Chapter 7

Error Control Coding

Mikael Olofsson — 2005

We have seen in Chapters 4 through 6 how digital modulation can be used to control error probabilities. This gives us a digital channel that in each time interval of duration T communicates a symbol, chosen from an alphabet of size μ, where we usually have $\mu = 2^i$, where i is a positive integer. Thus, each symbol can be used to represent i bits.

In this Chapter, we will see how we can further reduce the error probability using error control coding. This is done by encoding our data in the transmitter before the digital modulation, and by decoding the received data in the receiver after the digital demodulation. There are error control codes over alphabets of any size. We will only consider binary error control codes, i.e. the alphabet consists of two symbols, normally denoted 0 and 1.

The general idea of error control codes is to let the encoder calculate extra control bits from the information that we wish to transmit, and to transmit those control bits together with the information. If that is done in a clever way, then the decoder can detect or correct the most probable error patterns. Thus, both the encoding and the decoding of the data are done by clever mapping of sequences of bits on sequences of bits.

The available signal energy per information bit is always limited. Transmitting control bits together with the information demands extra energy. It is then natural to ask if that energy could be better used by simply amplifying the signals instead. In most reasonable situations, however, it is possible to show that using error control codes is a better way to utilize that energy.

7.1 Historical background

It all started in the late 1940's, with Shannon, Hamming and Golay. Shannon [4] introduced the basic theory on bounds for communication. He showed that it is possible to get

arbitrarily low error probability using coding on any channel, provided that the bit-rate is below a channel-specific parameter called the capacity of the channel. He did not, however, show how that can be accomplished. Shannons paper gave rise to at least two research fields, namely *information theory* which mainly deals with bounds on performance, and *coding theory* which deals with methods to achieve good communication using codes.

Coding theory started with Hamming and Golay. Hamming [3] published his construction of a class of *single*-error-correcting *binary* codes in 1950. These codes were mentioned by Shannon in his 1948 paper. Golay [2] apparently learned about Hamming's discovery through Shannon's paper. In 1949, he published a generalization of the construction to any alphabet of prime size. Both Hamming's original binary codes and Golay's generalizations are now referred to as Hamming codes. More important, Golay gave two constructions of *multiple*-error-correcting codes: one triple-error-correcting binary code and one double-error-correcting ternary code. Those codes are now known as the Golay codes.

The discoveries by Hamming and Golay initiated research activities among both engineers and mathematicians. The engineers primarily wanted to exploit the new possibilities for improved information transmission. Mathematicians, on the other hand, were more interested in investigating the algebraic and combinatorial aspects of codes. From an engineering point of view, it is not enough that a certain code can be used to obtain a certain error probability. We are also interested in efficient implementations. Of the two operations encoding and decoding, it is the decoding that is the most complex operation. Therefore, coding theory is often described as consisting of two parts, namely code construction and the development of decoding methods.

Today the theory of error control codes is well developed. A number of very efficient codes have been constructed. Error control codes are used extensively in modern telecommunication, e.g. in digital radio and television, in telephone and computer networks, and in deep space communication.

7.2 Binary Block Codes

For a *block code*, the information sequence is split into subsequences of length k. Those subsequences are called *information vectors*. Each information vector is then mapped on a vector of length n. Those vectors are called *codewords*. The code $\mathcal{C}$ is the set of those codewords. The encoding is the mentioned mapping, but the mapping itself is not part of the code. Many different mappings can be used with the same code, but whenever we use a code, we need to decide on what mapping to use. Normally, we have $n > k$. Here n is refered to as the *length* of the code. The number k is referred to as the number of information bits. We say that $\mathcal{C}$ is an (n, k) code. The number of codewords $M = 2^k$ is called the *size* of the code, and $R = k/n$ is called the *rate* of the code. The codes mentioned in Section 7.1 are all examples of block codes.

Example 7.1 *Consider the following mapping from information vectors to codewords.*

Information	Codeword
(00)	(11000)
(01)	(01110)
(10)	(10011)
(11)	(00101)

The code in this example is

$$\mathcal{C} = \{(11000), (01110), (10011), (00101)\}.$$

For this code we have $k = 2$, $n = 5$, $M = 4$ *and* $R = 2/5$.

7.2.1 ML Detection

After transmitting one of the codewords in Example 7.1 over a reasonable digital channel, we may receive any binary vector of length 5. The question now is the following. Given a received vector, how should the decoder interprete that vector? As always, that depends on the channel, on the probabilities of the codewords being sent and on the cost of the errors that may occur.

Given the received vector $\overline{x}$, we wish to estimate the codeword that has been sent according to some decision criterion. We are interested in correct detection, i.e. we want the estimate $\hat{\overline{C}}$ on the output to equal the sent codeword $\overline{C}$, where both $\hat{\overline{C}}$ and $\overline{C}$ are discrete stochastic variables.

The received vector $\overline{x}$ is a realization of the n-dimensional stochastic variable $\overline{X}$. Assuming that all erroneous decodings are equally serious, we need a decision rule that minimizes $\Pr\{\overline{C} \neq \overline{c}_k | \overline{X} = \overline{x}\}$. Let us formulate this as our first decision rule.

Decision rule 7.1 *Set* $\hat{\overline{c}} = \overline{c}_i$ *if* $\Pr\{\overline{C} \neq \overline{c}_k | \overline{X} = \overline{x}\}$ *is minimized for* $k = i$.

Using similar arguments as in Chapter 5 for detection of digital modulated signals, we arrive at the following MAP decision rule.

Decision rule 7.2 *Set* $\hat{\overline{c}} = \overline{c}_i$ *if* $\Pr\{\overline{C} = \overline{c}_k\}\Pr\{\overline{X} = \overline{x} | \overline{C} = \overline{c}_k\}$ *is maximized for* $k = i$.

We can also determine an ML decision rule by assuming that all codewords are equally probable, i.e. we assume $\Pr\{\overline{C} = \overline{c}_k\} = 1/M$, where M still is the size of the code. Then we get the following ML decision rule.

Decision rule 7.3 *Set* $\hat{\overline{c}} = \overline{c}_i$ *if* $\Pr\{\overline{X} = \overline{x} | \overline{C} = \overline{c}_k\}$ *is maximized for* $k = i$.

We still do not have a channel model ready, so we cannot at this point take the above decision rule any further.

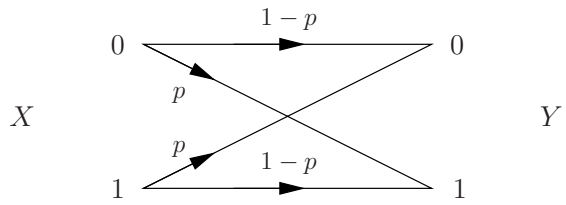


Figure 7.1: The binary symmetric channel with error probability p.

7.2.2 The Binary Symmetric Channel

Let X be the input to a binary channel, and let Y be the output. Both X and Y are stochastic variables taken from the binary alphabet $\{0, 1\}$. Consider a channel for which we have the transition probabilities

$$\begin{aligned} \Pr\{Y = 0|X = 1\} &= \Pr\{Y = 1|X = 0\} = p, \\ \Pr\{Y = 0|X = 0\} &= \Pr\{Y = 1|X = 1\} = 1 - p, \end{aligned}$$

and for which consecutive uses of the channel are statistically independent. This channel, refered to as the *binary symmetric channel* with error probability p, is displayed in Figure 7.1. Any binary digital modulation scheme used on an AWGN channel with an ML detector results in a binary symmetric channel. The binary symmetric channel is an often applied channel model, mainly because it often is a reasonable model, but also partly because of its simplicity.

Definition 1 *Let $\overline{a}$ and $\overline{b}$ be binary vectors, and let $d_{\mathrm{H}}(\overline{a}, \overline{b})$ denote the number of positions where $\overline{a}$ and $\overline{b}$ differ. Then $d_{\mathrm{H}}(\overline{a}, \overline{b})$ is called the Hamming distance between $\overline{a}$ and $\overline{b}$.*

Under the assumption that the channel is a binary symmetric channel, we obviously have

$$\Pr\{\overline{X} = \overline{x}|\overline{C} = \overline{c}_k\} = p^{d_{\mathrm{H}}(\overline{x},\overline{c}_k)}(1 - p)^{n - d_{\mathrm{H}}(\overline{x},\overline{c}_k)}.$$

Under the reasonable assumption $p < 0.5$, the ML decision rule can be reformulated in the following way for the binary symmetric channel.

Decision rule 7.4 *Set $\hat{\overline{c}} = \overline{c}_i$ if $d_{\mathrm{H}}(\overline{x}, \overline{c}_k)$ is minimized for $k = i$.*

So, just as for ML-detection of digital modulated data, the receiver should choose the possible signal that is closest to the received signal. Here the received signal is a binary vector, the possible signals are codewords from the binary code, and the distance measure is the

Hamming distance. Based on Decision rule 7.4 we get a decoding region for each codeword, consisting of all vectors that are closer to that codeword than any other codeword.

Apart from the vectors in the decoding regions, there may be vectors that are closest to more than one codeword. There are different ways to deal with those vectors in the decoder. The easiest way is to pick one of the closest codewords at random. That is also perfectly OK from an ML point of view. Another way is to let the decoder announce a decoding failure, that possibly results in a request of a retransmission of the codeword. A third way that can be used if consecutive codewords are statistically dependent, is to output the list of closest codewords (or to announce a decoding failure), and leave the decision to be made later based on the dependency of consecutive codewords.

Example 7.2 *We return to our example, where we have the code*

$$\mathcal{C} = \{(11000), (01110), (10011), (00101)\}$$

By listing all 5-dimensional vectors and checking the distances between each vector and all codewords in $\mathcal{C}$, we get the following decoding regions:

Codeword	Decoding region
(11000)	{ (11000), (01000), (10000), (11100), (11010), (11001) }
(01110)	{ (01110), (11110), (00110), (01010), (01100), (01111) }
(10011)	{ (10011), (00011), (11011), (10111), (10001), (10010) }
(00101)	{ (00101), (10101), (01101), (00001), (00111), (00100) }

The following vectors are not part of any decoding region:

$$(00000), (10110), (01011), (11101), (10100), (00010), (11111), (01001)$$

Those vectors are on distance 2 from 2 codewords each.

7.2.3 Distances Between Codewords

We have seen that when an error control code is used on a binary symmetric channel, then the decoder should choose the codeword that is closest to the received vector, and the distance measure to use is the Hamming distance. This also means that the Hamming distances between codewords are intimitely related to the resulting error probability. Let $\mathcal{S}$ be a set. Then $|\mathcal{S}|$ is to be interpreted as the size of $\mathcal{S}$, i.e. the number of elements in $\mathcal{S}$.

Definition 2 *The distance distribution of a code $\mathcal{C}$ of length n is the sequence B_i, $0 \leq i \leq n$, given by*

$$B_i = \frac{1}{M} \sum_{\overline{c}_1 \in \mathcal{C}} \left| \left\{ \overline{c}_2 \in \mathcal{C} : d_{\mathrm{H}}(\overline{c}_1, \overline{c}_2) = i \right\} \right|.$$

One way of interpreting the distance distribution is the following. Given a codeword $\overline{c}_1$, we could say that we have a distance distribution with respect to that codeword, which for every possible distance is the number of codewords on that distance from $\overline{c}_1$. The distance distribution of the code is then simply the average of those distance distributions with respect to each codeword in the code. From that interpretation it is easy to conclude that for any code of length n and size M, we have $B_0 = 1$ and $\sum_{i=0}^{n} B_i = M$.

Assume that a code $\mathcal{C}$ is used for error detection only, i.e. if we do not receive a codeword, then the receiver announces that an error has occured. This means that we get an undetected error only if we receive a codeword in $\mathcal{C}$ other than the one being sent. The probability of an undetected error, P_{ue}, is then given by

$$P_{\text{ue}} = \sum_{i=1}^{n} B_i p^i (1-p)^{n-i},$$

assuming equally probable codewords and communication over a binary symmetric channel with error probability p.

Example 7.3 *For the code* $\mathcal{C} = \{(11000), (01110), (10011), (00101)\}$ *we have the distances*

$$\begin{array}{lcllcl}
d_{\text{H}}\big((11000),(11000)\big) &=& 0, & d_{\text{H}}\big((10011),(11000)\big) &=& 3,\\
d_{\text{H}}\big((11000),(01110)\big) &=& 3, & d_{\text{H}}\big((10011),(01110)\big) &=& 4,\\
d_{\text{H}}\big((11000),(10011)\big) &=& 3, & d_{\text{H}}\big((10011),(10011)\big) &=& 0,\\
d_{\text{H}}\big((11000),(00101)\big) &=& 4, & d_{\text{H}}\big((10011),(00101)\big) &=& 3,\\
d_{\text{H}}\big((01110),(11000)\big) &=& 3, & d_{\text{H}}\big((00101),(11000)\big) &=& 4,\\
d_{\text{H}}\big((01110),(01110)\big) &=& 0, & d_{\text{H}}\big((00101),(01110)\big) &=& 3,\\
d_{\text{H}}\big((01110),(10011)\big) &=& 4, & d_{\text{H}}\big((00101),(10011)\big) &=& 3,\\
d_{\text{H}}\big((01110),(00101)\big) &=& 3, & d_{\text{H}}\big((00101),(00101)\big) &=& 0.
\end{array}$$

Thus, $\mathcal{C}$ *has the distance distribution* $B_0 = 1$, $B_1 = 0$, $B_2 = 0$, $B_3 = 2$, $B_4 = 1$ *and* $B_5 = 0$.

In the example above, the distance distribution consists of integers. Generally, distance distributions may very well consist of fractions. The most important distance between codewords in a code is the smallest distance between different codewords.

Definition 3 *The minimum distance* d *of a code* $\mathcal{C}$ *is given by*

$$d = \min_{\substack{\overline{c}_1, \overline{c}_2 \in \mathcal{C} \\ \overline{c}_1 \neq \overline{c}_2}} d_{\text{H}}(\overline{c}_1, \overline{c}_2).$$

Given a code $\mathcal{C}$ with minimum distance d, it is easily shown that all vectors on distance at most $\lfloor \frac{d-1}{2} \rfloor$ from a codeword belong to the decoding region of that codeword. The notation $\lfloor a \rfloor$ means a rounded down to the nearest integer, and $\lfloor a \rfloor$ is called the *floor* of a. We say that $\mathcal{C}$ can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. There is no vector on distance less than d from any codeword that is another codeword. Therefore, we can detect all erroneous vectors on distance at most $d-1$ from any codeword. We say that $\mathcal{C}$ can detect up to $d-1$ errors.

Example 7.4 *The distances in* $\mathcal{C} = \{(11000), (01110), (10011), (00101)\}$ *are 0, 3 and 4, as we have seen in Example 7.3. The smallest nonzero distance is 3 and thus, the minimum distance of* $\mathcal{C}$ *is 3. Used for error correction, this code can correct up to 1 error, and used for error detection, this code can detect up to 2 errors.*

It is very common to use decoders that correct all vectors on distance up to $\lfloor \frac{d-1}{2} \rfloor$ from the codewords, and that announce detected errors for all other vectors. In that case, the minimum distance is the third important parameter of the code. A code of length n, dimension k and with minimum distance d is often refered to as an (n, k, d) code.

A code does not necessarily have to be used for error correction only, or for error detection only. We can also combine error correction and error detection. A code of minimum distance d that is used for correcting up to t errors can also detect all erroneous vectors on distance up to v, $v > t$, from all codewords if $t + v < d$ holds.

7.3 Linear Codes

So far, we have not assumed that the code has any structure. Decoding a code using Decision rule 7.4 on Page 78 can be very costly. Given a received vector $\overline{x}$, Decision rule 7.4 states that we should check the distance between $\overline{x}$ and each codeword in the code. For a large code, this is completely out of the question. If we do construct codes with structure, that structure may be used to simplify the decoding.

7.3.1 The Binary Field

Before we introduce the class of linear codes, we need to develop some mathematical machinery. We are considering binary codes, i.e. each symbol both in the information vectors and in the codewords are taken from a binary alphabet, $\mathcal{A} = \{0, 1\}$. These symbols

are usually interpreted as elements of the *binary field*, $\mathbb{F}_2$. By doing that we have introduced two binary operations called *addition* and *multiplication*, defined in the following way.

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

We note that those two operations are exactly exlusive OR and boolean AND. This fact can be used – and is used – in implementations of communication systems using error control codes. We can also interprete those operations as ordinary integer addition and multiplication, with the slight modification that the results are to be reduced modulo 2. Then odd numbers are reduced to 1 and even numbers are reduced to 0. By interpreting our bits as elements in the binary field, we have a possibility to simplify the *description* of some codes. We will also be able to describe both the encoding and the decoding of those codes in algebraic terms.

The binary operations defined above obey the arithmetic rules that we are used to from real and complex numbers.

associativity :	$(x+y)+z = x+(y+z)$,	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$,
commutativity :	$x+y = y+x$,	$x \cdot y = y \cdot x$,
unity :	$x+0 = 0+x = x$,	$x \cdot 1 = 1 \cdot x = x$,
inverse :	$x+(-x) = (-x)+x = 0$,	$x \cdot x^{-1} = x^{-1} \cdot x = 1$.
distributive law :	$x(y+z) = xy+xz$	

All of these rules hold in general, *except for the inverse of multiplication*, which holds for *non-zero* elements only. We note that there is only one non-zero element, namely 1. The rules are all easily checked using the tables above. The *binary field* $\mathbb{F}_2$ is the set $\mathcal{A} = \{0, 1\}$ along with the two binary operations.

The reason that we make such a strong point of the fact that the binary operations in $\mathbb{F}_2$ obey those rules is that they are the rules of a fundamental algebraic structure called a *field*. Many mathematical results that we use more or less unconsciously depend only on those rules. For instance, most results from linear algebra hold as long as coefficients in vectors and matrices are taken from a field. Actually, words such as vectors and vector spaces are not even used unless the coefficients are taken from a field. Especially, concepts like linear (in-)dependence, bases of vector spaces, ranks of matrices and so on depend on the fact that we have a field. This observation opens up a number of possibilities for constructing codes.

Examples of fields that both mathematicians and engineers are well accustomed with are the *rational numbers*, the *real numbers*, and the *complex numbers*. These are all examples of *infinite* fields, i.e. fields with infinitely many elements. There are also infinitely many *finite* fields, i.e. fields with finitely many elements, of which the binary field is one.

7.3.2 What is a Linear Code?

Once we have defined a field, we can consider vector spaces over that field. So, consider a code $\mathcal{C}$, that is a k-dimensional subspace of the full vector space of dimension n over the binary field. Then $\mathcal{C}$ is refered to as a *linear code* of *dimension* k. The strict definition of a linear code is the following.

Definition 4 *A binary code $\mathcal{C}$ is said to be linear if $\overline{c}_1 + \overline{c}_2 \in \mathcal{C}$ holds for all $\overline{c}_1, \overline{c}_2 \in \mathcal{C}$.*

We should note that the addition used above is component-wise addition in the binary field, i.e. addition reduced modulo 2. One consequence of Definition 4 is that a code that is linear contains the all-zero vector. However, a code that does contain the all-zero vector may be nonlinear.

Example 7.5 *We return to our example code*

$$\mathcal{C} = \{(11000), (01110), (10011), (00101)\}$$

This code is not linear. That can for instance be realized by studying

$$(11000) + (01110) = (10110),$$

which is not a codeword. We can also say that $\mathcal{C}$ is non-linear since it does not contain the all-zero vector. Now, consider the code $\mathcal{C}'$, which is given by adding the vector (11000) *to all the codewords in $\mathcal{C}$. Thus, we have*

$$\mathcal{C}' = \{(00000), (10110), (01011), (11101)\}.$$

It is straightforward to check that all sums of codewords in $\mathcal{C}'$ are in $\mathcal{C}'$. Thus, $\mathcal{C}'$ is linear.

We should note that the codes $\mathcal{C}$ and $\mathcal{C}'$ in the example above are equivalent in the sense that the geometry of the codes are the same, since $\mathcal{C}'$ is a simple translation of $\mathcal{C}$. Therefore, if they are used on a binary symmetric channel, we get the same error probability for both codes.

7.3.3 Algebraic Description of Linear Codes

If $\mathcal{C}$ is a linear code, then we can use common algebraic language to describe $\mathcal{C}$ and its properties. First of all, assuming that $\mathcal{C}$ is a vector space, we can describe $\mathcal{C}$ using a basis, i.e. each codeword can be written as a linear combination of k linearly independent vectors.

We let those k vectors form the rows of a $k \times n$ *generator matrix* G. Then we can describe the code $\mathcal{C}$ as

$$\mathcal{C} = \left\{ \overline{m}G \; \forall \; \overline{m} \in \mathbb{F}_2^k \right\}.$$

G defines the mapping $\overline{c} = \overline{m}G$ from the information vector $\overline{m}$ to the codeword $\overline{c}$. Therefore, G plays an important role in the encoder of the code. Also, we note that the number of information bits, k, is the dimension of the code.

Let us recall a few concepts from linear algebra. Those are valid for matrices over any field, but the only field that we are interested in here is the binary field $\mathbb{F}_2$. Therefore, we express those concepts only for matrices over $\mathbb{F}_2$.

Definition 5 *The rank of a matrix H is the largest number of linearly independent rows in H.*

The rank of a matrix could also be defined as the largest number of linearly independent columns in the matrix. Those two rank definitions are equivalent. By definition, a matrix consisting of r rows, all linearly independent, has rank r.

Definition 6 *Let H be an $m \times n$ matrix over $\mathbb{F}_2$. The set $\left\{ \overline{c} \in \mathbb{F}_2^n : \; H\overline{c}^{\mathsf{T}} = \overline{0} \right\}$ is called the null-space of H.*

It can easily be shown that the null-space of a matrix is a vector space. It takes a bit more effort to prove the following, which is a special case of the rank-nullity theorem.

Theorem 1 *Let H be an $m \times n$ matrix over $\mathbb{F}_2$ of rank r. Then the null-space of H has dimension $n - r$.*

Thus, our code $\mathcal{C}$ can also be described as the null-space of an $(n-k) \times n$ *parity check matrix* H, consisting of linearly independent rows. We have

$$\mathcal{C} = \left\{ \overline{c} \in \mathbb{F}_2^n : H\overline{c}^{\mathsf{T}} = \overline{0} \right\}$$

We should note that this necessarily means that $HG^{\mathsf{T}} = \mathbf{0}$ holds, where $\mathbf{0}$ is an $(n-k) \times k$ all-zero matrix. We shall later find that H is important in the decoder of the code.

Example 7.6 *To find a generator matrix of our linear two-dimensional example code*

$$\mathcal{C}' = \{(00000), (10110), (01011), (11101)\},$$

we need two linearly independent codewords. A possible generator matrix is then given by

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

The codeword $\overline{c}$ corresponding to the information vector $\overline{m}$ is given by $\overline{c} = \overline{m}G$. Therefore, G defines the following mapping

Information	*Codeword*
(00)	(00000)
(01)	(01011)
(10)	(10110)
(11)	(11101)

A parity check matrix is given by

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

It is easily checked that the rows of G and of H are linearly independent. It is equally easily checked that $HG^{\mathsf{T}} = \mathbf{0}$ holds, and that $H\overline{c}^{\mathsf{T}} = \overline{0}$ holds for all $\overline{c} \in \mathcal{C}$.

There is a very simple way of determining a parity check matrix H of a code, given a generator matrix G of that code, given that G is on so called *systematic form*. By that we mean that the leftmost or rightmost k columns of G form a $k \times k$ identity matrix I_k. Let us return to our example.

Example 7.7 *Again, consider the generator matrix*

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} = (I_2,\ P) \qquad \textit{with} \qquad P = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

We note that for the parity check matrix we have

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} = \left(P^{\mathsf{T}},\ I_3\right).$$

The relation between G and H in the example is not a coincidence. It is a generally applicable method. Consider a code $\mathcal{C}$ defined by a $k \times n$ generator matrix $G = (I_k,\ P)$, i.e. G is on systematic form. The rows of G are trivially linearly independent. Now let H be the $(n-k) \times n$ matrix given by $H = (P^{\mathsf{T}},\ I_{n-k})$. The rows of H are equally trivially linearly independent. Let us check if H can serve as a parity check matrix of $\mathcal{C}$. Therefore, we study

$$HG^{\mathsf{T}} = \left(P^{\mathsf{T}},\ I_{n-k}\right) \begin{pmatrix} I_k \\ P^{\mathsf{T}} \end{pmatrix} = P^{\mathsf{T}} + P^{\mathsf{T}} = \mathbf{0}.$$

The result is an all-zero matrix. Thus, H can serve as a parity check matrix of $\mathcal{C}$.

We have noted that we can get a generator matrix of a linear (n, k) code by choosing k linearly independent codewords and use them as rows in the matrix. This means that there are typically several possible choices of a generator matrix. Generally, if G is a generator matrix of a code $\mathcal{C}$, and if T is a binary non-singular $k \times k$ matrix then TG is also a generator matrix of $\mathcal{C}$. From that we can conclude that the number of generator matrices of $\mathcal{C}$ is the same as the number of binary non-singular $k \times k$ matrices, namely $\prod_{i=0}^{k-1} \left(2^k - 2^i\right)$. In other terms, given a generator matrix G of $\mathcal{C}$, we can get all generator matrices of $\mathcal{C}$ by performing row operations on G. All those generator matrices define the same code, but each generator matrix defines its own mapping from the information vectors to the codewords.

By similar arguments, there are typically several possible choices of a parity check matrix. Given a parity check matrix H of $\mathcal{C}$, we can get all $\prod_{i=0}^{n-k-1} \left(2^{n-k} - 2^i\right)$ parity check matrices of $\mathcal{C}$ by performing row operations on H.

Example 7.8 *For our linear example code*

$$\mathcal{C}' = \{(00000), (10110), (01011), (11101)\},$$

we have found the generator matrix G and the parity check matrix H as

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} \qquad \textit{and} \qquad H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Another generator matrix G' of $\mathcal{C}'$ can be found by replacing the first row in G by the sum of the two rows in G. Similarily, an alternative parity check matrix H' of $\mathcal{C}'$ is given by row operations on H. Those are

$$G' = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix} \qquad \textit{and} \qquad H' = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

This technique can be used to find a parity check matrix of a linear code defined by a non-systematic generator matrix. First generate a new systematic generator matrix – if possible – for the code, by row operations on the original generator matrix. Then determine a parity check matrix from that new generator matrix using the method described on Page 85.

Example 7.9 *Let us start with the generator matrix*

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

Next, determine a generator matrix G'' on systematic form by using row operations on G'. Then we can for instance get

$$G'' = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix} = (P,\ I_2) \qquad \textit{with} \qquad P = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

Here we have chosen to let the rightmost columns form the needed identity matrix. From G'' we again get the parity check matrix

$$H' = \left(I_3,\ P^{\mathsf{T}}\right) = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Naturally, the same method can be used to determine a generator matrix from a parity check matrix.

7.3.4 Weights and Distances

We have noted that Hamming distances between codewords are important when analyzing the performance of error control codes. Linear codes have extra structure that gives us alternative ways of finding distances.

Definition 7 *Let $\overline{a}$ be a binary vector, and let $w_{\mathrm{H}}(\overline{a})$ denote the number of coefficients in $\overline{a}$ that are 1. The measure $w_{\mathrm{H}}(\overline{a})$ is called the Hamming weight of $\overline{a}$.*

Let $\overline{c}_1$ and $\overline{c}_2$ be codewords in a code $\mathcal{C}$. Obviously, we have $d_{\mathrm{H}}(\overline{c}_1, \overline{c}_2) = w_{\mathrm{H}}(\overline{c}_1 + \overline{c}_2)$. Consider the minimum distance d. According to the observation above, we have

$$d = \min_{\substack{\overline{c}_1, \overline{c}_2 \in \mathcal{C} \\ \overline{c}_1 \neq \overline{c}_2}} d_{\mathrm{H}}(\overline{c}_1, \overline{c}_2) = \min_{\substack{\overline{c}_1, \overline{c}_2 \in \mathcal{C} \\ \overline{c}_1 \neq \overline{c}_2}} w_{\mathrm{H}}(\overline{c}_1 + \overline{c}_2).$$

Let us assume that $\mathcal{C}$ is linear. Then $\overline{c}_1 + \overline{c}_2$ is a codeword in $\mathcal{C}$. Now, fix one of them, say $\overline{c}_1$, and let $\overline{c}_2$ run through all other codewords in $\mathcal{C}$. Then $\overline{c}_1 + \overline{c}_2$ runs through all non-zero codewords in $\mathcal{C}$. Thus, we have

$$d = \min_{\overline{c} \in \mathcal{C} \setminus \{\overline{0}\}} w_{\mathrm{H}}(\overline{c}).$$

So, the minimum distance of a linear code $\mathcal{C}$ is the minimum non-zero Hamming weight of the codewords in $\mathcal{C}$.

Example 7.10 *Back to our linear example code*

$$\mathcal{C}' = \{(00000), (10110), (01011), (11101)\}.$$

In this code we have the weights 0, 3 and 4. The minimum non-zero weight of $\mathcal{C}'$ is therefore 3, and since $\mathcal{C}'$ is linear, that number is also the minimum distance of $\mathcal{C}'$.

Since the distance between two codewords in a code $\mathcal{C}$ is the weight of some codeword in $\mathcal{C}$, we define the following in accordance to the definition of the distance distribution of $\mathcal{C}$.

Definition 8 *The weight distribution of a code $\mathcal{C}$ of length n is the sequence A_i, $0 \leq i \leq n$, given by*

$$A_i = \left| \left\{ \overline{c} \in \mathcal{C} : w_{\mathrm{H}}(\overline{c}) = i \right\} \right|.$$

In other words, A_i is the number of codewords in $\mathcal{C}$ with weight i. For a linear code, it is easily verified that the distance distribution and the weight distribution are equal. The reason that we are interested in this result is that there is less calculation involved in determining the weight distribution of a code, than in determining the distance distribution of the same code using the definitions.

Example 7.11 *For the nonlinear code $\mathcal{C} = \{(11000), (01110), (10011), (00101)\}$ we found the distance distribution $(B_0, B_1, B_2, B_3, B_4, B_5) = (1, 0, 0, 2, 1, 0)$ in Example 7.3. The weights of the four codewords are in order 2, 3, 3 and 2. Thus, the weight distribution of $\mathcal{C}$ is $(A_0, A_1, A_2, A_3, A_4, A_5) = (0, 0, 2, 2, 0, 0)$.*

From Example 7.11 we deduce that for nonlinear codes, the weight distribution may not be equal to the distance distribution. Actually, for non-linear codes, those two distributions are typically not equal.

7.3.5 Minimum Distances and Parity Check Matrices

Let $\overline{c}$ be a codeword of a linear code $\mathcal{C}$ with parity check matrix H. From the equality $H\overline{c}^{\mathsf{T}} = \overline{0}$, we conclude that $\overline{c}$ points out a set of linearly dependent columns in H, namely the columns in H corresponding to the coefficients in $\overline{c}$ that are 1.

Example 7.12 *Once again, we return to our linear example code with parity check matrix*

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

One of the codewords is

$$\overline{c} \;=\; (\;1 \quad 0 \quad 1 \quad 1 \quad 0\;),$$

which points out the first, third and fourth columns in H. Let us call those columns $\overline{h}_1$, $\overline{h}_3$ and $\overline{h}_4$. Then we have

$$H\overline{c}^{\mathsf{T}} = \overline{h}_1 + \overline{h}_3 + \overline{h}_4 = \overline{0}$$

and those columns are linearly dependent.

Not only does each codeword in $\mathcal{C}$ point out a set of linearly dependent columns in H. Since $\mathcal{C}$ is the null-space of H, there are no other sets of linearly dependent columns in H than those pointed out by the codewords. Thus, a third way of determining the minimum distance is to find the smallest number of linearly dependent columns in H.

Example 7.13 *Again, let us find the minimum distance of our linear example code $\mathcal{C}'$, but this time based on its parity check matrix*

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix} = \left(P^{\mathsf{T}} \; I_3\right).$$

We want to determine all sets of linearly dependent columns in H. We do that by studying all sums of columns in P^{T}, and by adding the columns in I_3 that are needed in order to get the sum $\overline{0}$. In this way we have determined all sets of linearly dependent columns in H. Again, let $\overline{h}_1$ through $\overline{h}_5$ denote the columns of H, from left to right.

Sum from P^{T}	Added columns from I_3	Total number of columns
$\overline{h}_1$	$\overline{h}_3 + \overline{h}_4$	3
$\overline{h}_2$	$\overline{h}_4 + \overline{h}_5$	3
$\overline{h}_1 + \overline{h}_2$	$\overline{h}_3 + \overline{h}_5$	4

The smallest number we have found is 3. Thus, $\mathcal{C}$ has minimum distance 3.

The example above is a very small one, but what if the code is large? Then P^T has many columns. It may then look like the method used in the example is a bit tedious, especially since the calculations that seem to be necessary are exactly the same as the calculations that are necessary to generate all codewords from the generator matrix. However, it is possible to determine a stop criterion if the sums of columns from P^T are chosen in a certain order. First consider all sums of one column, i.e. go through all columns in P^T. Then consider all sums of two columns. Then three columns, and so on. While doing that, keep track of the smallest number of linearly dependent columns found so far. These calculations can be stopped when the number of columns from P^T reaches the smallest number of linearly dependent columns found so far. This strategy can at least sometimes substantially reduce the calculations compared to calculating all codewords and checking the weights of them.

7.3.6 Decoding Linear Codes

Let $\mathcal{C}$ be a binary linear code with parity check matrix H, let $\overline{c}$ be the sent codeword, and let $\overline{r}$ be the received vector. Then we have $H\overline{c}^\mathsf{T} = \overline{0}$. If we have $H\overline{r}^\mathsf{T} = \overline{0}$, then $\overline{r}$ is also a codeword. It should therefore be reasonable for a decoder to check this. A decoder based on this idea calculates the *syndrome*

$$\overline{s} = H\overline{r}^\mathsf{T},$$

which is an $(n-k)$-dimensional vector. Define the *error vector* $\overline{e} = \overline{c} + \overline{r}$. Then we get

$$\overline{s} = H\left(\overline{c}^\mathsf{T} + \overline{e}^\mathsf{T}\right) = H\overline{c}^\mathsf{T} + H\overline{e}^\mathsf{T}.$$

Since we have $H\overline{c}^\mathsf{T} = \overline{0}$, this finally becomes

$$\overline{s} = H\overline{e}^\mathsf{T},$$

and $\overline{s}$ depends only on $\overline{e}$. The problem at this point is that there are several possible error vectors for each syndrome. More precisely, if $\overline{e}$ is an error vector that results in a certain syndrome, and if $\overline{c}$ is a codeword in $\mathcal{C}$, then $\overline{c} + \overline{e}$ is another error vector that results in the same syndrome. Therefore, there are 2^k different error vectors for each syndrome. Which one is the most likely error vector?

Recall that if a code is used on a binary symmetric channel, then an ML detector chooses the codeword that is closest to the received vector. Thus, we should find the error vector with smallest weight among the error vectors corresponding to the syndrome of the received vector. Let e_i denote the i'th coefficient in $\overline{e}$, i.e. we have

$$\overline{e} = (e_1, e_2, \ldots, e_n).$$

Similarily, let $\overline{h}_i$ denote the i'th column in H, i.e. we have

$$H = \begin{pmatrix} | & | & & | \\ \overline{h}_1 & \overline{h}_2 & \cdots & \overline{h}_n \\ | & | & & | \end{pmatrix}.$$

Then we get

$$\overline{s} = H\overline{e}^{\mathsf{T}} = \sum_{i=1}^{n} e_i \overline{h}_i.$$

Consider an error vector of weight 1, with its only 1 in position i. Then we have

$$e_j = \begin{cases} 1, & j = i, \\ 0, & j \neq i, \end{cases}$$

and $\overline{s} = \overline{h}_i$, which clearly points at position i. The next step would be to change the i'th bit in $\overline{r}$ in order to get the codeword estimate $\hat{\overline{c}}$.

Example 7.14 *Consider again our linear example code with parity check matrix*

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

Assume that we have received the vector $\overline{r} = (10111)$. *The corresponding syndrome is then given by*

$$\overline{s} = H\overline{r}^{\mathsf{T}} = \overline{h}_1 + \overline{h}_3 + \overline{h}_4 + \overline{h}_5 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \overline{h}_5.$$

A decoder that corrects single errors will therefore change position 5 in $\overline{r}$ *to produce the codeword estimate* $\hat{\overline{c}} = (10110)$.

Generally, finding the error vector with smallest weight corresponds to finding the smallest set of columns in H that sum up to $\overline{s}$.

7.3.7 Repetition Codes

Possibly, the simplest example of an error control code is a *repetition code*. This is a one-dimensional linear code. It simply repeats the single information bit n times. Thus, the only possible generator matrix of this code is

$$G = \begin{pmatrix} 1 & 1 & \ldots & 1 \end{pmatrix}.$$

We interprete the last coefficient in G as the 1×1 identity matrix I_1, and the rest of the matrix is then our P. Then we find a parity check matrix as

$$H = \begin{pmatrix} & & 1 \\ & I_{n-1} & \vdots \\ & & 1 \end{pmatrix}.$$

There are only two codewords in this code, namely the all-zero vector and the all-one vector. We therefore have $d = n$. Repetition codes can be decoded using majority vote, i.e. count the number of zeros and ones in the received vector, and output the majority choice.

7.3.8 Parity Check codes

One type of error control code that has been popular in memories are simple parity check codes. These are binary linear codes with $n = k + 1$ Apart from the k information bits, one extra bit is sent which is the sum of all information bits reduced modulo 2. This immediately gives us the generator matrix

$$G = \begin{pmatrix} & & 1 \\ & I_{n-1} & \vdots \\ & & 1 \end{pmatrix},$$

and we can find the only possible parity check matrix as

$$H = \begin{pmatrix} 1 & 1 & \dots & 1 \end{pmatrix}.$$

The smallest number of linearly dependent columns in H is 2. Thus, the code has minimum distance 2. This code contains all binary vectors of even weight, and it can only detect errors. More precisely, it detects all odd-weight errors.

Parity check codes as described here are often called *even parity check* codes, since all codewords have even weight. There are also variants of the above called *odd parity check* codes, where all codewords have odd weight. Those non-linear codes are most easily described as even parity check codes where the last bit is inverted in the Boolean sense.

7.3.9 Hamming Codes

Hamming [3] codes are linear binary codes, defined by one parameter, m, which is an integer satisfying $m \geq 2$. The columns of a parity check matrix H of a Hamming code are

all non-zero binary vectors of length m, and each binary vector appears once in H. There are $2^m - 1$ non-zero binary vectors of length m. So, H is an $m \times (2^m - 1)$ matrix. But H is supposed to be an $(n-k) \times n$ matrix, where n is the length and k is the dimension of the code, as usual. Thus, we can identify the parameters

$$\begin{aligned} n &= 2^m - 1, \\ k &= 2^m - m - 1, \end{aligned}$$

of the code. As usual, let d be the minimum distance of the code. The all-zero vector is not a column in H. Thus, there is no linearly dependent set of one column in H and we have $d > 1$. No two columns in H are equal. Therefore, there is no linearly dependent set of two columns in H and we have $d > 2$. Since the columns in H are all binary vectors of length m, there is at least one linearly dependent set of three columns. For instance, the vectors $(100\ldots0)^\mathsf{T}$, $(010\ldots0)^\mathsf{T}$, and $(110\ldots0)^\mathsf{T}$ form a linearly dependent set of columns in H. We can conclude that the minimum distance is given by $d = 3$.

Example 7.15 *Consider the Hamming code corresponding to $m = 2$. This is a $(3, 1)$ code with parity check matrix*

$$H = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

and generator matrix

$$G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}.$$

This code is obviously a repetition code of length 3.

Example 7.16 *Consider the Hamming code corresponding to $m = 3$. This is a $(7, 4)$ code with parity check matrix*

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

From H, we can immediately determine the generator matrix

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

G defines the following mapping between information vectors and codewords.

Info	*Codeword*	*Weight*	*Info*	*Codeword*	*Weight*
(0000)	(0000000)	0	(1000)	(1101000)	3
(0001)	(1110001)	4	(1001)	(0011001)	3
(0010)	(0110010)	3	(1010)	(1011010)	4
(0011)	(1000011)	3	(1011)	(0101011)	4
(0100)	(1010100)	3	(1100)	(0111100)	4
(0101)	(0100101)	3	(1101)	(1001101)	4
(0110)	(1100110)	4	(1110)	(0001110)	3
(0111)	(0010111)	4	(1111)	(1111111)	7

The weight distribution of this code is thus $A_0 = A_7 = 1$, $A_3 = A_4 = 7$, and $A_i = 0$ for all other values of i. As we have already shown, the minimum distance is 3.

The minimum distance of a Hamming code is 3, and therefore we can correct all error patterns of weight up to 1. It is natural to ask if there are possible received vectors that are not within distance 1 from the codewords. Consider a sphere of radius 1 around a codeword. By that we mean the set containing the codeword and all vectors on distance 1 from the codeword. We have $n = 2^m - 1$. Thus, there are $2^m = 2^{n-k}$ vectors in each such sphere. Now, consider the union of all those disjoint spheres. There are 2^k codewords and 2^k spheres. The union therefore contains $2^k 2^{n-k} = 2^n$ vectors, which is exacly the number of n-dimensional vectors. Thus, there are no vectors except those within distance 1 from the codewords, which is a very uncommon property.

7.3.10 Duality

Consider a linear code $\mathcal{C}$ with generator matrix G and parity check matrix H. Those matrices share the property that they have full rank. This means that we may very well define a code for which we have interchanged the meaning of those matrices.

Definition 9 *Consider a linear code $\mathcal{C}$ with generator matrix G. The code $\mathcal{C}^\perp$ for which G is a parity check matrix is called the dual of $\mathcal{C}$.*

Obviously, if $\mathcal{C}^\perp$ is the dual of $\mathcal{C}$, then $\mathcal{C}$ is the dual of $\mathcal{C}^\perp$. If $\mathcal{C}$ is an (n, k) code, then $\mathcal{C}^\perp$ is an $(n, n-k)$ code. We noted on pages 91 and 92 that a generator matrix of the repetition code of length n is a parity check matrix of the simple parity check code of length n. Thus, those codes are each others duals.

Example 7.17 *Consider a* $(7, 4)$ *Hamming code* $\mathcal{C}$ *with parity check matrix*

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

Then H *is a generator matrix of the dual code* $\mathcal{C}^\perp$*, i.e. the codewords of* $\mathcal{C}^\perp$ *are given as* $\overline{m}H$*, where* $\overline{m}$ *is a 3-dimensional information vector. We get the (7,3) code*

Info	*Codeword*	*Weight*
(000)	(0000000)	0
(001)	(0010111)	4
(010)	(0101011)	4
(011)	(0111100)	4

Info	*Codeword*	*Weight*
(100)	(1001101)	4
(101)	(1011010)	4
(110)	(1100110)	4
(111)	(1110001)	4

The only non-zero weight is 4. Thus, $\mathcal{C}^\perp$ *has minimum distance 4.*

The code $\mathcal{C}^\perp$ in the example is a so called *simplex code*. Such codes share the characterization that all distances between different codewords are equal. All duals of binary Hamming codes are simplex codes. If $\mathcal{C}$ is a $(2^m - 1, 2^m - m - 1, 3)$ Hamming code, then $\mathcal{C}^\perp$ is a $(2^m - 1, m, 2^{m-1})$ simplex code.

Let $\overline{c}_1$ be a codeword in the linear code $\mathcal{C}$ with generator matrix G and parity check matrix H, and let $\overline{c}_2$ be a codeword in the dual code $\mathcal{C}^\perp$. Then $\overline{c}_2 \cdot \overline{c}_1^\mathsf{T} = 0$ holds, which follows from $HG^\mathsf{T} = \mathbf{0}$. In other words, $\overline{c}_1$ and $\overline{c}_2$ are orthogonal. To be precise, $\mathcal{C}^\perp$ consists of all vectors that are orthogonal to all codewords in $\mathcal{C}$. The concept of orthogonality can be somewhat confusing for vectors over the binary field. For instance, a vector can be orthogonal to itself. More precisely, every vector of even weight is orthogonal to itself. That opens the possibility that there might exist linear codes that are their own duals.

Example 7.18 *The* $(4, 2)$ *code with generator matrix*

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

is its own dual, since G *also is a parity check matrix of the code.*

A code that is its own dual is called *self-dual*. It can easily be realized that $n = 2k$ holds for all self-dual codes, and that all codewords in a self-dual code have even weight.

7.3.11 Cyclic Codes

By introducing linear codes, we imposed some structure on our codes that simplified the description of the codes, but also simplified the decoding of the codes. Let us impose even more structure, that among other things simplifies the encoding.

Definition 10 *A linear code, for which all cyclic shifts of all codewords are codewords, is called cyclic.*

Many linear block codes can be described as cyclic codes.

Example 7.19 *Consider a* $(7,4)$ *code* $\mathcal{C}$ *with parity check matrix*

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

$\mathcal{C}$ *is obviously a Hamming code, since the columns are all non-zero binary vectors of length 3. Compared to the description given earlier, we have simply reordered the columns in the parity check matrix. This can be interpreted as reordering the coefficients in the codewords in the same way. Let* c_i *denote the* i*'th coefficient in the codeword* $\overline{c}$*, i.e. we have*

$$\overline{c} = (c_1, c_2, \ldots, c_7).$$

Then the parity check relation $H\overline{c}^{\mathsf{T}} = \overline{0}$ *can be rewritten as*

$$\begin{aligned} c_1 + c_2 + c_3 + c_5 &= 0, \\ c_2 + c_3 + c_4 + c_6 &= 0, \\ c_3 + c_4 + c_5 + c_7 &= 0, \end{aligned}$$

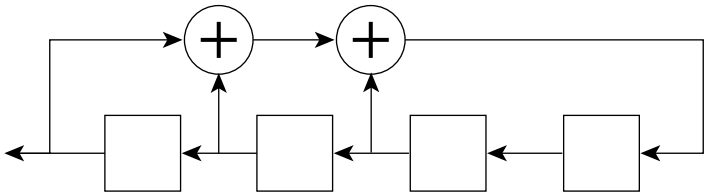
which in turn can be rewritten as

$$\begin{aligned} c_5 &= c_1 + c_2 + c_3, \\ c_6 &= c_2 + c_3 + c_4, \\ c_7 &= c_3 + c_4 + c_5. \end{aligned}$$

Thus, we can write this as

$$c_i = c_{i-4} + c_{i-3} + c_{i-2}, \qquad \text{for } i \in \{5, 6, 7\}.$$

Actually, the relation holds for all i *if the indices are reduced modulo 7. Anyhow, this gives us the encoder in Figure 7.2. A codeword is generated by first loading the register with the*



c_1	c_2	c_3	c_4
c_2	c_3	c_4	c_5
c_3	c_4	c_5	c_6
c_4	c_5	c_6	c_7

Figure 7.2: Encoder circuit for a cyclic Hamming code.

information symbols c_1, c_2, c_3 and c_4 and then performing successive shifts. The codeword $\overline{c} = (c_1, c_2, \ldots, c_7)$ will then be generated as the output, with the last three symbols satisfying the correct parity relations. The generator matrix corresponding to this encoder is

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

It is left as an exercise to show that.

7.4 Product Codes

There are a number of ways to construct large codes with small codes as building blocks. One alternative is given by *product codes*.

The encoding of a product code $\mathcal{C}$ is done by first placing information bits in a $k_1 \times k_2$ matrix. Each column of this $k_1 \times k_2$ matrix is encoded with an (n_1, k_1, d_1) code $\mathcal{C}_1$, resulting in an $n_1 \times k_2$ matrix. Then each row of this $n_1 \times k_2$ matrix is encoded with an (n_2, k_2, d_2) code $\mathcal{C}_2$, resulting in an $n_1 \times n_2$ matrix, which is the codeword corresponding to the original $k_1 k_2$ information bits. Obviously, we have created an $(n_1 n_2, k_1 k_2)$ code. If both $\mathcal{C}_1$ and $\mathcal{C}_2$ are linear, then so is $\mathcal{C}$. The principle is displayed in Figure 7.3a.

We would like to determine the minimum distance of the product code $\mathcal{C}$ above. For simplicity, let us assume that $\mathcal{C}_1$, $\mathcal{C}_2$ and thus also $\mathcal{C}$ are linear. Then we can determine

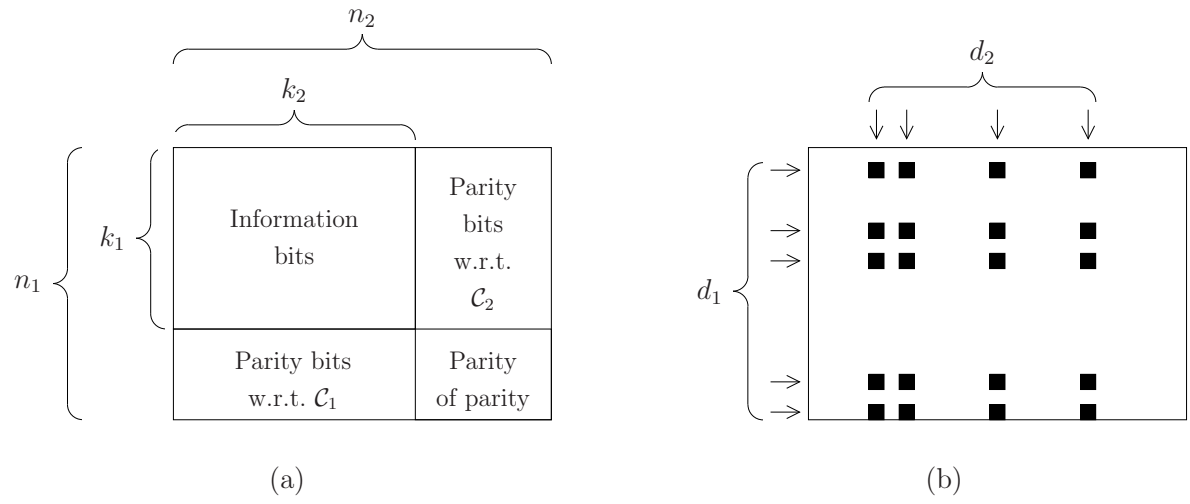


Figure 7.3: A product code. (a) Encoding. (b) A minimum non-zero weight codeword. The black squares represent positions with ones.

the smallest non-zero weight of the codewords in $\mathcal{C}$. Consider a codeword that is non-zero in one position, say (i, j). Then the weight of column j has to be at least d_1. Thus, we have at least d_1 non-zero rows, including row i. Each of those rows have to have weight at least d_2, from which we can conclude that the weight of the codeword is at least $d_1 d_2$. But, consider a position i, such that there is a codeword $\overline{c}_1$ in $\mathcal{C}_1$ with weight d_1 and with 1 in position i. Also, consider a position j, such that there is a codeword $\overline{c}_2$ in $\mathcal{C}_2$ with weight d_2 and with 1 in position j. Then one of the codewords in $\mathcal{C}$ is given by choosing $\overline{c}_1$ as column j, and choosing $\overline{c}_2$ as rows l for each l such that the l'th coefficient in $\overline{c}_1$ is 1. This gives us d_1 rows with weight d_2. Thus, there is at least one codeword with weight $d_1 d_2$, and the minimum distance of $\mathcal{C}$ is $d_1 d_2$. Such a minimum weight codeword is displayed in Figure 7.3b.

A simple and obvious way of decoding product codes is by iteratively decoding columns and rows. By that we mean that first we decode the columns using a decoder for $\mathcal{C}_1$, then we decode the rows using a decoder for $\mathcal{C}_2$, then the columns again, and the rows, and so on until we have found a codeword, or cannot decode further. However, this only guarantees that we can decode errors of weight up to $\lfloor \frac{d_1-1}{2} \rfloor \cdot \lfloor \frac{d_2-1}{2} \rfloor \approx \frac{d_1 d_2}{4}$. A decoder that is able to decode up to $\lfloor \frac{d_1 d_2-1}{2} \rfloor$ has to take the whole code into account.

Many error control codes used in electronic memories are product codes or codes obtained in similar ways, where the component codes are very simple. The resulting code is not especially good in terms of minimum distance. The reason that they are used in memories is instead that the decoding can be made very simple, and can therefore be done very fast in hardware.

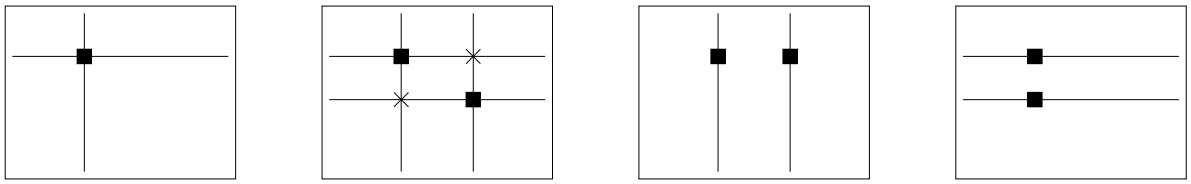


Figure 7.4: Decoding a product code based on simple parity check codes.
(a) A single error.
(b) Two errors in in different rows and columns.
(c) Two errors in the same row.
(d) Two errors in the same column.
The black squares represent errors, and the horizontal and vertical lines indicate errors, detected by the subcodes. The crosses in (b) represent alternative error positions that would give rise to the same detection of errors.

Example 7.20 *Consider a product code $\mathcal{C}$, for which the component codes $\mathcal{C}_1$ and $\mathcal{C}_2$ are simple parity check codes. Then we have the minimum distances $d_1 = d_2 = 2$, and $\mathcal{C}$ has minimum distance $d = d_1 d_2 = 4$. Thus, $\mathcal{C}$ can correct one error and detect two errors, while the two component codes can only detect one error.*

First, consider a single error in position (i, j). Then $\mathcal{C}_1$ will detect an error in column j and $\mathcal{C}_2$ will detect an error in row i. We have thus identified the error position, as can be seen in Figure 7.4a, and we can correct this error pattern.

Second, consider two errors in different rows and columns. Let (i_1, j_1) and (i_2, j_2) be the two error positions. Then $\mathcal{C}_1$ will detect errors in columns j_1 and j_2, while $\mathcal{C}_2$ will detect errors in columns i_1 and i_2. However, there is another error pattern that gives rise to the same error detection and that is the error pattern with errors in positions (i_1, j_2) and (i_2, j_1), as indicated in Figure 7.4b. So, we cannot correct this error pattern. All the decoder can do is to report a detected error, and possibly report that the two mentioned error patterns are the most likely ones.

Finally, we can have two errors in the same row or column. Let us consider two errors in the same row, in positions (i, j_1) and (i, j_2). Then $\mathcal{C}_1$ will detect errors in columns j_1 and j_2, but $\mathcal{C}_2$ will not detect any errors at all, see Figure 7.4c. Thus, we cannot correct this error pattern. All the decoder can do in this situation is to report a detected error, and possibly report that the most likely error patterns have an error in column j_1 and an error in column j_2, and that those two errors are on the same row. Based on the same arguments, we cannot correct two errors in the same column either. This situation is displayed in Figure 7.4d.

7.5 Bounds for Error Control Codes

There are a number of natural question to pose. Some of those are the following. Given n and k, what is the largest d such that there is an (n, k, d) code? Given n and d, what is the largest k such that there is an (n, k, d) code? Given k and d, what is the smallest n such that there is an (n, k, d) code?

These questions can be answered by computer search for fairly small values of the parameters, but generally those are very hard questions to answer. To help answering those questions, people have derived a large number of bounds. Most bounds cannot tell us if a code with given parameters definitely exists, but they can tell us if a code definitely does not exist. We will only mention a few of those bounds. There are also bounds that can tell us if a code with given parameters definitely exists, but cannot tell us if a code definitely does not exist. Many of those bounds are based on explicit code constructions.

7.5.1 The Hamming Bound

For a binary (n, k, d) code, place spheres of radius $\lfloor \frac{d-1}{2} \rfloor$ around each of the 2^k codewords. Those spheres are disjoint, and the union of them cannot contain more than 2^n vectors, since that is the total number of n-dimensional vectors. The number of n-dimensional vectors in a sphere of radius $\lfloor \frac{d-1}{2} \rfloor$ is

$$\sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i}.$$

Thus, we have

$$2^n \geq 2^k \sum_{i=0}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i}.$$

This is the *Hamming bound*, which is also known as the *sphere packing bound*. A code $\mathcal{C}$ which meets the Hamming bound is called *perfect*. When we say that a code meets a bound, we mean that equality holds where the bound has an inequality.

As we actually already have seen, binary Hamming codes meet the Hamming bound, and are thus perfect. There are very few other examples of perfect binary codes. The binary Golay code is perfect, repetition codes of odd lengths are perfect, and the trivial linear codes with $k = n$ are perfect.

7.5.2 The Singleton Bound

For a binary (n, k, d) code $\mathcal{C}$, create a new $(n - d + 1, k)$ code $\mathcal{C}'$ by deleting the first $d - 1$ coefficients in each codeword in $\mathcal{C}$. Since the minimum distance of $\mathcal{C}$ is d, those codewords are still distinct, and thus, there are 2^k codewords in $\mathcal{C}'$. But, the length of $\mathcal{C}'$ is $n - d + 1$. Therefore, there cannot be more than 2^{n-d+1} codewords in $\mathcal{C}'$. Thus, we have

$$2^{n-d+1} \geq 2^k,$$

which can be rewritten as

$$n - k \geq d - 1.$$

This bound is known as the *Singleton bound*, and codes meeting the Singleton bound are called *maximum distance separable* (MDS) codes. Just as there are very few perfect binary codes, there are very few binary MDS codes. Repetition codes are MDS codes, which is easily shown.

7.6 Cyclic Redundancy Check Codes

Cyclic redundancy check codes (CRC codes) are linear codes that are used for *automatic repeat request* (ARQ). CRC codes are used for error detection only, even though they may very well have minimum distances that make it possible to use them for error correction. When an error is detected, a request for retransmission of the codeword is transmitted back to the sender.

7.6.1 Polynomial Division

The following theorem should be well known. It has been around for over 2000 years.

Theorem 2 (Division algorithm for integers)
Given integers a and b, with $b \neq 0$, there are uniquely determined integers q and r, with $0 \leq r < |b|$, such that $a = qb + r$ holds.

Determining the quotient q and the remainder r is normally referred to as *integer division*. Recall the method to perform division from primary school, with $a = 2322$ and $b = 12$.

Then the division is performed as

$$\begin{array}{r|l} & 193 \\ \hline 12 & 2322 \\ & -12 \\ \hline & 112 \\ & -108 \\ \hline & 42 \\ & -36 \\ \hline & 6 \end{array}$$

and we have found $q = 193$ and $r = 6$.

Instead of integers, consider polynomials over $\mathbb{F}_2$, i.e. the coefficients of the polynomials are 0 and 1, and additions and multiplications of such polynomials are done as for ordinary polynomials over the reals, but each coefficient should be reduced modulo 2. There is a version of Theorem 2 for such polynomials as well.

Theorem 3 (Division algorithm for binary polynomials)
Given binary polynomials $a(x)$ and $b(x)$, with $b(x) \neq 0$, there are uniquely determined binary polynomials $q(x)$ and $r(x)$, with $\deg\{r(x)\} < \deg\{b(x)\}$, such that the relation $a(x) = q(x)b(x) + r(x)$ holds.

This theorem will be used to explain the properties of CRC codes. Determining the quotient $q(x)$ and the remainder $r(x)$ is normally referred to as *polynomial division*.

Polynomial division can be done in a similar way as the integer division above. Consider the binary polynomials $a(x) = x^4 + x^3 + 1$ and $b(x) = x^2 + 1$. Then the division is performed as

$$\begin{array}{r|l} & 1 \cdot x^2 + 1 \cdot x + 1 \\ \hline 1 \cdot x^2 + 0 \cdot x + 1 & 1 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 1 \\ & 1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 \\ \hline & 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x \\ & 1 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x \\ \hline & 1 \cdot x^2 + 1 \cdot x + 1 \\ & 1 \cdot x^2 + 0 \cdot x + 1 \\ \hline & 1 \cdot x + 0 \end{array}$$

and we have found $q(x) = x^2 + x + 1$ and $r(x) = x$. Since it is a bit tedious to write all those powers of x, we may allow ourselves to write only the coefficients, like so:

$$
\begin{array}{r|l}
 & 111 \\
\hline
101 & 11001 \\
 & 101 \\
\hline
 & 110 \\
 & 101 \\
\hline
 & 111 \\
 & 101 \\
\hline
 & 10
\end{array}
$$

Note that it is essential that the numbers above are not interpreted as integers. The addition (subtraction) has to be done component-wise reduced modulo 2.

7.6.2 Generation of a CRC Codeword

Consider a message consisting of k bits. Let those bits be the coefficients of the binary polynomial $m(x)$ of degree at most $k-1$. For a CRC code, there is a binary polynomial $p(x)$ of degree $n-k$. This fixed polynomial is used to generate parity bits from $m(x)$, by dividing $x^{n-k}m(x)$ by $p(x)$. Theorem 3 then states that there are uniquely determined polynomials $q(x)$ and $r(x)$ with $\deg r(x) < \deg p(x)$, such that

$$x^{n-k}m(x) = q(x)p(x) + r(x)$$

holds. The codeword, $c(x)$, is then given by

$$c(x) = x^{n-k}m(x) + r(x).$$

So, what we need to calculate by polynomial division is the remainder $r(x)$ only. That can easily be done using a linear feedback shift register as in Figure 7.5. The shift register is initiated with all zeros, and after that the coefficients of $m(x)$ are shifted in with most significant bit first. After n clock cycles, $r(x)$ is in the register. and the coefficients of $c(x)$ are transmitted as a sequence of bits. The interpretation of Figure 7.5 may be clarified by Figure 7.6, where there is an example of a shift register for the case $p(x) = x^4 + x^3 + x + 1$.

The most common CRC polynomials are listed in Table 7.1.

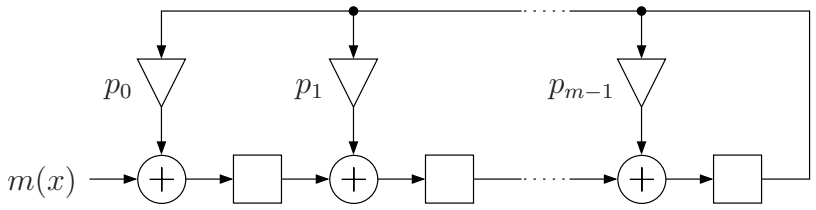


Figure 7.5: A linear feedback shift register for CRC code generation with $p(x) = \sum_{i=0}^{m} p_i x^i$, the general case. The input $m(x)$ is the coefficients of $m(x)$ fed with most significant bit first.

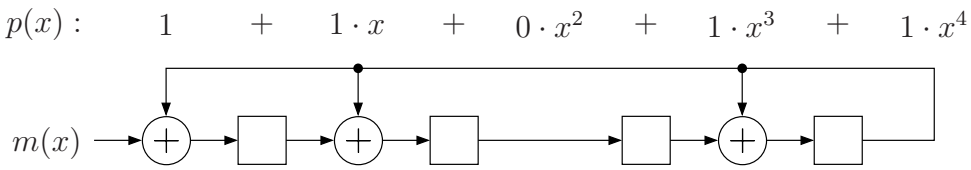


Figure 7.6: A linear feedback shift register for CRC code generation with $p(x) = x^4 + x^3 + x + 1$. The input $m(x)$ is the coefficients of $m(x)$ fed with most significant bit first.

$n-k$	$p(x)$
8	$x^8 + x^2 + x + 1$
8	$x^8 + x^7 + x^4 + x^3 + x + 1$
10	$x^{10} + x^9 + x^5 + x^4 + x + 1$
12	$x^{12} + x^{11} + x^3 + x^2 + x + 1$
16	$x^{16} + x^{15} + x^2 + 1$
16	$x^{16} + x^{12} + x^5 + 1$
24	$x^{24} + x^{23} + x^6 + x^5 + x + 1$
32	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Table 7.1: The most common CRC polynomials.

7.6.3 Detection of Errors

Consider the sent codeword,

$$c(x) = x^{n-k}m(x) + r(x).$$

According to the reasoning above, we also have

$$x^{n-k}m(x) = q(x)p(x) + r(x).$$

Identifying in the first equation, we get

$$c(x) = q(x)p(x) + r(x) + r(x) = q(x)p(x).$$

Thus, dividing $c(x)$ by $p(x)$ results in the remainder 0. A CRC decoder checks exactly that. It takes the received sequence of bits and interpretes it as a polynomial, $y(x)$. The error is now an error polynomial, $w(x)$, added to $c(x)$, i.e.

$$y(x) = c(x) + w(x).$$

The decoder calculates the remainder of $y(x)/p(x)$. If there are no errors, we get

$$y(x) = c(x) + w(x) = q(x)p(x) + w(x).$$

Thus, the remainder of $y(x)/p(x)$ is also the remainder of $w(x)/p(x)$. It can be shown that this remainder is zero if and only if $w(x)$ is a codeword. So, if this remainder is non-zero, we know for sure that $w(x)$ is non-zero.

7.6.4 Undetected Errors

We noted that if the error $w(x)$ happens to be a codeword, then the remainder will be zero. Thus, if $w(x)$ is a non-zero codeword, that error will pass undetected. The probability of undetected errors depends on the weight distribution of the code. However, a CRC code is primarily characterized by the polynomial $p(x)$. That polynomial can be used with different codeword length n, and the weight distribution of the code varies with n. Therefore, it can be hard to determine the probability of an undetected error. One simple thing that can be said is that if the weight of $p(x)$ is even, then the polynomial is divisible by $x + 1$, and the CRC code will at least detect all errors of odd weight. Also, all error bursts of length less than the degree of $p(x)$ are detected.