds 0 and ± 2 for both the real and the imaginary part. If the input symbols are equally likely, the variance of the input is 5. The 16 input points are conveniently mapped to binary input symbols by coding the four levels as 00, 01, 11, 10. In this way an error that causes the received signal to be interpreted as a neighboring symbol in the real or imaginary direction will be mapped to a single bit error. Thus there is no great loss in using a binary code on this channel. Another alternative is a Reed–Solomon code.

B.3 ML Decoding

When equally likely codewords from a discrete alphabet are used in the transmission, the receiver should select the codeword that maximizes $P[r|c_i]$. This is the Maximum Likelihood decision. For a memoryless channel this probability can be expressed as a product over the symbols, and the calculations are simplified by maximizing the logarithm

$$\sum_j \log P\left[r_j|c_{ij}\right]. \tag{B.4}$$

In the binary case we can simplify the expression by subtracting the logarithm conditioned on the opposite transmitted symbol. This does not change the decision. Using the expression for the Gaussian distribution we get

$$\sum_j \left(-\left(\frac{r_j - c_{ij}}{2\rho}\right)^2 + \left(\frac{r_j + c_{ij}}{2\rho}\right)^2 \right).$$

This result shows that the receiver should maximize the correlation

$$\sum_j r_j c_{ij}, \tag{B.5}$$

or equivalently minimize the error

$$\sum_j c_{ij} \left(c_{ij} - r_j\right). \tag{B.6}$$

Thus when the channel output is quantized using $A-1$ equally spaced thresholds, we can simply use the integers $\{0, 1, .., A-1\}$ as weights for the symbols.

Using such a set of weights, the Viterbi algorithm for convolutional codes, iterative decoding of block codes, and certain other techniques can be modified with these so-called soft decisions.

Appendix C

Tables of minimal polynomials

This appendix provides tables of minimal polynomials of α^i, where α is a primitive element of $\mathbb{F}_{2^m}$.

The table is used in the following way.

1. Choose the field by choosing m.
2. Choose the power of α by choosing i.
3. Read the powers of x from the table.

Example C.0.1. We list the minimal polynomial of α^3 in $\mathbb{F}_8$.

1. Since $\mathbb{F}_8 = \mathbb{F}_{2^3}$, we have $m = 3$.
2. We have α^3, i.e., $i = 3$.
3. We read from the table the powers of x as $(0, 2, 3)$, which means that the polynomial is $x^3 + x^2 + 1$.

$m = 2$

Power of α	Powers of x
$i = 1$	$(0, 1, 2)$

$m = 3$

Power of α	Powers of x	Power of α	Powers of x
$i = 1$	$(0, 1, 3)$	$i = 3$	$(0, 2, 3)$

$m = 4$

$i = 1$	$(0, 1, 4)$	$i = 3$	$(0, 1, 2, 3, 4)$	$i = 5$	$(0, 1, 2)$	$i = 7$	$(0, 3, 4)$

$m = 5$

$i = 1$	$(0, 2, 5)$	$i = 3$	$(0, 2, 3, 4, 5)$	$i = 5$	$(0, 1, 2, 4, 5)$
$i = 7$	$(0, 1, 2, 3, 5)$	$i = 11$	$(0, 1, 3, 4, 5)$	$i = 15$	$(0, 3, 5)$

$m = 6$

$i = 1$	$(0, 1, 6)$	$i = 3$	$(0, 1, 2, 4, 6)$	$i = 5$	$(0, 1, 2, 5, 6)$
$i = 7$	$(0, 3, 6)$	$i = 9$	$(0, 2, 3)$	$i = 11$	$(0, 2, 3, 5, 6)$
$i = 13$	$(0, 1, 3, 4, 6)$	$i = 15$	$(0, 2, 4, 5, 6)$	$i = 21$	$(0, 1, 2)$
$i = 23$	$(0, 1, 4, 5, 6)$	$i = 27$	$(0, 1, 3)$	$i = 31$	$(0, 5, 6)$

$m = 7$

$i = 1$	$(0, 3, 7)$	$i = 3$	$(0, 1, 2, 3, 7)$	$i = 5$	$(0, 2, 3, 4, 7)$
$i = 7$	$(0, 1, 2, 4, 5, 6, 7)$	$i = 9$	$(0, 1, 2, 3, 4, 5, 7)$	$i = 11$	$(0, 2, 4, 6, 7)$
$i = 13$	$(0, 1, 7)$	$i = 15$	$(0, 1, 2, 3, 5, 6, 7)$	$i = 19$	$(0, 1, 3, 6, 7)$
$i = 21$	$(0, 2, 5, 6, 7)$	$i = 23$	$(0, 6, 7)$	$i = 27$	$(0, 1, 4, 6, 7)$
$i = 29$	$(0, 1, 3, 5, 7)$	$i = 31$	$(0, 4, 5, 6, 7)$	$i = 43$	$(0, 1, 2, 5, 7)$
$i = 47$	$(0, 3, 4, 5, 7)$	$i = 55$	$(0, 2, 3, 4, 5, 6, 7)$	$i = 63$	$(0, 4, 7)$

$m = 8$

$i = 1$	$(0, 2, 3, 4, 8)$	$i = 3$	$(0, 1, 2, 4, 5, 6, 8)$	$i = 5$	$(0, 1, 4, 5, 6, 7, 8)$
$i = 7$	$(0, 3, 5, 6, 8)$	$i = 9$	$(0, 2, 3, 4, 5, 7, 8)$	$i = 11$	$(0, 1, 2, 5, 6, 7, 8)$
$i = 13$	$(0, 1, 3, 5, 8)$	$i = 15$	$(0, 1, 2, 4, 6, 7, 8)$	$i = 17$	$(0, 1, 4)$
$i = 19$	$(0, 2, 5, 6, 8)$	$i = 21$	$(0, 1, 3, 7, 8)$	$i = 23$	$(0, 1, 5, 6, 8)$
$i = 25$	$(0, 1, 3, 4, 8)$	$i = 27$	$(0, 1, 2, 3, 4, 5, 8)$	$i = 29$	$(0, 2, 3, 7, 8)$
$i = 31$	$(0, 2, 3, 5, 8)$	$i = 37$	$(0, 1, 2, 3, 4, 6, 8)$	$i = 39$	$(0, 3, 4, 5, 6, 7, 8)$
$i = 43$	$(0, 1, 6, 7, 8)$	$i = 45$	$(0, 3, 4, 5, 8)$	$i = 47$	$(0, 3, 5, 7, 8)$
$i = 51$	$(0, 1, 2, 3, 4)$	$i = 53$	$(0, 1, 2, 7, 8)$	$i = 55$	$(0, 4, 5, 7, 8)$
$i = 59$	$(0, 2, 3, 6, 8)$	$i = 61$	$(0, 1, 2, 3, 6, 7, 8)$	$i = 63$	$(0, 2, 3, 4, 6, 7, 8)$
$i = 85$	$(0, 1, 2)$	$i = 87$	$(0, 1, 5, 7, 8)$	$i = 91$	$(0, 2, 4, 5, 6, 7, 8)$
$i = 95$	$(0, 1, 2, 3, 4, 7, 8)$	$i = 111$	$(0, 1, 3, 4, 5, 6, 8)$	$i = 119$	$(0, 3, 4)$
$i = 127$	$(0, 4, 5, 6, 8)$				

$m = 9$

$i = 1$	$(0, 4, 9)$	$i = 3$	$(0, 3, 4, 6, 9)$	$i = 5$	$(0, 4, 5, 8, 9)$
$i = 7$	$(0, 3, 4, 7, 9)$	$i = 21$	$(0, 1, 2, 4, 9)$	$i = 35$	$(0, 8, 9)$
$i = 63$	$(0, 2, 5, 6, 9)$	$i = 77$	$(0, 3, 6, 8, 9)$	$i = 91$	$(0, 1, 3, 6, 9)$
$i = 119$	$(0, 1, 9)$	$i = 175$	$(0, 5, 7, 8, 9)$		

$m = 10$

$i = 1$	$(0, 3, 10)$

$m = 11$

$\boxed{i = 1 \quad (0, 2, 11)}$

$m = 12$

$\boxed{i = 1 \quad (0, 1, 4, 6, 12)}$

$m = 13$

$\boxed{i = 1 \quad (0, 1, 3, 4, 13)}$

$m = 14$

$\boxed{i = 1 \quad (0, 1, 6, 10, 14)}$

$m = 15$

$\boxed{i = 1 \quad (0, 1, 15)}$

$m = 16$

$\boxed{i = 1 \quad (0, 1, 3, 12, 16)}$

Appendix D

Solutions to Selected Problems

D.1 Solutions to problems in Chapter 1

Problem 1.5.1.

(1) No, the sum of the last two words is not in the code. Or 5 is not a power of 2.

(2) If we add (010110), (011001), and (100101) we get a linear code.

(3) There are many possibilities, e.g., (101010), (010110), (001111).

Problem 1.5.2.

(1) Using row operations we get

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

(2) Since $H_{C^\perp} = G_C$, this is the G that we have.

(3) No, $G(111111)^T = (100) \neq (000)$.

Problem 1.5.3.

(1) The dimension of the code is 4, and therefore the dimension of the dual code is $12 - 4 = 8$.

To get the minimum distance of the code (= minimum weight), at this point we list the $2^4 = 16$ codewords and find $d_{\min} = 6$.

For the dual code we have $H_{C^\perp} = G_C$ and since columns 1, 4 and 5 in G sum to zero and there is no zero column and no two columns are equal, the minimum distance of $C^\perp$ is 3.

(2) The original code can correct two errors and the dual code can correct one error.

Problem 1.5.4.

(1) h_j.

(2) $h_1 + h_4 + h_5$.

(3) $(1,1,1,1,1)$ is a codeword if and only if $H(11111)^T = h_1 + h_2 + h_3 + h_4 + h_5 = 0$.

Problem 1.5.5. Since $1 + \binom{14}{1} + \binom{14}{2} + \binom{14}{3} = 470$ and $470 < 2^9$, but $470 > 2^8$, the GV-bound says that there exists a $(15, 6, 5)$ code.
Actually there is a $(15, 7, 5)$ code, as we will see.

Problem 1.5.6.

(1) Parameters of the binary Hamming codes (see Definition 1.2.5).

m	3	4	5	8
$n = 2^m - 1$	7	15	31	255
$k = 2^m - 1 - m$	4	11	26	247
d	3	3	3	3

(2) Parameters of the extended binary Hamming codes (see Defintion 1.2.6).

m	3	4	5	8
$n = 2^m$	8	16	32	256
$k = 2^m - 1 - m$	4	11	26	247
d	4	4	4	4

Problem 1.5.7.

(1) The code is a $(7, 4, 3)$ code, so the dual code has dimension 3. By listing all the $2^3 = 8$ codewords we see that the minimum weight (= the minimum distance) is 4.

(2) By looking at all the eight codewords we see that all the distances are 4.

(3) A generator matrix for the dual Hamming code $(2^m - 1, m)$ has as columns all the nonzero binary m-vectors.

The minimum distance (= the minimum weight) of this code is 2^{m-1}, and any two codewords have this distance.

There are many ways of seeing that; here is a proof by induction on m.

For $m = 1$ the statement is trivial, since the only words are 0 and 1. Suppose the statement is true for $m = i$; to prove that it is also true for $m = i + 1$, note that

$$G_{i+1} = \begin{bmatrix} G_i & G_i & \underline{0} \\ 0\ldots0 & 1\ldots1 & 1 \end{bmatrix}.$$

From this the statement follows easily.

Problem 1.5.8.

(1) $B(2)$ has generator matrix

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

From this the result is trivial.

(2) This follows directly from Problem 1.5.8.

(3) Since the distance between any two words is $\frac{n}{2}$, this is obvious (except that the all 1s word and the all -1s word are not orthogonal).

Problem 1.5.9.

(1) We get

$$\begin{aligned} n^* &= n - 1, \\ k^* &= k \text{ (since } d \geq 2), \\ d^* &= \begin{cases} d, & \text{if all minimum weight words} \\ & \text{have a zero in the deleted position,} \\ d - 1, & \text{otherwise.} \end{cases} \end{aligned}$$

(2) We get

$$\begin{aligned} n^* &= n - 1, \\ k^* &= k - 1, \\ d^* &\geq d. \end{aligned}$$

Problem 1.5.10. We get

$$k_{\text{ext}} = k$$

and

$$d_{\text{ext,min}} = \begin{cases} d, & \text{if } d \text{ is even,} \\ d + 1, & \text{otherwise.} \end{cases}$$

The parity check matrix for the extended code is obtained by adding a column of zeroes to H and a row of 1s, i.e.,

$$H_{\text{ext}} = \begin{bmatrix} H_c & & 0 \\ & & \vdots \\ & & 0 \\ 1 & \dots & 1 \end{bmatrix}$$

Problem 1.5.11.

(1) $c_j = 0$, if not satisfied for all the 2^k codewords, gives one new equation, so the dimension drops by one (i.e., there are 2^{k-1} such codewords).

(2) $\sum_{c \in C} w(c)$ = total number of 1s $\leq n \cdot 2^{k-1}$ by (1).

(3) Since $\sum_{c \in C} w(c) \geq (2^k - 1)d_{\min}$, the result follows.

Problem 1.5.14. No, since that would give a word of weight 4 in the code.

Problem 1.5.15. Just avoid the sum of the two first columns in H.

Problem 1.5.16. The Hamming bound gives $k \leq 8$.

Problem 1.5.17. $1 + 28z^2 + 70z^4 + 28z^6 + z^8$.

D.2 Solutions to problems in Chapter 2

Problem 2.6.1.

(1) All elements have an additive inverse, therefore the sum of all elements equals 0.

(2) The elements 16 and 1 are their own multiplicative inverses and all other elements have another element as an multiplicative inverse, therefore the product of all nonzero elements is $1 \cdot 1 \cdots 16 = 16$.

(3) $\operatorname{ord}(2) = 8$.

(4) The possible orders are $1, 2, 4, 8, 16$.

(5) $\operatorname{ord}(1) = 1, \operatorname{ord}(16) = 2, \operatorname{ord}(4) = 4, \operatorname{ord}(2) = 8, \operatorname{ord}(3) = 16$.

(6) Eight primitive elements.

(7) No solutions.

(8) 2 and 14.

Problem 2.6.2.

(1) If $a \neq 0$, we get $a^{-1}(ab) = b = 0$.

(2) $2 \cdot 2 = 0$, so by (1) it cannot be a field!

Problem 2.6.3. If $a^i = 1$, we get -1 and else 0, since $a^{q-1} = 1$.

Problem 2.6.4. $x^3 + x + 1$ and $x^3 + x^2 + 1$.

Problem 2.6.5.

binary 3-tuple	power of α	polynomial
000		0
100	α^0	1
010	α	x
001	α^2	x^2
110	α^3	$x+1$
011	α^4	x^2+x
111	α^5	x^2+x+1
101	α^6	x^2+1

Problem 2.6.6. x^4+x+1.

Problem 2.6.7.

(1) If $m=1$, the sum is 1. For $m>1$ the sum of all binary vectors is $(0,0,\dots,0)$.

(2) All elements have a multiplicative inverse, except for 1, which is its own inverse, therefore the product $1\cdot 1\cdots 1=1$.

(3) From Example 2.3.3 we get that α^i is primitive iff $\gcd(i,15)=1$.

Problem 2.6.8.

(1) z^4+z^3+1 has zeroes $\alpha^7,\alpha^{11},\alpha^{13}$ and α^{14}.

(2) z^4+z^2+z has only 0.

Problem 2.6.9. $m=\operatorname{lcm}(2,3)=6$.

Problem 2.6.10. 1, 31, and 1, 3, 7, 9, 21, 63.

Problem 2.6.13. $x^9-1=(x+1)(x^2+x+1)(x^6+x^3+1)$.

Problem 2.6.14.

$$\begin{aligned} x^{73}-1 = {} & (x+1)(x^9+x^7+x^4+x^3+1)(x^9+x^4+x^2+x+1) \\ & \cdot(x^9+x^8+1)(x^9+x^6+x^5+x^2+1)(x^9+x^8+x^6+x^3+1) \\ & \cdot(x^9+x^6+x^3+x+1)(x^9+x+1)(x^9+x^8+x^7+x^5+1). \end{aligned}$$

Problem 2.6.15.

$$\begin{aligned} x^{85}-1 = {} & (x+1)(x^8+x^6+x^5+x^4+x^2+x+1) \\ & \cdot(x^8+x^7+x^5+x^4+x^3+x^2+1) \\ & \cdot(x^8+x^7+x^6+x^4+x^2+x+1)(x^8+x^7+x^3+x+1) \\ & \cdot(x^8+x^5+x^4+x^3+x^2+x+1) \\ & \cdot(x^8+x^7+x^6+x^5+x^4+x^3+1)(x^8+x^5+x^4+x^3+1) \\ & \cdot(x^4+x^3+x^2+x+1)(x^8+x^7+x^6+x^4+x^3+x^2+1) \\ & \cdot(x^9+x^7+x^5+x^2+1)(x^9+x^6+x^3+x+1). \end{aligned}$$

Problem 2.6.16. $x^{18} - 1 = (x^9 - 1)^2$.

Problem 2.6.17. Yes, it is on the list, see Appendix C, $m = 8, i = 5$.

D.3 Solutions to problems in Chapter 3

Problem 3.5.1.

(1) The entropy of the source is $H = 1.5$ bits.

(2) $H_{\max} = \log(3) = 1.585$ bits.

(3) $C = \{0, 10, 11\}$.

(4) $\left[\frac{1}{2}, \frac{1}{4}, \frac{1}{4}\right]$ for $[0, 1, -1]$. V is not memoryless. There is a one-to-one mapping between the binary and the ternary sequences.

Problem 3.5.2.

(1) When the input distribution is $\left(\frac{1}{2}, \frac{1}{2}\right)$, the output distribution becomes $P(Y) = \left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$.

(2) $I = H(Y) - H(Y|X) = \log(3)/2 - 2/3 = 0.126$ bits.

(3) $C = I$; this is the maximizing distribution.

Problem 3.5.3.

(1) $C = 1 - H(0.05) = 0.714$ bits.

(2) The average number of errors is $t_a = 256 \cdot 0.05 = 12.8$.

(3) The Hamming bound gives: $\log_2\left(\binom{256}{t}\right) < 256 - 160$ thus $t = 19$.

(4) $P(e) = P[> 19 \text{ errors}] = \binom{256}{20}p^{20}(1-p)^{236} + \binom{256}{21}p^{21}(1-p)^{235} + \cdots = 0.03$.

(5) The Gilbert bound for the same parameters is $d_g = 21$ by a similar calculation. (There is a code with $d = 26$.)

Problem 3.5.6.

(1)
$$\mu = np = 16 \cdot 0.01 = 0.16$$
and
$$\sigma = \sqrt{np(1-p)} = 0.401$$

(2) From (3.2) we get $0.852, 0.138, 0.010, 0.0005, 0.00002$.

(3) From (3.3) we get $0.852, 0.134, 0.011, 0.00058, 0.00002$. Thus the agreement is quite good for these parameters.

(4) From (3.10) we get $P_{\text{fail}} = 1 - 0.852 - 0.138 = 0.011$ since $t = 1$ here.

(5) So it is the probability of getting three errors i.e. 0.0005.

Problem 3.5.7.

(1) The weight distribution shows $d = 5$, thus $t = 2$.

(2) $P_{\text{fail}} = P(> 2) = 0.0004$.

(3) 3 bit errors cause a decoding error if the received vector is at distance 2 from a codeword of weight 5. Thus there are $18 \cdot \binom{5}{3} = 180$ such error patterns (exact result).

The total number of weight-3 words is $\binom{15}{3} = 455$. The number could also be found using (3.12) with $j = 3, w = 5, s = 2, i = 0$.

(4) $P_{\text{err}} = 180 \cdot 0.01^3 \cdot 0.99^{12} = 0.00016$. The next term for $j = 4$ gives $90 + 450$ error patterns, but the total probability is much smaller.

Problem 3.5.8.

(1) The minimum distance is 6, so the code corrects two errors.

(2) Assuming that only two errors are corrected, we have

$$P_{\text{fail}} \approx P(3) = \binom{16}{3} \cdot p^3 \cdot (1-p)^{13} = 0.0005.$$

(3) From (3.11) we get $T(4, 2, 6) = 15$ and $T(6, 2, 6) = 6 \cdot 10 = 60$.

Problem 3.5.9.

(1) $np = 2.56$.

(2) $P(9) = 0.00094$, $P(10) = 0.00024$, $P(11) = 0.00005$, so $P(> 8) = 0.00125$.

(3) We get about 0.00132, all terms are slightly larger.

Problem 3.5.11.

(1) $C = 1 - H(0.1) = 0.531$.

(2) $Z = 0.6$, $R_0 = 1 - \log(1.6) = 0.322$ From (3.22) we get $E(R) = 0.322 - R$. To apply (3.24), we need $H(\delta) = 1 - R$, and $T_{0.1}(\delta) = \log(10) - (1-\delta)\log(9)$. The point where the two bounds meet is found from $w'/n = 3/8$, $\delta_c = 1/4$ Thus the rate is $R = 1 - H(1/4) = 0.188$.

(3) For $R = 1/3$ we have $\delta = 0.174$. Thus $T(1/3) = 0.704$ and $E(1/3) = 0.037$. For a code of length 1024 we get the estimated error probability as $2^{-nE} = 4 \cdot 10^{-12}$.

D.4 Solutions to problems in Chapter 4

Problem 4.5.1.

(1) There are $(4, 1, 4)$, $(4, 3, 2)$, and $(4, 4, 1)$ binary codes.

(2) A $(4, 2, 3)$ ternary code could have

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

Problem 4.5.2.

(1)

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 2 & 6 & 4 & 5 \\ 1 & 2 & 4 & 1 & 2 & 4 \\ 1 & 6 & 1 & 6 & 1 & 6 \end{bmatrix}.$$

(2) Using the polynomials

$$\begin{aligned} 2(x-3)(x-2)(x-6), \\ (x-1)(x-2)(x-6), \\ 2(x-1)(x-3)(x-6), \\ 2(x-1)(x-3)(x-2), \end{aligned}$$

we get

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 6 & 2 \\ 0 & 1 & 0 & 0 & 2 & 2 \\ 0 & 0 & 1 & 0 & 2 & 5 \\ 0 & 0 & 0 & 1 & 5 & 6 \end{bmatrix}.$$

(3) $d_{\min} = n - k + 1 = 6 - 4 + 1 = 3$.

(4) Addition of the second and fourth row of G from (1) gives $(2, 2, 3, 5, 5, 4)$, so $r = (2, 4, 3, 5, 5, 4)$.

(5) $l_0 = 4$ and $l_1 = 1$, so we have seven coefficients (and six equations).

(6) $Q_0(x) = 4x^4 + 3x^3 + 4x^2 + 3x$, $Q_1(x) = x + 4$, so $-\frac{Q_0(c)}{Q_1(x)} = x + x^3$.

(7)

$$H = \begin{bmatrix} 1 & 3 & 2 & 6 & 4 & 5 \\ 1 & 2 & 4 & 1 & 2 & 4 \end{bmatrix}.$$

(8) A single error would give a syndrome that is a multiple of a column of H.

Problem 4.5.3.

(1)

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

(2) $(8, 3, 6)$.

(3) With $r = (0, \alpha^4, \alpha, 0, 0, \alpha^2, \alpha)$ one gets $Q_0(x) = x^4 + \alpha^2 x^3 + \alpha^2 x^2 + x$, $Q_1(x) = x^2 + \alpha^6 x + 1$, and $f(x) = x + x^2$.

(4)

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

(5) One error would give $\frac{S_2}{S_1} = \frac{S_3}{S_2} = \frac{S_4}{S_3}$.

Problem 4.5.4.

(1) $n = 10, k = 3$, so with $l = 2$ we can correct 4 errors.

(2) The system of equations is

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 5 & 10 \\ 1 & 4 & 5 & 9 & 3 & 1 \\ 1 & 8 & 9 & 6 & 4 & 10 \\ 1 & 5 & 3 & 4 & 9 & 1 \\ 1 & 10 & 1 & 10 & 1 & 10 \\ 1 & 9 & 4 & 3 & 5 & 1 \\ 1 & 7 & 5 & 2 & 3 & 10 \\ 1 & 6 & 3 & 7 & 8 & 10 \end{bmatrix} \begin{bmatrix} Q_{00} \\ Q_{01} \\ Q_{02} \\ Q_{03} \\ Q_{04} \\ Q_{05} \end{bmatrix}$$

$$+ \begin{bmatrix} 0 &&&&&&&&& \\ & 0 &&&& \mathbf{0} &&&& \\ && 6 &&&&&&& \\ &&& 9 &&&&&& \\ &&&& 1 &&&&& \\ &&&&& 6 &&&& \\ &&&&&& 0 &&& \\ &&&&&&& 0 && \\ &&& \mathbf{0} &&&&& 0 & \\ &&&&&&&&& 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 4 & 5 & 9 \\ 1 & 8 & 9 & 6 \\ 1 & 5 & 3 & 4 \\ 1 & 10 & 1 & 10 \\ 1 & 9 & 4 & 3 \\ 1 & 7 & 5 & 2 \\ 1 & 3 & 9 & 5 \\ 1 & 6 & 3 & 7 \end{bmatrix} \begin{bmatrix} Q_{10} \\ Q_{11} \\ Q_{12} \\ Q_{13} \end{bmatrix}$$

$$+ \begin{bmatrix} 0 &&&&&&&&& \\ & 0 &&&& \mathbf{0} &&&& \\ && 3 &&&&&&& \\ &&& 4 &&&&&& \\ &&&& 1 &&&&& \\ &&&&& 3 &&&& \\ &&&&&& 0 &&& \\ &&&&&&& 0 && \\ &&& \mathbf{0} &&&&& 0 & \\ &&&&&&&&& 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 4 \\ 1 & 8 \\ 1 & 5 \\ 1 & 10 \\ 1 & 9 \\ 1 & 7 \\ 1 & 3 \\ 1 & 6 \end{bmatrix} \begin{bmatrix} Q_{20} \\ Q_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

and it is easy to check that 000000, $0 - 2, 3 - 1, 10$ solves this system.

(3) $Q(x, y) = y(y - (x^2 - 3x + 21))$, so the possible codewords are $(0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ and $(0, 0, 6, 9, 1, 6, 1, 8, 2, 9)$.

Problem 4.5.5.

(1) $Q_1(x) = x^2 + \alpha^6 x + 1 = (x - \alpha^3)(x - \alpha^4)$.

(2) Yes.

(3) α^4 and α^2.

D.5 Solutions to problems in Chapter 5

Problem 5.5.1.

(1) $x^9 - 1 = (x^3 - 1)(x^6 + x^3 + 1)$.

(2) $k = 3$.

(3)

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

(4) Yes, since $x^8 + x^6 + x^5 + x^3 + x^2 + 1 = (x^2 + 1)(x^6 + x^3 + 1)$.

(5) $d_{\min} = 3$ (List the eight codewords).

Problem 5.5.2.

(1) $k = 15 - 5 = 10$.

(2) No, $g(x)$ does not divide.

(3) The possible generator polynomials are

$$\begin{aligned} &(x^3 + 1)(x^4 + x + 1), \\ &(x^3 + 1)(x^4 + x^3 + 1), \\ &(x^3 + 1)(x^4 + x^3 + x^2 + x + 1). \end{aligned}$$

(4) 2^5, but two of them are trivial, so there are 30.

Problem 5.5.3.

(1)

$$h(x) = \frac{x^{15} - 1}{x^4 + x + 1} = x^{11} + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1,$$

so

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

(2) $d_{\min} = 3$.

(3) C is the $(15, 11, 3)$ Hamming code.

(4) The dual code has dimension 4.

Problem 5.5.4. $g(x)|c(x)$ so $0 = c(1) = c_0 + c_1 + \cdots + c_{n-1} \Leftrightarrow c$ has even weight.

Problem 5.5.5. Since C has a word of odd weight, $x + 1$ does not divide $g(x)$, and therefore $g(x)|\frac{x^n-1}{x+1} = 1 + x + \cdots + x^{n-1}$, so $(1111\ldots1)$ is a codeword.

Problem 5.5.6.

(1) $g(x)$ has among its zeroes $\beta, \beta^2, \beta^3, \beta^4$, so $d_{\min} \geq 5$.

(2) $g^{\perp}(x) = (x-1)m_{\beta}(x)m_{\beta^7}(x)m_{\beta^3}(x)$ so,

(3) $d^{\perp}_{\min} \geq 6$.

Problem 5.5.7. $g(x)$ has among its zeroes $\alpha^{-2}, \alpha^{-1}, 1, \alpha, \alpha^2$, hence $d_{\min} \geq 6$, but by the Hamming bound there is no $(63, 50, 7)$ code, so it is exactly 6.

Problem 5.5.8. With $g(x) = m_{\alpha}(x)m_{\alpha^3}(x)m_{\alpha^5}(x)$, α primitive in $\mathbb{F}_{2^5}$, we get a $(31, 16, 7)$ code and this is best possible.

D.6 Solutions to problems in Chapter 6

Problem 6.4.1.

(1) The transmission efficiency of the frame is $\eta_{\text{f}} = \frac{30\cdot0.5}{32\cdot(1-H(0.01))} = 0.510$, since the frame check word is part of the channel code.

(2) This follows from the linearity of the code.

(3) The probability that a word:

a) Follows from the binomial distribution, $P[< 4 \text{ errors}] = 0.99971$.

b) Can be found from the previous answers as $\simeq 1 - 0.99971 = 0.00029$.

c) We have from Chapter 3 that there are 620 codewords at distance 8 and most decoding errors occur if 5 errors brings us within distance 3 of one of these. Thus the probability is about $620 \cdot 56 \cdot p^5(1-p)^{27} = 0.0000026$. Thus these events do not significantly change the previous answer.

(4) $P_{\text{fe}} \simeq 1 - 0.99971^{127} = 0.036$. An undetected error would occur if two words were wrongly decoded with errors in the same positions. $P_{\text{ue}} \simeq 127 \cdot 126/2 \cdot 0.0000026 \cdot 0.01^5 = 2 \cdot 10^{-12}$.

(5) $P_{\text{fe}} \simeq 0.00066$, two words are not decoded. $P_{\text{ue}} \simeq 0.000012$, one word decoded in error, another not decoded.

(6) A frame error occurs if there are 5 nondecoded words (or two errors and one non-decoded word), an undetected error if three errors are corrected to a wrong RS word (which happens with probability about 1/2). $P_{\text{fe}} = 5 \cdot 10^{-10}$, $P_{\text{ue}} = 3 \cdot 10^{-12}$.

D.7 Solutions to problems in Chapter 7

Problem 7.7.1.

(1) 11.10.10.10.00.00.10.11.00.00.00.

(2) $M = 3$.

(3) 8.

Problem 7.7.2.

(1) As noted in Section 7.1, the sequence h for a code of rate $\frac{1}{2}$ is found by reversing g. Since g is symmetric here, $h = g$.

(2) The nonzero part of the syndrome for a single error is (1011) for the first position in a pair and (1101) for the second position.

(3) Adding the two syndromes from the previous question we find that the error 11.00.00.... gives the syndrome (0110...).

(4) From the previous questions it follows that at least three errors are needed to give the syndrome. One possibility is 10.11.

Problem 7.7.3. Code from Problem 7.7.1:

(1) All row distances are 6, which is also the free distance.

(2) 2.

(3) Since $g = 11.10.01.11$ is a codeword, 11.10.00.00 and 00.00.01.11 have the same syndrome, and at most one of them can be correctly decoded. The error pattern 11.00.00.11 would be decoded as 00.10.01.00

D.8 Solutions to problems in Chapter 8

Problem 8.4.1.

(1) (49, 16, 9).

(2) 3. At least two by Lemma 8.1.1, less than four from the comments in the proof.

(3) 49, since there are seven words of weight three in the Hamming code.

(4) There are three rows in the parity check matrix of the Hamming code. These are copied to give parity checks on the seven rows and seven columns; however, the last three sets of checks are linear combinations of other rows. Thus we get 33 rows, which is the right number.

(5) a) They will be corrected as single errors.

b) One error is corrected by the column code, in the other column two errors cause a decoding error. The three positions in this column are corrected by the row codes.

Problem 8.4.2.

(1) (256, 144, 25).

(2) 12.

(3) 6.

Problem 8.4.4.

(1) (35, 24, 4).

(2) Write the codewords as 5 by 7 arrays. Any cyclic shift of rows or columns gives a codeword. By taking a shift of both rows and columns, a symbol cycles through all 35 positions. It is important for this argument that n_1, and n_2 are relatively prime.

(3) The generator polynomial has degree 11, and since the weights are even, one factor is $x + 1$. The factors of $x^{35} - 1$ are 2 polynomials of degree 12, 1 of degree 4, and 2 of degree 3. Thus the generator polynomial includes the last 3.

Problem 8.4.5.

(1) $(60, 40, \geq 6)$.

(2) $(75, 40, \geq 12)$.

(3) $(135, 80, \geq 12)$.

Problem 8.4.6.

(1) From the binomial distribution $P_{\mathrm{f}} = 4 \cdot 10^{-4}$. The Poisson approximation with mean value 1 gives $6 \cdot 10^{-4}$.

(2) By Theorem 8.2.3, $P_{\mathrm{ue}} \approx \frac{P_{\mathrm{f}}}{120} = 3.3 \cdot 10^{-6}$.

(3) The probability of more than one error in a (14, 10, 3) codeword is 0.0084.

(4) For one RS code we find $1.75 \cdot 10^{-7}$ (by the same calculation as in Problem 8.4.1). P_{f} for the concatenated code is at most this number multiplied by $I = 2$, i.e., $3.5 \cdot 10^{-7}$ (Theorem 8.2.4).

D.9 Solutions to problems in Chapter 9

Problem 9.6.1.

(1) $r(x) = x^3 + \alpha^2 x^2 + \alpha x$, so we know already that the error locations are $\alpha, \alpha^2, \alpha^3$ with the corresponding error values $\alpha, \alpha^2, 1$,

$$\begin{aligned}
S_1 &= r(\alpha) = \alpha^2 + \alpha^4 + \alpha^3 = \alpha^{12},\\
S_2 &= r(\alpha^2) = \alpha^3 + \alpha^6 + \alpha^6 = \alpha^3,\\
S_3 &= r(\alpha^3) = \alpha^4 + \alpha^8 + \alpha^9 = \alpha^6,\\
S_4 &= r(\alpha^4) = \alpha^5 + \alpha^{10} + \alpha^{12} = \alpha^{11},\\
S_5 &= r(\alpha^5) = \alpha^6 + \alpha^{12} + 1 = \alpha,\\
S_6 &= r(\alpha^6) = \alpha^7 + \alpha^{14} + \alpha^3 = \alpha^9.
\end{aligned}$$

Hence, $S(x) = \alpha^{12}x^5 + \alpha^3 x^4 + \alpha^6 x^3 + \alpha^{11}x^2 + \alpha x + \alpha^9$.

The Euclidean algorithm on x^6 an $S(x)$ gives:

i	g_i	r_i	q_i
-1	0	x^6	–
0	1	$S(x)$	–
1	$\alpha^3x+\alpha^9$	$\alpha^8x^4+\alpha^3x^3+\alpha^8x^2+\alpha^3x+\alpha^3$	$\alpha^3x+\alpha^9$
2	$\alpha^7x^2+\alpha^2x+\alpha^{10}$	$\alpha^9x^3+\alpha^5x^2+\alpha^4$	$\alpha^4x+\alpha^{11}$
3	$\alpha^6x^3+\alpha^2x^2+\alpha^4x+\alpha^{12}$	$\alpha^{13}x^2+\alpha^6$	$\alpha^{14}x+\alpha^{13}$

so the error locator is $g_3(x) = \alpha^6x^3 + \alpha^2x^3 + \alpha^4x + \alpha^{12}$, which indeed has α, α^2, and α^3 as zeroes. Since $g_3' = \alpha^6x^2 + \alpha^4$, Formula (9.2) gives $e_1 = \alpha^{-7} \cdot \frac{\alpha^{15}+\alpha^6}{\alpha^8+\alpha^4} = \alpha$, $e_2 = \alpha^{-14} \cdot \frac{\alpha^{17}+\alpha^6}{\alpha^{10}+\alpha^4} = \alpha^2$, $e_3 = \alpha^{-21} \cdot \frac{\alpha^{19}+\alpha^6}{\alpha^{12}+\alpha^4} = 1$.

(2) $r(x) = \alpha^2x^3 + \alpha^7x^2 + \alpha^{11}x + \alpha^6$.

$$\begin{aligned}
S_1 &= r(\alpha) = \alpha^5 + \alpha^9 + \alpha^{12} + \alpha^6 = \alpha^{12},\\
S_2 &= r(\alpha^2) = \alpha^8 + \alpha^{11} + \alpha^{13} + \alpha^6 = \alpha^9,\\
S_3 &= r(\alpha^3) = \alpha^{11} + \alpha^{13} + \alpha^{14} + \alpha^6 = \alpha^5,\\
S_4 &= r(\alpha^4) = \alpha^{14} + 1 + 1 + \alpha^6 = \alpha^8,\\
S_5 &= r(\alpha^5) = \alpha^2 + \alpha^2 + \alpha + \alpha^6 = \alpha^{11},\\
S_6 &= r(\alpha^6) = \alpha^5 + \alpha^4 + \alpha^2 + \alpha^6 = \alpha^{13}.
\end{aligned}$$

So $S(x) = \alpha^{12}x^5 + \alpha^9x^4 + \alpha^5x^3 + \alpha^8x^2 + \alpha^{11}x + \alpha^{13}$.

The Euclidean algorithm on x^6 an $S(x)$ gives:

i	g_i	r_i	q_i
-1	0	x^6	–
0	1	$S(x)$	–
1	α^3x+1	$\alpha^{12}x^4+\alpha^3x^3+\alpha^6x^2+\alpha^6x+\alpha^{13}$	α^3x+1
2	$\alpha^3x^2+\alpha^9x+\alpha$	$x^3+\alpha^{11}x^2+\alpha^2x+\alpha^{14}$	$x+\alpha^4$
3	$x^3+\alpha^{11}x^2+\alpha^{11}x+\alpha^3$	$\alpha^{12}x^2+\alpha^4x+\alpha$	$\alpha^{12}x+\alpha^{13}$

But g_3 does not have any zeroes in $\mathbb{F}_{16}$, so more than three errors must have occured there.

Problem 9.6.2.

(1) $r(x) = x^8 + x^5 + x^2 + 1$.

$$\begin{aligned}
S_1 &= r(\alpha) = \alpha^8 + \alpha^5 + \alpha^2 + 1 = \alpha^5,\\
S_2 &= r(\alpha^2) = \alpha^{10},\\
S_3 &= r(\alpha^3) = 1 + \alpha^6 + 1 + \alpha^9 = \alpha^5,\\
S_4 &= r(\alpha^4) = \alpha^5,\\
S_5 &= r(\alpha^5) = \alpha^{10} + \alpha^{10} + \alpha^{10} + 1 = \alpha^5,\\
S_6 &= r(\alpha^6) = \alpha^{10}.
\end{aligned}$$

Hence, $S(x) = \alpha^5x^5 + \alpha^{10}x^4 + \alpha^5x^3 + \alpha^5x^2 + \alpha^5x + \alpha^{10}$.

The Euclidean algorithm on x^6 an $S(x)$ gives:

i	g_i	r_i	q_i
-1	0	x^6	–
0	1	$S(x)$	–
1	$\alpha^{10}x + 1$	$\alpha^5x^4 + \alpha^{10}x^3 + \alpha^{10}x^2 + \alpha^{10}$	$\alpha^{10}x + 1$
2	$\alpha^{10}x^2 + x + 1$	$x^3 + \alpha^5x^2 + x + \alpha^{10}$	x
3	$x^3 + \alpha^5x^2 + x + 1$	$x^2 + x + \alpha^{10}$	α^5x

The zeroes of $g_3(x)$ are α, α^4 and α^{10}, and therefore $e(x) = x^{10} + x^4 + x$.

(2) $r(x) = x^{11} + x^7 + x^3 + x^2 + x + 1$.

$$\begin{aligned}
S_1 &= r(\alpha) = \alpha^{11} + \alpha^7 + \alpha^3 + \alpha^2 + \alpha + 1 = \alpha^9,\\
S_2 &= r(\alpha^2) = \alpha^3,\\
S_3 &= r(\alpha^3) = \alpha^3 + \alpha^6 + \alpha^9 + \alpha^6 + \alpha^3 + 1 = \alpha^7,\\
S_4 &= r(\alpha^4) = \alpha^6,\\
S_5 &= r(\alpha^5) = \alpha^{10} + \alpha^5 + 1 + \alpha^{10} + \alpha^5 + 1 = 0,\\
S_6 &= r(\alpha^6) = \alpha^{14}.
\end{aligned}$$

Hence, $S(x) = \alpha^9x^5 + \alpha^3x^4 + \alpha^7x^3 + \alpha^6x^2 + \alpha^{14}$.

The Euclidean algorithm on x^6 an $S(x)$ gives:

i	g_i	r_i	q_i
-1	0	x^6	–
0	1	$S(x)$	–
1	$\alpha^6x + 1$	$\alpha^8x^4 + \alpha^2x^3 + \alpha^6x^2 + \alpha^5x + \alpha^{14}$	$\alpha^6x + 1$
2	$\alpha^7x^2 + \alpha x + 1$	$x + \alpha^{14}$	αx

Here $g_2(x)$ has α^{10} and α^{13} as zeroes, so $e(x) = x^{10} + x^{13}$ and we decode into $x^{13} + x^{11} + x^{10} + x^7 + x^3 + x^2 + x + 1$.

D.10 Solutions to problems in Chapter 10

Problem 10.7.1. The parity bits are (00011001001).
The encoding can be interpreted as correcting erasures on the last $n - k$ positions.
The decoded word is (110110110110110110110).

Problem 10.7.2. The code has $n = 12$ symbols and 8 parity checks, 7 of these independent. So it is a (12, 5, 4) code:

$$\begin{bmatrix}
1&1&1&0&0&0&0&0&0&0&0&0\\
1&0&0&1&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&1&0&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0\\
0&0&0&1&0&0&0&1&0&1&0&0\\
0&0&0&0&1&0&1&0&0&0&1&0\\
0&0&0&0&0&1&0&0&1&0&0&1\\
0&0&0&0&0&0&0&0&0&1&1&1
\end{bmatrix}.$$

A tree code can have four parity checks and nine symbols. A single error is identified by the two parity checks that fail. They can be in the same tree code, or there may be one in each code.

Problem 10.7.3. Three errors are corrected, but the result depends on the order in which the positions are considered.
With message passing the weights indicate that three errors have occurred, but there is not a unique result.

D.11 Solutions to problems in Chapter 11

Problem 11.3.1.

(1) $(64, 22, \geq 37)$.

(2) Let α be a primitive element of $\mathbb{F}_{16}$ with $\alpha^4 + \alpha + 1 = 0$. Then the 64 points on the curve $x^5 + y^4 + y = 0$ are

$$\begin{gathered}
(0,0), (0,1), (0,\alpha^5), (0,\alpha^{10});\\
(1,\alpha), (\alpha^3,\alpha), (\alpha^6,\alpha), (\alpha^{12},\alpha), (\alpha^9,\alpha);\\
(1,\alpha^2), (\alpha^3,\alpha^2), (\alpha^6,\alpha^2), (\alpha^{12},\alpha^2), (\alpha^9,\alpha^2);\\
(1,\alpha^4), (\alpha^3,\alpha^4), (\alpha^6,\alpha^4), (\alpha^{12},\alpha^4), (\alpha^9,\alpha^4);\\
(1,\alpha^8), (\alpha^3,\alpha^8), (\alpha^6,\alpha^8), (\alpha^{12},\alpha^8), (\alpha^9,\alpha^8);\\
(\alpha,\alpha^6), (\alpha^4,\alpha^6), (\alpha^7,\alpha^6), (\alpha^{10},\alpha^6), (\alpha^{13},\alpha^6);\\
(\alpha,\alpha^7), (\alpha^4,\alpha^7), (\alpha^7,\alpha^7), (\alpha^{10},\alpha^7), (\alpha^{13},\alpha^7);\\
(\alpha,\alpha^9), (\alpha^4,\alpha^9), (\alpha^7,\alpha^9), (\alpha^{10},\alpha^9), (\alpha^{13},\alpha^9);\\
(\alpha,\alpha^{13}), (\alpha^4,\alpha^{13}), (\alpha^7,\alpha^{13}), (\alpha^{10},\alpha^{13}), (\alpha^{13},\alpha^{13});\\
(\alpha^2,\alpha^3), (\alpha^5,\alpha^3), (\alpha^8,\alpha^3), (\alpha^{11},\alpha^3), (\alpha^{14},\alpha^3);\\
(\alpha^2,\alpha^{11}), (\alpha^5,\alpha^{11}), (\alpha^8,\alpha^{11}), (\alpha^{11},\alpha^{11}), (\alpha^{14},\alpha^{11});\\
(\alpha^2,\alpha^{12}), (\alpha^5,\alpha^{12}), (\alpha^8,\alpha^{12}), (\alpha^{11},\alpha^{12}), (\alpha^{14},\alpha^{12});\\
(\alpha^2,\alpha^{14}), (\alpha^5,\alpha^{14}), (\alpha^8,\alpha^{14}), (\alpha^{11},\alpha^{14}), (\alpha^{14},\alpha^{14}),
\end{gathered}$$

and the 22 polynomials in which one evaluates these points are:

$$\begin{gathered}
1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y,\\
x^2y^2, xy^3, y^4, x^4y, x^3y^2, x^2y^3, xy^4, y^5, x^4y, x^3y^3.
\end{gathered}$$

This gives a 22×64 generator matrix for the code.

(3) $(64, 42, \geq 17)$.

(4) Since $H(22)^\perp = H(42)$ we can copy the previous method.

Problem 11.3.3.

$$\begin{aligned}&x^{q+1}-y^q-y\\&=(a(x)+yb(x,y))(c(x)+yd(x,y))\\&=a(x)c(x)+y(b(x,y)c(x)+d(x,y)a(x))+y^2b(x,y)d(x,y),\end{aligned}$$

and therefore $a(x)c(x)=x^{q+1}$. Hence, $a(x)=x^{q+1-i}$, $c(x)=x^i$, but this implies

$$b(x,y)x^i+d(x,y)x^{q+1-i}+yb(x,y)d(x,y)=-y^{q-1}-1,$$

and so $i=0$ and $b(x,y)=-1$ or $i=q+1$ and $d(x,y)=-1$. In the first case we get $d(x,y)=y^{q-2}$ and in the second case $b(x,y)=y^{q-1}$. If we insert these in the original expression we get a contradiction.

Problem 11.3.4. If $\alpha^{(q-1)(a_1+a_2)}=1$, we have $q^2-1\big|(q-1)(a_1+a_2)$, and hence $a_1+a_2=(q+1)$. So the relevant sum in this case is

$$\begin{aligned}&\sum_{i=0}^{q-2}\alpha^{i(q+1)}\sum_{\beta^q+\beta=\alpha^{i(q+1)}}\beta^{b_1+b_2}\\&=\sum_{i=0}^{q-2}\alpha^{i(q+1)}(\alpha^{i(q+1)})^{b_1+b_2}\sum_{\beta^q+\beta=1}\beta^{b_1+b_2}\\&=\frac{\alpha^{(q+1)(1+b_1+b_2)(q-1)}-1}{\alpha^{(q+1)(1+b_1+b_2)}-1}\sum_{\beta^q+\beta=1}\beta^{b_1+b_2}=0,\end{aligned}$$

when the denominator is nonzero.
In the remaining case we get with $1\le a\le q-1$

$$\begin{aligned}-\sum_{\beta^q+\beta=1}\beta^{a(q-1)-1}&=-\sum_{\beta^q+\beta=1}\beta^{(a-1)q+q-a-1}\\&=-\sum_{\beta^q+\beta=1}(\beta^{q-a-1})(1-\beta)^{a-1}\\&=-\sum_{\beta^q+\beta=1}(\beta^{q-a-1})\sum_{j=0}^{a-1}\binom{a-1}{j}(-1)^{a-1-j}\beta^{q-2-j}=0,\end{aligned}$$

since $\sum_{\beta^q+\beta=1}\beta^s=0$ when $s<q-1$ as we will now prove:

$$\sum_{\beta^q+\beta=1}\beta^s=\sum_{\alpha^q+\alpha=0}(\gamma+\alpha)^s,$$

where γ is a fixed element with $\gamma^q + \gamma = 1$. So we get

$$\begin{aligned}\sum_{\alpha^q+\alpha=0} (\gamma+\alpha)^s &= \sum_{\alpha^q+\alpha=0} \sum_{j=0}^{s} \binom{s}{j} \gamma^{s-j} \alpha^j \\ &= \sum_{j=0}^{s} \binom{s}{j} \gamma^{s-j} \sum_{\alpha^q+\alpha=0} \alpha^j = 0.\end{aligned}$$

Index

Jørn Justesen and Tom Høholdt

A Course In Error-Correcting Codes

Second edition

This book, updated and enlarged for the second edition, is written as a text for a course aimed at 3rd or 4th year students. Only some familiarity with elementary linear algebra and probability is directly assumed, but some maturity is required. The students may specialize in discrete mathematics, computer science, or communication engineering. The book is also a suitable introduction to coding theory for researchers from related fields or for professionals who want to supplement their theoretical basis. The book gives the coding basics for working on projects in any of the above areas, but material specific to one of these fields has not been included. The chapters cover the codes and decoding methods that are currently of most interest in research, development, and application. They give a relatively brief presentation of the essential results, emphasizing the interrelations between different methods and proofs of all important results. A sequence of problems at the end of each chapter serves to review the results and give the student an appreciation of the concepts. In addition, some problems and suggestions for projects indicate direction for further work. The presentation encourages the use of programming tools for studying codes, implementing decoding methods, and simulating performance. Specific examples of programming exercises are provided on the book's home page.

ISBN 978-3-03719-179-8

9 783037 191798

www.ems-ph.org