ERROR CORRECTING CODES MT361/MT461/MT5461

MARK WILDON

These notes are intended to give the logical structure of the course; proofs and further remarks will be given in lectures. Further installments will be issued as they are ready. All handouts and problem sheets will be put on Moodle.

I would very much appreciate being told of any corrections or possible improvements to these notes.

You are warmly encouraged to ask questions in lectures, and to talk to me after lectures and in my office hours. I am also happy to answer questions about the lectures or problem sheets by email. My email address is `mark.wildon@rhul.ac.uk`.

Lecture times: Monday 3pm (MFLEC), Tuesday 3pm (BLT2) and Thursday 10am (BLT2).

Extra lecture for MT461/MT5461: Thursday noon (ABLT3).

Office hours in McCrea 240: Tuesday 11am, Thursday 2pm and Friday 11am.

Date: Second term 2012/13.

PRELIMINARIES

Error correcting codes were invented in the late 1940s and have since become an essential part of the modern electronic world. Compact discs, satellites, space probes and mobile phone networks all depend on ideas we will develop in this course. Many of these ideas are due to Richard W. Hamming (1915–1998).

> *"Mathematics is not merely an idle art form, it is an essential part of our society."*
>
> R. W. Hamming[1].

Learning Objectives. This course will give a straightforward introduction to error detecting and error correcting codes. We will begin with some basic definitions and examples of codes. There will then be three main parts.

(A) Further examples of codes. Error detection and error correction and connection with Hamming distance and Hamming balls. Information rate and the binary symmetric channel.

(B) The Main Coding Theory Problem. Singleton bound and codes based on Latin squares. Plotkin bound and Hadamard codes. Hamming and Gilbert–Varshamov bounds.

(C) Linear codes. Generator matrices and encoding. Cosets and decoding by standard arrays. Parity check matrices and syndrome decoding. Hamming codes. Dual codes.

The MT461/MT5461 course has extra material on Reed–Solomon codes and cyclic codes. Questions on problem sheets and handouts on Moodle labelled **MSc/MSci** are on this extra material and can safely be ignored if you are doing MT361.

Recommended Reading.

[1] *Combinatorics: Topics, Techniques, Algorithms.* Peter J. Cameron, CUP, 1994. (Chapter 17 gives a concise account of coding theory.)

[2] *Coding and Information Theory.* Richard W. Hamming, Prentice-Hall, 1980. (Chapters 2, 3 and 11 are relevant to this course.)

[3] *A First Course in Coding Theory.* Raymond Hill, OUP, 1986. (Highly recommended. It is very clear, covers all the 3rd year course, and the library has several copies.)

[1] *Mathematics on a Distant Planet*, Amer. Math. Monthly, **105** (1998) 640–650, bottom of page 640.

[4] *Coding Theory: A First Course.* San Ling and Chaoping Xing, CUP, 2004.

[5] *The Theory of Error-Correcting Codes.* F. J. MacWilliams and N. J. A. Sloane, North-Holland, 1977. (For reference.)

Hamming's original paper, *Error Detecting and Error Correcting Codes*, Bell Systems Technical Journal, **2** (1950) 147–160, is beautifully written and, for a mathematics paper, very easy to read. It is available from `www.lee.eng.uerj.br/~gil/redesII/hamming.pdf`.

Prerequisites.

- Basic discrete probability.
- Modular arithmetic in $\mathbf{Z}_p$ where p is prime. If you are happy with calculations such as $5 + 4 \equiv 2 \bmod 7$, $5 \times 4 \equiv 6 \bmod 7$ and $5^{-1} \equiv 3 \bmod 7$, that should be enough.
- Some basic linear algebra: matrices, vector spaces, subspaces, row-reduced echelon form. This will be reviewed when we need it in Part C of the course.

Problem sheets and exercises. There will be eight marked problem sheets, one sheet for the vacation, and one preliminary sheet that you can mark for yourself. Exercises set in these notes are intended to be basic tests that you are following the material. Some will be gone through in lectures: please attempt all the rest yourself.

Note on optional questions. Optional questions on problem sheets are included for interest and to give extra practice. Harder optional questions are marked $(\star)$. **If you can do the compulsory questions and know the bookwork, i.e. the definitions, main theorems, and their proofs, as set out in the handouts and lectures, you should do very well in the exam.**

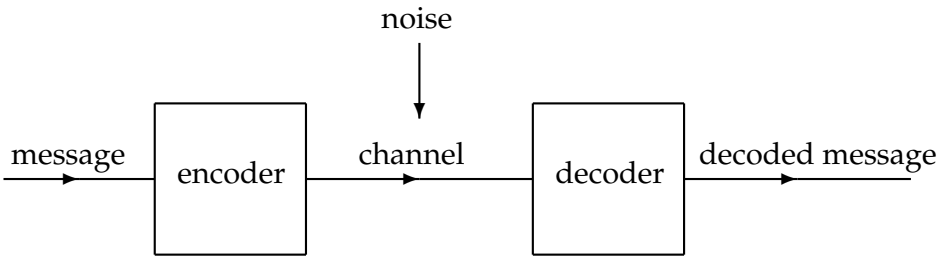
1. INTRODUCTION

The basic problem we will consider in this course is as follows.

Problem 1.1. Alice wants to send a message to Bob. She can communicate with him by sending him a word formed from symbols taken from some fixed set. But every time she sends a word, there is a chance that some of its symbols will be corrupted, so the word that Bob receives may not be the word that Alice sent. How can Alice and Bob communicate reliably?

The symbols are sent through a *channel*. The channel could be a phone line, a fibre-optic cable, the air in a room (the medium through which sound-waves travel), a compact-disc, and so on. The errors could come from human error, imperfections in the equipment, aeroplanes flying overhead, scratches on the disc, and so on. These unwanted phenomena are called *noise*.

Our basic setup is shown in the diagram below.



Example 1.2. Alice wants to send the message 'Yes' or 'No' to Bob. The available symbols are 0 and 1, and we will imagine that the channel is a noisy phone-line to which Alice and Bob have connected their respective computers.

Scheme 1. The two decide, in advance, that Alice will send

- 00 for 'No',
- 11 for 'Yes'.

If Bob receives 00 or 11 then he will assume this is the word that Alice sent, and decode her message. If he receives 01 or 10 then he knows an error has occurred, but he does not know which symbol is wrong. If he can get in touch with Alice to ask her to resend the message, this may be acceptable.

Scheme 2. Suppose instead they decide that Alice will send

- 000 for 'No',
- 111 for 'Yes'.

Then Bob can decode Alice's message correctly, provided at most one error occurs, by assuming that the symbol in the majority is correct.

Under either scheme, if two errors occur then, when Bob decodes the received word, he gets the wrong message.

Like all mathematical subjects, coding theory has its own technical vocabulary. Fortunately most of the words have fairly intuitive meanings, but as always, you will need to think carefully about the definitions, and refer to them often when doing questions.

Definition 1.3. Let $q \in \mathbf{N}$. A q-ary alphabet is a set of q different elements, called *symbols*. A *word* of *length* n over an alphabet A is a sequence $(x_1, x_2, \ldots, x_n)$ where $x_i \in A$ for each i.

Equivalently, a word is an element of $A^n = A \times A \times \cdots \times A$. We will usually omit the round brackets and commas when writing words. For example, in Example 1.2 the alphabet is $\{0, 1\}$ and the word $(1, 1, 1)$ of length 3, corresponding to 'Yes' in Scheme 2, is 111.

Definition 1.4. Let A be an alphabet and let $n \in \mathbf{N}$. A *code over* A of *length* n is a subset C of A^n containing at least two words. The elements of C are called *codewords*. The *size* of C is $|C|$.

See Remarks 1.7 for some discussion of this definition.

Exercise: Why is it reasonable to require that a code always has at least two codewords?

Definition 1.5. The *binary alphabet* of ***bi**nary digi**ts***, or *bits*, is $\{0, 1\}$. A *binary code* is a code over $\{0, 1\}$.

The binary alphabet and binary codes are particularly important because modern computers store and send data as sequences of bits.

We will now see how Example 1.2 is described using the language of coding theory.

Example 1.2 (continued). In *Scheme 1*, Alice and Bob use the binary code

$$C = \{00, 11\}$$

which has length 2 and size 2. The encoder is defined by

$$\text{'No'} \overset{\text{encoded as}}{\longmapsto} 00$$

$$\text{'Yes'} \overset{\text{encoded as}}{\longmapsto} 11.$$

The decoder decodes 00 as 'No' and 11 as 'Yes'. If 01 or 10 is received then rather than take a blind guess, Bob requests retransmission.

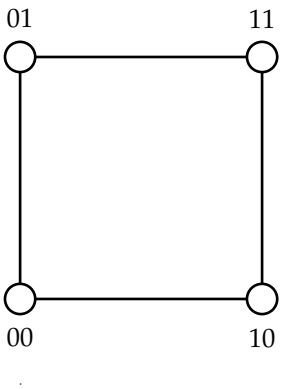
In *Scheme 2*, Alice and Bob use the binary code

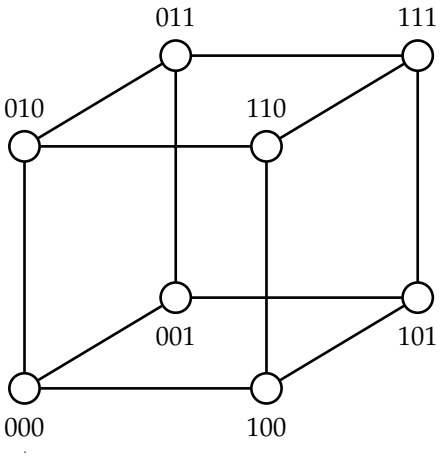
$$D = \{000, 111\}$$

which has length 3 and size 2. The encoder encodes 'No' as 000 and 'Yes' as 111. The decoder decodes a received word according to its majority symbol, so if Bob receives

$000, 001, 010, 100$	he assumes Alice sent 000 and decodes as 'No'.
$111, 110, 101, 011$	he assumes Alice sent 111 and decodes as 'Yes'.

In Section 2 we will define the Hamming distance between two words of the same length and use this to generalize the decoding strategy in Scheme 2. You may be able to guess the definition of Hamming distance (for binary words) from the diagrams showing $\{0,1\}^2$ and $\{0,1\}^3$ below.





Example 1.2 (concluded). Suppose that whenever a bit 0 or 1 is sent down the channel used by Alice and Bob, there is a probability p that it flips, so a 0 becomes a 1, and a 1 becomes a 0.

Exercise: Why is it reasonable to assume that $p < 1/2$?

For definiteness we shall suppose that Alice sends 'Yes' to Bob: you should be able to check that we get the same behaviour if Alice sends

'No'. Using Scheme 2, Alice sends 111 and Bob decodes wrongly if and only if he receives 000, 001, 010 or 100. This event has probability

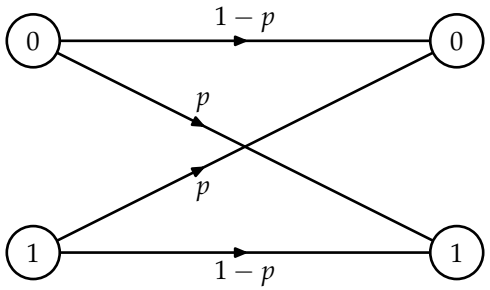
$$p^3 + 3p^2(1-p).$$

The preliminary problem sheet leads you through a step-by-step analysis of Scheme 1 and asks you to compare it with Scheme 2.

The channel used in this example is a good model for a number of physically different channels.

Definition 1.6. The binary channel in which each transmitted bit flips, so a 0 becomes a 1 and a 1 becomes a 0, independently with probability p, is called the *binary symmetric channel* with *cross-over probability* p.

The transition probabilities for the binary symmetric channel are shown in the diagram below. Question 4 on Problem Sheet 1 asks you to work out the analogous definition of a q-ary symmetric channel.



More general channels, in which symbols can be erased, as well as flipped into other symbols, are also studied, but we shall not look at them in this course.

Remarks 1.7. The following features of Definition 1.4 should be noted.

(1) By Definition 1.4, all the codewords in a code have the same length. This is true for all modern codes, and is a helpful simplifying assumption.

(2) We assume that all our codes have size ≥ 2, because if a code has no codewords, or only one, then it is useless for communication.

(3) It is very important to realise that the codes in this course **are not secret codes**. The set of codewords, and how Alice and Bob plan to use the code to communicate, should be assumed to be known to everyone.[2]

(4) The definition of a code does not mention the encoder or decoder. This is deliberate: the same code might be used for different sets of messages, and with different decoding strategies: see Example 1.8.

Example 1.8. Suppose ALICE wants to send BOB one of the messages 'Stand-down' or 'Launch nukes'. They decide to use the binary code $D = \{000, 111\}$ from Example 1.2, with the encoder

$$\text{'Stand-down'} \overset{\text{encoded as}}{\longmapsto} 000$$

$$\text{'Launch nukes'} \overset{\text{encoded as}}{\longmapsto} 111.$$

Erring on the side of safety, they decide that if BOB receives a non-codeword (i.e. one of $001, 010, 100, 110, 101, 011$), then he will request retransmission. So the same code is used as Example 1.2, but with different messages, a different encoder and a different decoding strategy.

The following two exercises will be discussed in lectures.

Exercise: Alice thinks of a number between 0 and 15. Playing the role of Bob, how many yes/no questions do you need to ask Alice to find out her number?

Exercise: Now suppose that Alice is allowed to tell *at most* one lie when she answers Bob's questions. (Or, corresponding more closely to noise in the channel, suppose that Alice can choose to mumble in one of her answers so that Bob mishears her answer.) Repeat the game in the previous exercise. How many questions do you need?

Example 1.9. We will convert some of the possible questioning strategies for Bob into binary codes of size 16.

[2] Of course there is nothing to stop Alice encrypting her message to Bob before it is encoded for the channel. In addition, we will see that it is often important that Alice is roughly equally likely to send each of her possible messages: this is achieved by *source encoding*, which is the subject of MT441 Channels. This course is about the innermost step in the chain of communication.

At the end of the course we will see the Hamming code of length 7 and size 16. This gives Bob an optimal questioning strategy with the surprising property that he can write down all his questions in advance.[3] (So the questions he asks do not depend on how Alice replies.)

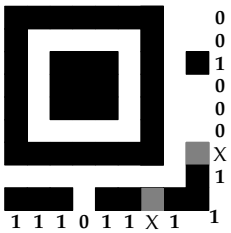
Definition 1.10. Let C be a code of length n and size M. We define the *rate* of C to be $(\log_2 M)/n$.

Roughly put, the rate of a binary code measures the proportion of the bits of the codewords that give direct information about the encoded message. For example, the binary codes $\{00, 11\}$ and $\{000, 111\}$ used by Alice and Bob in Example 1.2 have rates $1/2$ and $1/3$, respectively. A code of length n used for the guessing game has size 16, so has rate $(\log_2 16)/n = 4/n$.

We end with four examples of codes that are, or were, important in real life. I hope you will find these examples interesting, but you are **not** expected to know anything about them for examination purposes. *Exercise:* read the examples and think about how each fits into our setup of encoder, code, channel and decoder.

Example 1.11. A *Quick-Response code*, or *QR-code* is a 21×21 grid of small black and white squares. The squares represent a binary sequence (black is 1 and white is 0) encoding a short message. If your mobile phone has a camera, you might be able to use it to decode the QR-code below.[4]

Each QR-code has a 15-bit *format string*, repeated twice in the grid. The format string in this example is 111011111000100. (One place to find it is in 15 of the 17 bits anticlockwise around the top-left square, as shown in the margin.) The first 5 bits of the format string specify the coding scheme used for the main message. The remaining 10 bits are used for error correction. After a fixed masking pattern is added, the format string becomes a single codeword in the binary BCH $[15, 5, 7]$-code of



[3] Try searching for 'Ulam's Liar Game' on the web to find some general results.

[4] Free decoding apps are available for most phones, or you could use `zxing.org/w/decode.jspx`.

length 15 and size 32. This code will be seen at the end of the MSc/MSci course.

The remaining data in this QR-code consists of $26 \times 8 = 208$ bits which correspond to a single codeword in a Reed–Solomon code of length 26 over an alphabet of size 2^8. This codeword encodes a message of upper-case characters and some limited punctuation, suitable for transmitting web addresses.

The encoder begins by converting each character in the message into a number between 0 and 44: for example, 'C' is sent to 12, 'D' to 13, and so on. Each pair of numbers is then converted into an 11 bit binary word. (This is possible because $45^2 = 2025 \leq 2^{11} = 2048$.) These words are then concatenated into a string of $19 \times 8 = 152$ bits which is encoded using the Reed–Solomon code.

This describes QR-codes with the lowest degree of error correction. The decoder will always be able to correct up to three errors, in which a black square is read as a white square, or *vice versa*. For reasons that will be mentioned in the MSc/MSci course, many more errors can often be corrected, but this is not guaranteed.

Exercise: The QR-code below encodes a very short message with a higher degree of error correction. Try shading in some little squares (or randomly writing on the code). You should be able to inflict quite a lot of damage before decoding fails.

Exercise: Why do you think the format string is repeated twice? What is the point of the three squares in the corners? (Including the white border, these squares occupy $8^2 \times 3 = 192$ of the $21^2 = 441$ little squares, so they must be important for some reason!)

A MATHEMATICA notebook that implements a basic QR-encoder is available from Moodle.[5]

[5] Although it is not the most exciting read, it is clear from the official specification that a lot of thought went into the design of the QR-code. The hard way to obtain this document is to pay ISO (International Standards Organization) 210 Swiss Francs and wait one month.

Example 1.12. A compact disc contains information in the form of a sequence of microscopic pits on a disc that are read by a laser. Here the compact disc is the channel, and its primary purpose is to transmit information reliably through time.

The pits encode a long sequence of the bits 0 and 1. The encoding scheme combines two different Reed–Solomon codes. The first code alone guarantees to correct one error in each block of 128 consecutive bits. If, however, the errors occur in adjacent bits, as is usual for a scratch then, mainly because of the clever way in which the two codes are combined, many more errors can be corrected. Up to $16 \times 32 \times 8 = 4096$ adjacent bits can be corrupted in a block of $124 \times 28 \times 8 = 27776$ bits and the compact disc will still be decoded successfully.[6]

Example 1.13. The Australian railway company 'Victorian Railways' used a telegraph system and codebook. The entire codebook, as used in 1972, can be read online at `www.railpage.org.au/telecode/tc01.gif`. Here is an extract from near the start.

Ayah	Provide locomotive to work
Aybu	Return locomotive at once
Azaf	Breakdown train left at ...
Azor	Arrange to provide assistance locomotive
Azub	A second locomotive will be attached to ...

In telegraph transmission only upper case were used. So a typical message might be something like 'Breakdown train left at Sydney, provide locomotive to work', encoded as AZAF SYDNEY AYAH. The code has these properties.

(1) All codewords are of length 4 and are words over the alphabet $\{A, B, C, \ldots, X, Y, Z\}$. (Except for place names such as SYDNEY, which break our rule that all codewords must have the same length.)

(2) The codewords are easily pronounceable. Most, but not all, have the pattern vowel–consonant–vowel–consonant. Probably this reduced operator error when encoding messages.

[6] Larger numbers of errors on audio compact discs are hidden by interpolating the music on either side of the error. Compact discs used to store data are physically the same as audio discs, but have a further layer of coding that enables even more errors to be corrected. See Chapter 5 of *Error Correcting Codes: Classification by Isometry and Applications*, Anton Betten *et al*, Algorithms and Computation in Mathematics Volume 18, Springer 2006 for a nice account.

(3) Most codewords differ from one another in at least two letters. So if I mean to send '**Ayah**', but because of human error, or a problem with the line, '**Ayam**' is received, then it will be clear an error has occurred.

Exercise: Most codewords are not English words, although a few are: '**Coma**' is an instruction about extra trucks, '**Cosy**' is an instruction about loading trucks. Why do you think English words were usually avoided?

Exercise: Related instructions often start with the same letter: is this a good feature of the coding scheme?

The high degree of redundancy in normal English text means that a native speaker can detect and correct many errors. For example, even though the sentence

'Tae mext hrxin tb Lxnton kas &e*n oznce!Hed'

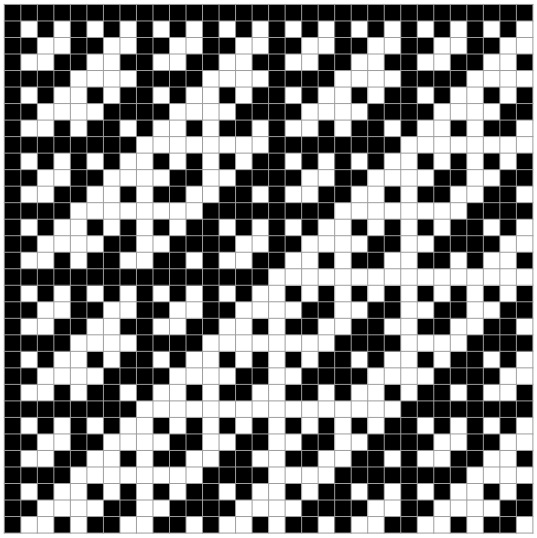
has many errors, you can probably work out what it says. Here it is worth bearing in mind that what is easy for a human may be very hard to program on a computer.

The Australian telegraph code removes much of the redundancy from English, in the interests of efficient use of the telegraph lines. However, it is still the case that only a tiny number of all four letter strings are used, so the code can still detect errors. The encoding process, which turns long sentences about train manoeuvers into four letter codewords, replaces the complicated (and poorly understood) redundancy of English with redundancy over which we have more control.

Example 1.14. The Mariner 9 probe, launched in 1971, took the first pictures of Mars, ultimately transmitting 7239 pictures at a resolution of 700×832 pixels. The images were grey-scale, using 64 different shades of grey. The pictures were transmitted back to Earth by sending one pixel at a time, so we can think of a message as a single number between 0 and 63. The channel of radio waves can be modelled by the binary symmetric channel in Definition 1.6.

The naïve approach of just encoding each pixel as a string of 6 bits would not have worked well. One survey article[7] gives the cross-over probability as 0.05. So about 5% of all bits were flipped by the channel (with a 0 becoming a 1, and a 1 becoming a 0). With this scheme, the probability that any particular pixel would be correctly transmitted is then $0.95^6 \approx 0.74$. So about 26% of the image would have been wrong.

[7] Van Lint, *Coding, decoding and combinatorics*, available from `alexandria.tue.nl/repository/freearticles/593591.pdf`.



The matrix shows 32 of the 64 codewords in the binary Hadamard code used in Mariner 9. A black square represents 0 and a white square represents 1. The other 32 codewords are obtained by flipping each bit in the 32 codewords shown. For example, the first row shows the codeword $00\ldots0$, which flips to $11\ldots1$. Please don't confuse this diagram with a QR-code!

It was acceptable for each pixel to be encoded by up to 32 bits, so increasing the amount of data to be stored and transmitted by a factor of 5. A code in which each bit was repeated 5 times, along the lines of the codes C and D in Example 1.2, would have reduced the probability that a pixel was incorrectly decoded to about 0.8%.

The code actually used was a binary Hadamard code of length 32 and size 64. (Half of the codewords are shown in the diagram above.) We will study these codes in Part B. This code is capable of correcting any 7 errors in a received word. So of the 32 bits sent for each pixel, even if 7 of them are corrupted by the channel, the pixel will still be correctly decoded. This reduces the probability of incorrect decoding a pixel to less than 0.014%. Most images could be expected to have fewer than 100 incorrect pixels.

When working with codes of long length and large size, it is no longer at all obvious how to decode a received word. For example, suppose you receive the word $(0,0,1,1,1,0,\ldots,1,0,1,1)$, represented by

.

It is far from obvious which codeword in the Mariner 9 code it is nearest to. In fact the received word differs from the bottom row in the matrix above in 7 positions, and from all other rows in at least 11 positions, so

would be decoded as $(0, 1, 1, 0, 1, 0 \ldots, 1, 0, 0, 1)$. (And then this codeword would be converted back into the corresponding shade of grey.)

There is a very elegant decoding algorithm for the Mariner 9 code, based on the Discrete Fourier Transform. Critically, this algorithm was easy to implement on the relatively primitive computers available to NASA in 1971. See Van Lint's survey article for an outline of how this was done. Finding fast and accurate decoders for large codes, such as Hadamard codes and the Reed–Solomon codes used on compact discs, is a central problem in coding theory and has motivated much work in the subject.

Part A: Hamming distance and error detection and correction

2. HAMMING DISTANCE AND NEAREST NEIGHBOUR DECODING

In this section we define Hamming distance and use it to give a precise statement of what it means to decode by choosing the 'closest codeword' to a received word. We will also see some more examples of codes used for error detection and correction.

In the rest of Part A we will use Hamming distance to make precise our intuitive idea that a code has a maximum number of errors it can hope to detect and correct reliably, and see different ways to calculate this number.

Definition 2.1. Let A be an alphabet. Let $u, v \in A^n$ be words of length n. The *Hamming distance* between u and v, denoted $d(u, v)$, is the number of positions in which u and v are different.

In mathematical notation, $d(u, v) = \left|\{i \in \{1, 2, \ldots, n\} : u_i \neq v_i\}\right|$. We will often abbreviate 'Hamming distance' to '*distance*'.

Example 2.2. Working with binary words of length 4, we have

$$d(0011, 1101) = 3$$

because the words 0011 and 1101 differ in their first three positions, and are the same in their final position. Working with words over the alphabet $\{A, B, C, \ldots, X, Y, Z\}$ we have $d(\text{TALE}, \text{TAKE}) = 1$ and $d(\text{TALE}, \text{TILT}) = 2$.

Exercise: Check that $d(1010, 1001) = 2$. Find the number of binary words v of length 4 such that $d(1010, v) = r$ for each $r \in \{0, 1, 2, 3, 4\}$. Do you recognize the sequence you get?[8]

The next theorem shows that Hamming distance has the expected properties of a distance. Part (iii) is the triangle inequality for Hamming distance: it will be used in Sections 3 and 4 later.

Theorem 2.3. *Let A be a q-ary alphabet and let u, v, w be words over A of length n.*

(i) $d(u, v) = 0$ *if and only if* $u = v$;
(ii) $d(u, v) = d(v, u)$;
(iii) $d(u, w) \leq d(u, v) + d(v, w)$.

[8] If not, try typing it into Google.

Exercise: Find all English words v such that

$$d(\text{WARM}, v) = d(\text{COLD}, v) = 2.$$

Check that the triangle inequality holds when u, v, w are WARM, WALL and COLD, respectively. For a connection with a Victorian word game, see Question 9 on Sheet 1.

The following important family of codes will be useful examples for nearest neighbour decoding.

Definition 2.4 (Repetition codes)**.** Let $n \in \mathbf{N}$ and let A be a q-ary alphabet with $q \geq 2$. The *repetition code* of length n over A has as its codewords all words of length n of the form

$$(s, s, \ldots, s)$$

where $s \in A$. The *binary repetition code* of length n is the repetition code of length n over the binary alphabet $\{0, 1\}$.

Note that if u and v are distinct codewords in a repetition code of length n then $d(u, v) = n$.

In Scheme 2 in Example 1.2, Alice and Bob used the binary repetition code of length 3, with codewords $\{000, 111\}$, Bob decoded a received word by its majority symbol, so for example, 101 is decoded as 111 (meaning 'Yes'). Equivalently, Bob decoded a received word *by choosing the closest codeword in the Hamming distance*.

Definition 2.5 (Nearest neighbour decoding)**.** Let C be a code. Suppose that a codeword is sent through the channel and we receive the word v. To decide v using *nearest neighbour decoding* look at all the codewords of C and pick the one that is nearest, in Hamming distance to v. If there is no unique nearest codeword to v, then we say that nearest neighbour decoding *fails*.

Note that the word 'fail' has a technical meaning in Definition 2.5. It **does not** mean that the decoded codeword is not the send codeword. This is illustrated in the next example.

Example 2.6. An internal review of the code in Example 1.8 has uncovered several deficiencies. The new proposal uses a ternary repetition code of length 6 over the alphabet $\{0, 1, 2\}$. The new encoder is

$$\text{'Stand-down'} \overset{\text{encoded as}}{\longmapsto} 000000$$
$$\text{'Stay on your toes'} \overset{\text{encoded as}}{\longmapsto} 111111$$
$$\text{'Launch nukes'} \overset{\text{encoded as}}{\longmapsto} 222222.$$

Suppose ALICE sends 'Stand-down'. If BOB receives 001102, then

$$d(001102, 000000) = 3, d(001102, 111111) = 4, d(001102, 222222) = 5,$$

so under nearest neighbour decoding, 001102 is decoded as 000000, which in turn BOB will decode as 'Stand-down'.

Now suppose 000111 is received. Then

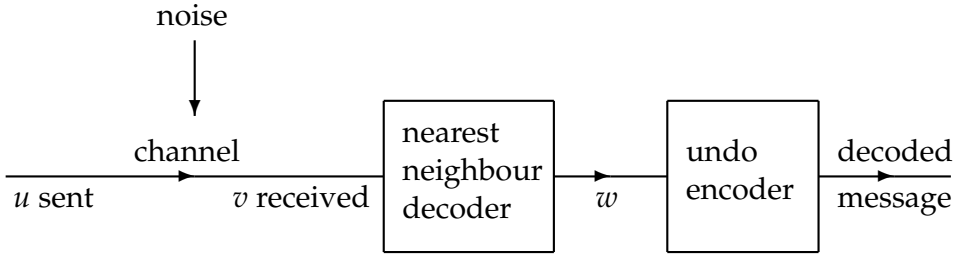
$$d(000111, 000000) = 3, d(000111, 111111) = 3, d(000000, 222222) = 6,$$

so there is no unique codeword closest to 000111. Hence nearest neighbour decoding fails.

Finally suppose four errors occur in the channel and 020222 is received. Then nearest neighbour decoding gives 222222, which BOB will decode as 'Launch nukes'. In the technical sense, nearest neighbour decoding has not failed. However, the outcome will probably not be what ALICE was hoping for.

This example shows that the ternary repetition code of length 6 can be used to correct 2 errors in a reliable way. If more errors occur than it seems that reliable decoding is impossible.

Example 2.6 shows that nearest neighbour decoding is only one step in the decoding process. The diagram below extends our model of decoding to a two-step process. We suppose that the codeword u is sent, the word v is received, and that nearest neighbour decoding gives the codeword w, which is then decoded into the message.



The decoded message is correct if and only if $w = u$. This is the case if and only if u is the unique nearest codeword to v, i.e.

$$d(u, v) < d(u', v)$$

for all codewords u'.

Nearest neighbour decoding should seem like the intuitively obvious way to decode. However, it is still interesting to give it a probabilistic justification. We shall do this in the case of binary codes, using the binary symmetric channel defined in Definition 1.6.

Lemma 2.7. *Let C be a binary code of length n used to communicate on a binary symmetric channel with cross-over probability p. Suppose that the codeword $u \in C$ is sent. If v is a word of length n, then the probability that v is received $p^{d(u,v)}(1-p)^{n-d(u,v)}$.*

In symbols, we write $\mathbf{P}[v \text{ received} \mid u \text{ sent}] = p^{d(u,v)}(1-p)^{n-d(u,v)}$. For example, if C is the binary repetition code of length 3, then according to Lemma 2.7,

$$\mathbf{P}[001 \text{ received} \mid 111 \text{ sent}] = p^2(1-p)$$
$$\mathbf{P}[000 \text{ received} \mid 111 \text{ sent}] = p^3.$$

These agree with the calculations in Example 1.2.

Theorem 2.8. *Suppose that we use a binary code C of length n to send messages through the binary symmetric channel, and that each codeword in C is equally likely to be sent. Suppose we receive a binary word v. For each $u \in C$,*

$$\mathbf{P}[u \textit{ sent} \mid v \textit{ received}] = p^{d(u,v)}(1-p)^{n-d(u,v)}C(v)$$

where $C(v)$ does not depend on u. Hence $\mathbf{P}[u \textit{ sent} \mid v \textit{ received}]$ is maximized by choosing u to be the nearest codeword to v.

Thus, provided we accept that maximizing $\mathbf{P}[u \text{ sent} \mid v \text{ received}]$ is a good idea, we are inevitably led to nearest neighbour decoding. The syllabus talks only about 'probability calculations', so while interesting, Theorem 2.8 may be regarded as non-examinable.

The assumption in Theorem 2.8 that each codeword is equally likely to be transmitted is critical. See Question 9 on Sheet 2 for a case where

nearest neighbour decoding no longer works well because one codeword is much more likely to be sent than another.[9]

One final remark about nearest neighbour decoding. In practice it might be essential that the decoder always gives some result, even if nearest neighbour decoding has failed. Then the decoder will have to make an arbitrary choice between two or more codewords that are equally likely to be the sent word. Mathematically it is much better just to report that nearest neighbour decoding has failed.

We end this section with two important examples of codes that are used only for error detection, not error correction.

Example 2.9 (Parity check codes)**.** Let $n \in \mathbf{N}$ and let C be the binary code consisting of all binary words of length n. Let C_{ext} be the code of length $n+1$ whose codewords are obtained by appending an extra bit to each codeword in C, so that the total number of 1s in each codeword is even.

For instance, if $n = 4$ then C is the binary code of size 16 consisting of all binary words of length 4. The extended code C_{ext} has length 5 and the same size as C, namely 16. Its codewords are

$$00000, 00011, 00101, 00110, \ldots, 11101, 11110.$$

Suppose that a codeword $u \in C_{\text{ext}}$ is sent through the channel and the word v is received. If a single bit is corrupted, so $d(u, v) = 1$, then v must have an odd number of 1s. Hence $v \notin C_{\text{ext}}$ and we detect that an error has occurred.

Exercise: How would you use the length 5 code C_{ext} to encode a number between 0 and 15? Try to specify an encoding algorithm that is easy to perform in practice!

[9] This is related to a famous statistical paradox: suppose there is an illness that infects one person in a thousand. If a test for the illness always identifies infected people, but gives a false positive with probability 1/500, then, when 1000 people are tested, there will, on average, be one infected person, and two false positives. So any particular person who tests positive for the illness still has a 2/3 chance of being healthy. To translate this into a coding problem, imagine that we send 'healthy' with probability 999/1000 and 'ill' with probability 1/1000, but whenever 'healthy' is sent, it has a 1/500 chance of being corrupted into 'ill'. Then when we receive 'ill' it is still more likely than not that 'healthy' was sent.

Example 2.10 (ISBN-10 code)**.** All recent books have an International Standard Book Number (ISBN) assigned by the publisher. The coding system changed in 2007 because the old scheme was running out of space. However, the older ISBN-10 code is mathematically more interesting, so we will use it. In this scheme, each book is assigned a codeword of length 10 over the 11-ary alphabet $\{0, 1, 2, \ldots, 9, X\}$.

For example, [5] in the list of recommended reading has ISBN

0-444-85193-3.

Here

- 0 identifies the country of publication;
- 444 identifies the publisher;
- 85193 is the item number assigned by the publisher;
- 3 is the *check digit*.

The hyphens are put in to make the ISBN more readable; they are **not** part of the code. For us, the important feature is the check digit. It is chosen so that if $u_1u_2u_3u_4u_5u_6u_7u_8u_9u_{10}$ is an ISBN then

$$\sum_{j=1}^{10}(11-j)u_j = 10u_1 + 9u_2 + \cdots + 2u_9 + u_{10} \equiv 0 \bmod 11.$$

There is one technical point: it might be necessary to take 10 as a check-digit. In this case the letter X is used to stand for 10 (it is never used in the main part of an ISBN).

We will say that $u_1u_2u_3u_4u_5u_6u_7u_8u_9u_{10}$ is an *ISBN* if it satisfies the check condition above, ignoring the question of whether it was ever assigned to a book.

Lemma 2.11. *If a single error is made when writing down an ISBN, the result is not an ISBN.*

The ISBN code also detects when two unequal adjacent symbols are swapped. (See Question 5 on Sheet 2.) This is a sort of error likely to be made by a busy person. However it does not detect all errors involving two symbols. For example, starting from 0000000000 we can change two symbols to get 1000000001, which is also an ISBN.

Exercise: In the next section we will see the accepted definitions of what it means for a code C to be t-error detecting or t-error correcting. From the examples you have seen so far, how would you define these terms?

3. t-ERROR DETECTING AND t-ERROR CORRECTING CODES

Here is the accepted answer to the exercise on the previous page.

Definition 3.1. Let C be a code of length n over an alphabet A and let $t \in \mathbf{N}$. We say that C is

- *t-error detecting* if whenever $u \in C$ is a codeword, and v is a word of length n over A such that $v \neq u$ and $d(u, v) \leq t$, then $v \notin C$.
- *t-error correcting* if whenever $u \in C$ is a codeword, and v is a word of length n over A such that $d(u, v) \leq t$, then v is decoded to u using nearest neighbour decoding.

Equivalently, a code C is t-error detecting if whenever a codeword is sent, and **between** 1 **and** t **errors occur**, the received word is not a codeword. So the receiver will know that something has gone wrong in the channel.

Some further remarks intended to clarify Definition 3.1 are below.

Remarks 3.2.

(1) Recall that when there is no unique nearest codeword to u then nearest neighbour decoding fails. So a code C is t-error correcting if and only if whenever v is a word within distance t of a codeword $u \in C$ then

$$d(u, v) < d(u', v) \quad \text{for all } u' \in C \text{ with } u' \neq u.$$

This gives a more abstract way to state the definition of t-error correcting that does not mention nearest neighbour decoding.

(2) It may seem a bit odd to say that a code is 't-error correcting' when it is the decoder (be it a human or a computer) that has to do all the work of decoding. Moreover, we have seen in Example 1.6 that the same code can reasonably be used with different decoders. A code that is t-error correcting promises to be able to correct up to t-errors *provided nearest neighbour decoding is used.*

(3) In both definitions we have $d(u, v) \leq t$, so v is obtained from u by changing **up to** t positions. Hence if $s < t$ then a t-error detecting code is also s-error detecting, and a t-error correcting code is also s-error correcting.

If instead we required **exactly** t changes then, by Question 11 on Sheet 1, there would be codes that are 2-error detecting but

not 1-error detecting. This could be confusing in practical applications. Mathematically, it would lead to theorems with long-winded hypotheses of the form 'Suppose C is a code that is t-error detecting for all $t \leq c, \ldots$'. Both are undesirable.

We will now show that Definition 3.1 agrees with our findings in Examples 2.6 and 2.9.

Lemma 3.3. *Let $n \in \mathbf{N}$. Let C be the repetition code of length n over a q-ary alphabet A, where $q \geq 2$.*

(i) *C is $(n-1)$-error detecting but not n-error detecting.*
(ii) *If $n = 2m+1$ then C is m-error correcting but not $(m+1)$-error correcting.*
(iii) *If $n = 2m$ then C is $(m-1)$-error correcting, but not m-error correcting.*

The proof of part (iii) is left to you in Question 2 of Sheet 2.

Lemma 3.4. *Let $n \in \mathbf{N}$ and let C_{ext} be the binary parity check code of length $n+1$ defined in Example 2.9. Then C_{ext} is 1-error detecting but not 2-error detecting. It is not 1-error correcting.*

We saw in Lemma 2.11 that if a single error is made when writing down an ISBN, the resulting word is not an ISBN. So the ISBN code is 1-error detecting. By Question 5 on Sheet 2, some double errors can be detected, but as the example on page 20 shows, this is not guaranteed. So, according to Definition 3.1, the ISBN code is *not* 2-error detecting.

Lemma 3.5. *The ISBN code is 1-error detecting but not 2-error detecting. It is not even 1-error correcting.*

4. Minimum distance and Hamming balls

Question 1 on Sheet 2 shows that if C is a code such that $d(u, u') \geq 3$ for all distinct $u, u' \in C$, then C is 2-error detecting and 1-error correcting. This is a special case of a very useful general result. We need the following definition.

Definition 4.1. Let C be a code. The *minimum distance* of C, denoted $d(C)$, is defined by $d(C) = \min\{d(u, w) : u, w \in C, u \neq w\}$.

By Definition 1.4, any code has at least two codewords, so the minimum distance of a code is always well-defined.

Example 4.2. Here is an example from Hamming's original paper (see reference on page 3). Let C be the binary code of length 3 with codewords

$$001, \quad 010, \quad 100, \quad 111$$

as seen on Sheet 1, Question 2. Then $d(u, w) = 2$ for all distinct u and w in C, so $d(C) = 2$. If we adjoin 000 as another codeword then the minimum distance goes down to 1 since $d(000, 001) = 1$.

It is not hard to find the minimum distance of the codes seen in the examples so far.

Lemma 4.3. *Let $n \in \mathbf{N}$.*

(i) *The minimum distance of any length n repetition code is n.*

(ii) *The minimum distance of the length $n + 1$ binary parity check code C_{ext} in Example 2.9 is 2.*

(iii) *The minimum distance of the square code is 3.*

To add to the results in the lemma above, Question 3 on Sheet 2 gives a step-by-step proof that the 1-error correcting code of length 9 seen in Example 1.9 has minimum distance 3.

There is a special notation for recording the most important parameters of a code.

Notation 4.4. If C is a code of length n, size M and minimum distance d, then C is said to be a (n, M, d)-code.

For example, a repetition code of length n over a q-ary alphabet is a (n, q, n)-code, and the binary parity check code of length $n + 1$ in Example 2.6 is a $(n, 2^n, 2)$-code. The square code is a $(8, 16, 3)$-code.

Exercise: Let C be a code. Bearing in mind all the examples seen so far, what do you think is the relationship between the maximum t for which C is t-error detecting and the minimum distance of C? Now think about the same question with 'error detecting' replaced with 'error correcting'.

The solution to this exercise is revealed in the next theorem. The proof of forwards '$\Longrightarrow$' direction in (ii) will be postponed to the end of this section.[10]

Theorem 4.5. *Let C be a code with minimum distance d. Let $t \in \mathbf{N}$.*
(i) *C is t-error detecting $\iff$ $t \leq d-1$;*
(ii) *C is t-error correcting $\iff$ $2t \leq d-1$.*

It is usually easier to compute the minimum distance of a code than it is to determine (without using any other results) the maximum t for which it is t-error detecting or correcting. This makes the '$\Longleftarrow$' directions in Theorem 4.5 very useful.

Recall that if $x \in \mathbf{R}$ then $\lfloor x \rfloor$ is the greatest integer n such that $n \leq x$. For instance $\lfloor 2 \rfloor = \lfloor 2\frac{1}{2} \rfloor = 2$.

Corollary 4.6. *A code of minimum distance d is $(d-1)$-error detecting and $\lfloor \frac{d-1}{2} \rfloor$-error correcting.*

The table below (taken from Hamming's original paper) shows the maximum number of errors a code of small minimum distance can detect and correct.

$d(C)$	error detection / correction
1	no detection possible
2	1-error detecting
3	2-error detecting / 1-error correcting
4	3-error detecting / 1-error correcting
5	4-error detecting / 2-error correcting

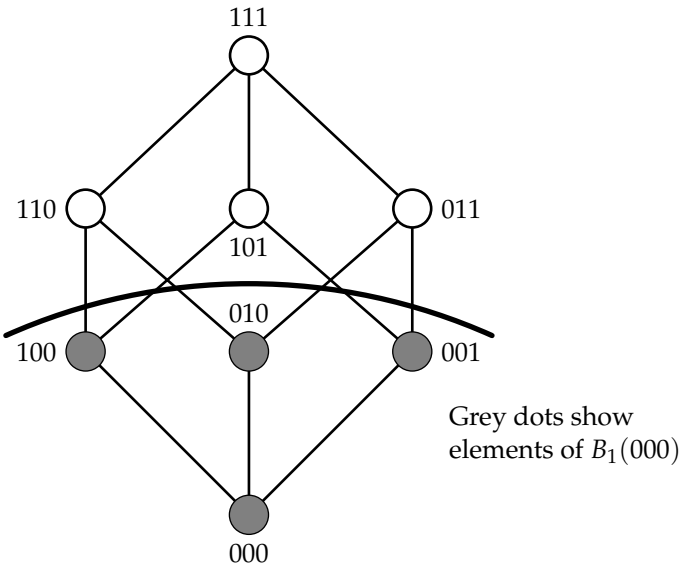
We end Part A with a more geometric way of looking at Hamming distance and t-error correcting codes.

Definition 4.7. Let A be a q-ary alphabet and let u be a word of length n. The *Hamming ball of radius t about u* is

$$B_t(u) = \{v \in A^n : d(u,v) \leq t\}.$$

[10] Remember that if P and Q are mathematical statements then $P \Longrightarrow Q$ means 'if P is true, then Q is true' and $P \Longleftarrow Q$ means 'if Q is true, then P is true'. You can think of $P \Longrightarrow Q$ as a sort of promise or contract: 'if you tell me that P is true, then I promise you that Q is true'. The double arrow $P \Longleftrightarrow Q$, read as 'if and only if' means that both $P \Longrightarrow Q$ and $P \Longleftarrow Q$ are true.]

An equivalent definition is that $B_t(u)$ consists of all words that can be obtained from u by changing up to t of its positions. The diagram below shows the Hamming ball of radius 1 about 000.



Example 4.8. The Hamming balls about the binary word 0000 are

$$B_0(0000) = \{0000\}$$
$$B_1(0000) = \{0000, 1000, 0100, 0010, 0001\},$$
$$B_2(0000) = B_1(0000) \cup \{1100, 1010, 1001, 0110, 0101, 0011\}$$
$$B_3(0000) = B_2(0000) \cup \{1110, 1101, 1011, 0111\}$$

and $B_4(0000)$ is the set of all binary words of length 4.

(ii) If $u = 1010$ then

$$B_0(1010) = \{1010\},$$
$$B_1(1010) = \{1010, 0010, 1110, 1000, 1011\},$$
$$B_2(1010) = B_1(1010) \cup \{0110, 0000, 0011, 1100, 1111, 1001\}.$$

Also $B_3(1010)$ consists of all binary words of length 4 *except* 0101, and $B_4(1010)$ is the set of all binary words of length 4.

In §5 we will use the lemma below to prove the remarkable Hamming Packing Bound on the maximum size of a t-error correcting code of length n. (The converse to Lemma 4.9 also holds, as will be seen at the end of this section.)

Lemma 4.9. *Let C be a code. If C is t-error correcting then for all distinct codewords $u, u' \in C$, the Hamming balls $B_t(u)$ and $B_t(u')$ are disjoint.*

In lectures we will use Lemma 4.9 to complete the proof of Theorem 4.5(ii) by showing that if C is a t-error correcting code of minimum distance d then $2t \leq d - 1$.

SUMMARY OF PART A. In Part A we have seen the formal definition (Definition 3.1) of t-error detecting and correcting codes. The examples in §2 and the results in Lemmas 3.3 and 3.4 show that this definition is a reasonable one. We then saw other ways of thinking about t-error correcting codes, using minimum distance (Definition 4.1) and Hamming balls (Lemma 4.9). These are summarised in the following theorem.

Theorem 4.10. *Let C be a code. The following are equivalent*

(a) *C is t-error correcting;*
(b) *Nearest neighbour decoding always gives the sent codeword (without failing), provided at most t errors occur;*
(c) *If $u \in C$ and $d(u, v) \leq t$ then $d(u, v) < d(u', v)$ for all $u' \in C$ such that $u' \neq u$;*
(d) *If $u, u' \in C$ are distinct codewords then the Hamming balls $B_t(u)$ and $B_t(u')$ are disjoint;*
(e) *The minimum distance of C is at least $2t + 1$.*

Proof. Part (b) is a slightly more informal way to state Definition 3.1, and (c) is the equivalent restatement mentioned in Remarks 3.2(1). So (a), (b) and (c) are all equivalent. For (d) and (e) we use results already proved:

- (a) $\Longrightarrow$ (d): see Lemma 4.9.
- (d) $\Longrightarrow$ (e): this was proved immediately after Lemma 4.9.
- (e) $\Longrightarrow$ (a): see the part of Theorem 4.5(ii) proved in lectures. □

Exercise: What would you say to someone who objected: 'this proof is incomplete: the argument given has no part showing that (e) $\Longrightarrow$ (d)'?

The idea of a t-error correcting code, as defined in Definition 3.1, is fundamental in coding theory. Depending on the context, it is useful to think about it in different ways. Doing this also helps one to understand what it means.

The original definition uses nearest neighbour decoding. It shows the algorithmic side of the subject and is informed by probabilistic ideas. Parts (d) and (e) show the geometric side of the subject. We have seen, for example in Lemma 4.3, that (e) is often the easier condition to work with; we will use it throughout Parts B and C.

Part B: Main Coding Theory Problem

5. Main Coding Theory Problem and Hamming's bound

Theorem 4.5 and Corollary 4.6 show that the maximum number of errors a code can detect or correct is determined by its minimum distance. Intuitively it should seem reasonable that if a code has large minimum distance then it cannot have too many codewords. In this part of the course, we shall prove a number of bounds that make this precise.

Problem 5.1. The *Main Coding Theory Problem* is to find codes over a given alphabet with

(1) small length;
(2) large size;
(3) high minimum distance.

Equivalently, we want to find (n, M, d)-codes over the given alphabet with small n, high M and high d.

Remark 5.2. Another desirable property is that there should be an efficient way to perform nearest neighbour decoding on received words. For example, the Reed–Solomon code used on compact discs has the largest possible size for its length and minimum distance. There are now efficient decoding algorithms, but when it was first invented, it was impractical because there was no good way to decode received words.

The Main Coding Theory Problem is hard because the requirements are conflicting. We shall begin by proving Hamming's Packing Bound which gives an upper bound on the size of a binary code with specified length and minimum distance.[11]

In the next lemma we need binomial coefficients. Recall that the binomial coefficient $\binom{n}{k}$ is the number of ways to choose k objects from a set of size n. It is given by the formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \quad \text{for } 0 \leq k \leq n.$$

[11] This bound can be extended easily to codes over a general q-ary alphabet: see Question 8 on Sheet 4. Only the binary case is examinable.

Lemma 5.3. *Let u be a binary word of length n. The number of words in the Hamming ball $B_t(u)$ of radius t about u is*

$$\sum_{k=0}^{t} \binom{n}{k}.$$

We also need (e) $\Longrightarrow$ (d) from Theorem 4.10: if C has minimum distance at least $2t+1$ then the Hamming balls of radius t about distinct codewords are disjoint.

Theorem 5.4 (Hamming's Packing Bound)**.** *Let C be a binary (n, M, d)-code. If $e = \lfloor \frac{d-1}{2} \rfloor$ then*

$$M \leq \frac{2^n}{\sum_{k=0}^{e} \binom{n}{k}}.$$

By Theorem 4.5(ii) that a 1-error correcting code has minimum distance at least 3. Hamming's bound therefore implies that a 1-error correcting binary code of length n has size at most $2^n/(1+n)$. For example, if $n = 7$ we get $M \leq 2^7/(1+7) = 2^4 = 16$ and if $n = 5$ we get $M \leq 2^5/(1+5) = 5\frac{1}{3}$. Since it is impossible to have $\frac{1}{3}$ of a codeword, we can tighten this bound to $M \leq 5$.

The table below shows some other values of Hamming's bound for a 1-error correcting binary code.

length n	3	4	5	6	7	8	9	10
bound on size M	2	3	5	9	16	28	51	93

It is very important to realise that while Hamming's bound is a necessary condition for a binary code of specified length, size and minimum distance to exist, it is **not in general sufficient**.

Exercise: According to Hamming's Packing Bound, the largest possible size of a binary code of length 4 and minimum distance 3 is 3. Show that in fact the largest size of a binary code of length 4 and minimum distance 3 is 2.

The following definition gives a more concise way to state Hamming's Packing Bound, and the other bounds we shall see in Part B.

Definition 5.5. Let $q \geq 2$ and let $n \in \mathbf{N}$, $d \in \mathbf{N}$ be such that $n \geq d$. We denote by $A_q(n, d)$ the largest size of a code of length n and minimum distance d over a q-ary alphabet.

Exercise: Convince yourself that $A_q(n,d)$ does not depend on which q-ary alphabet is used. Working over the q-ary alphabet $\{0,1,\ldots,q-1\}$, show that there is at least one code of length n and minimum distance d.

The previous exercise implies that $A_q(n,d)$ is well-defined.[12] We can now restate Hamming's Packing Bound as $A_2(n,d) \leq 2^n / \sum_{k=0}^{e} \binom{n}{k}$, where $e = \lfloor (d-1)/2 \rfloor$.

The table below shows some values of $A_2(n,d)$; some of these will be proved in the following sections. (You are *not* expected to memorize any part of this table!) One result visible in the table is that $A_2(n,d) = A_2(n+1,d+1)$ whenever d is odd: see the optional questions on Sheet 4.

d	$A_2(2,d)$	$A_2(3,d)$	$A_2(4,d)$	$A_2(5,d)$	$A_2(6,d)$	$A_2(7,d)$	$A_2(8,d)$
1	4	8	16	32	64	128	256
2	2	4	8	16	32	64	128
3		2	2	4	8	16	20
4			2	2	4	8	16
5				2	2	2	4
6					2	2	2
7						2	2
8							2

The values of $A_2(n,d)$ are often powers of 2 because many good codes are linear (see Part C), and all linear binary codes have size a power of 2. But $A_2(n,d)$ is not always a power of 2. For example, $A_2(8,3) = 20$; equivalently (by Theorem 4.5) the largest 1-error correcting of length 8 has 20 codewords.

6. Bounds from equivalences of codes

Looking at the first row and the main diagonal in the table on the previous page you might conjecture that the following lemma holds.

Lemma 6.1. *Let $q \geq 2$ and let $n \in \mathbf{N}$. Then*
(i) $A_q(n,1) = q^n$;
(ii) $A_q(n,n) = q$.

[12] To say that a definition is 'well-defined' means that the quantity being defined does not depend on any of the choices allowed in the definition. Here, as well as checking that the choice of alphabet is irrelevant, we also have to check that there is at least one code of length n and minimum distance d, since if there were none, it would make no sense to talk of the maximum size of such a code.

To get some further results, we need the idea of equivalences of codes.

Definition 6.2. Let C and C' be codes over a q-ary alphabet A. We say that C and C' are *equivalent* if one can be obtained from the other by repeatedly applying the following two operations to all the codewords:

(a) relabelling the symbols appearing in a fixed position;
(b) shuffling the positions within each codeword.

Exercise: Are $\{0000, 1100, 1111\}$ and $\{0000, 1100, 0011\}$ equivalent?

We are interested in equivalences because, by the following lemma, equivalent codes have the same distances between codewords.

Lemma 6.3. *If u and w are codewords in a code C, and u' and w' are the corresponding codewords in an equivalent code C' obtained by relabelling and/or shuffling positions then $d(u, w) = d(u', w')$.*

Proof. For binary words this was proved in Question 4 on Sheet 3. The only shuffles allowed in this question were swaps on two positions, but any shuffle can be obtained by repeated swaps, so this suffices. The extension to a general alphabet is routine. □

In particular, if C and C' are equivalent then C and C' have the same minimum distance. However the converse does not hold.

Example 6.4. Consider the four binary codes

$$C = \{0000, 1100, 1010, 0110\}$$
$$C' = \{1010, 0110, 0011, 1111\}$$
$$D = \{0000, 1100, 0011, 1111\}$$
$$E = \{1000, 0100, 0010, 0001\}.$$

All four codes have minimum distance 2. By applying operations (a) and (b) we will show that C and C' are equivalent. No other two of these codes are equivalent.

For C and D this can be shown quite easily: there are codewords in D that are distance 4 apart, for example $d(0000, 1111) = 4$, but all codewords in C are distance 2 apart.

For C and E this argument does not apply, because all codewords in both codes are distance 2 apart. But despite this C and E are not equivalent: you are asked to prove this in Question 1(b) on Sheet 4.

As an example of how to use Lemma 6.3 we shall solve Question 6 on Sheet 1.

Lemma 6.5. $A_2(5,3) = 4$.

The proof of Lemma 6.5 actually shows something stronger: up to equivalence there is a unique binary $(5,4,3)$-code, namely the code $\{00000, 11100, 00111, 11011\}$ first seen in Question 3 on Sheet 1.

Using similar ideas it is possible to find $A_2(8,5)$. The next lemma isolates one critical step. In it, we say that a binary word u has *weight* r, and write $\mathrm{wt}(u) = r$, if exactly r positions of u are equal to 1, and the rest are equal to 0.

Lemma 6.6. *Let u and w be binary codewords of length n. Suppose that $\mathrm{wt}(u) = r$ and $\mathrm{wt}(w) = s$. If $r + s \geq n$ then $d(u,w) \leq 2n - (r+s)$.*

In Question 2 of Sheet 4 you are asked to fill in the details of the proof of the following theorem.

Theorem 6.7. $A_2(8,5) = 4$.

7. Codes from Mutually Orthogonal Latin Squares

Definition 7.1. Let $q \in \mathbf{N}$ and let A be a q-ary alphabet. A *Latin square with entries from A* is a $q \times q$ array in which every row and column contains each symbol in A exactly once. We say that q is the *order* of the square.

Note that since there are q symbols and each row and column has length q, it is equivalent to require *either*

(i) each row and column contains every symbol in A; *or*
(ii) no symbol appears twice in any row or column of A.

For a large number of examples of Latin squares, see the Sudoku answers in the newspaper of your choice. A completed Sudoku grid is a Latin square of order 9 over the alphabet $\{1,2,\ldots,9\}$, satisfying the extra condition that each of its 3×3 subsquares contains each number exactly once.

Example 7.2. A Latin square of order 4 over the alphabet $\{0, 1, 2, 3\}$, constructed using the addition table for the integers modulo 4, is shown below.

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 0 \\ 2 & 3 & 0 & 1 \\ 3 & 0 & 1 & 2 \end{array}$$

Convention: It will be convenient to number the rows and columns of a Latin square over the alphabet $A = \{0, 1, \ldots, q-1\}$ by the numbers in A. So if X is the Latin square in Example 7.2 then $X_{00} = 0$, $X_{12} = 3$ and $X_{33} = 2$.

Definition 7.3. Let X and Y be Latin squares over an alphabet A. We say that X and Y are *orthogonal* if for each each $a, b \in A$ there exist unique $i, j \in A$ such that $X_{ij} = a$ and $Y_{ij} = b$.

Equivalently, X and Y are orthogonal if for all $a, b \in A$ there is a unique position in which X contains a and Y contains b. We shall abbreviate 'X and Y are a pair of mutually orthogonal Latin squares', as 'X and Y are MOLs'.

Example 7.4. Two MOLs over the alphabet $\{0, 1, 2, 3\}$ are shown below.

$$\begin{array}{cccc} 0 & 1 & 2 & 3 \\ 1 & 0 & 3 & 2 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{array} \qquad \begin{array}{cccc} 0 & 1 & 2 & 3 \\ 2 & 3 & 0 & 1 \\ 3 & 2 & 1 & 0 \\ 1 & 0 & 3 & 2 \end{array}$$

To show that these squares are orthogonal we form a new square whose entries are pairs of entries from the two squares,

$$\begin{array}{cccc} 00 & 11 & 22 & 33 \\ 12 & 03 & 30 & 21 \\ 23 & 32 & 01 & 10 \\ 31 & 20 & 13 & 02 \end{array}$$

and then check that each of the 16 pairs $00, 01, \ldots, 33$ appears exactly once.

Exercise: Show that there is no pair of MOLs of order 2.

Remark 7.5. In 1782 Euler posed the following problem: 36 officers belong to six regiments and hold six different ranks, so that each combination of rank and regiment corresponds to a unique officer. Can be the officers be paraded on a 6×6 parade ground so that in any line each regiment and rank occurs precisely once? Equivalently, does there exist a pair of MOLs of order 6? Euler conjectured that the answer was no, but this was not proved until 1900.

In fact there are pairs of MOLs of all orders other than 2 and 6. The existence of MOLs of orders $10, 14, 18, \ldots$ is quite tricky, and was only proved in 1960. Here we will only prove existence for odd prime orders. Note that there are other pairs of MOLs of odd prime order that do not come from this construction.

Lemma 7.6. *Let $q \geq 3$ be prime and let $A = \{0, 1, \ldots, q-1\}$. For $i, j \in A$ let*

$$X_{ij} = i + j \bmod q$$
$$Y_{ij} = 2i + j \bmod q$$

Then X and Y are mutually orthogonal Latin squares.

We now show how to use MOLs to construct a family of 1-error correcting codes. These codes all have length 4 and minimum distance 3.

Theorem 7.7. *Let A be the alphabet $\{0, 1, \ldots, q-1\}$. There is a pair of MOLs over A of order q $\iff$ there is a $(4, q^2, 3)$-code over A.*

In lectures we will prove the '$\Longrightarrow$' direction. See Question 1 on Sheet 5 for the '$\Longleftarrow$' direction.

Example 7.8. Let X and Y be the MOLs in Example 7.4. The corresponding code has a codeword (i, j, X_{ij}, Y_{ij}) for every i, j such that $0 \leq i, j \leq q-1$. So the codewords are

0000	0111	0222	0333	1012	1103	1230	1321
2023	2132	2201	2310	3031	3120	3213	3302

Conversely given this list of codewords we can reconstruct X and Y.

The questions on Sheet 5 will help you to practice these constructions.

8. The Singleton bound and puncturing a code

In this section we shall prove another bound on the maximum size of a code of length n and minimum distance d over a q-ary alphabet. This bound is often stronger than Hamming's bound when q is large.

Definition 8.1. Let C be a code of length $n \geq 2$ and minimum distance ≥ 2. Let $C^\star$ be the code whose codewords are obtained by removing the final position from each codeword in C. We say that $C^\star$ is obtained by *puncturing C in its final position*.

Note that since C has minimum distance ≥ 2, it is impossible for two codewords in C to become equal when their final position is removed. So $C^\star$ has the same size as C.

Example 8.2. Let C be the binary code whose codewords are all binary words of length 4 with an even number of 1s. Let $C^\star$ be the code obtained by puncturing C in its final position. Then

$$C = \{0000, 1100, 1010, 0110, 1001, 0101, 0011, 1111\}$$
$$C^\star = \{000, 110, 101, 011, 100, 010, 001, 111\}$$

Thus C has minimum distance 2 and $C^\star$ has minimum distance 1.

Lemma 8.3. *Let C be a code of length n and minimum distance d. The punctured code $C^\star$ has length $n-1$ and minimum distance $\geq d-1$.*

Theorem 8.4 (Singleton Bound)**.** *If C is a q-ary code of length n and minimum distance d then $|C| \leq q^{n-d+1}$. Hence $A_q(n,d) \leq q^{n-d+1}$.*

Remarks 8.5. We make the following remarks on Theorem 8.4.

(1) If $n = 4$ and $d = 3$ then the Singleton bound gives $A_q(4,3) \leq q^{4-3+1} = q^2$. The codes constructed by MOLs achieve the bound. So there is a pair of MOLs of order q if and only if $A_q(4,3) = q^2$.

(2) The Reed–Solomon codes constructed in the MSc/MSci course achieve the Singleton bound. They show that $A_q(n,d) = q^{n-d+1}$ whenever q is a prime power and $q \geq n$.

(3) The special case of the Singleton bound when $d = n$ is

$$A_q(n, n) \leq q.$$

This was proved in Lemma 6.1(ii) by putting codewords into pigeonholes according to their first position. A similar argument can be used to prove the general Singleton bound: see Questions 3 and 6 on Sheet 5.

9. HADAMARD CODES AND THE PLOTKIN BOUND

Hadamard codes are a family of binary codes that have high minimum distance and so can detect and correct many errors. We shall see that, like the codes constructed from MOLs, Hadamard codes have the largest possible size for their length and minimum distance.

The Hadamard $(32, 64, 16)$-code used in the 1971 Mariner 9 mission to Mars was discussed in Example 1.14. Since the code has minimum distance 16, it follows from Theorem 4.5(ii) that it is 7-error correcting, as claimed (informally) in this example.

Hadamard codes are constructed using certain matrices with entries $+1$ and -1.

Definition 9.1. Let $n \in \mathbf{N}$. A *Hadamard matrix of order* n is an $n \times n$ matrix H such that each entry of H is either $+1$ or -1 and $HH^{tr} = nI$. Here I is the $n \times n$ identity matrix and H^{tr} is the transpose matrix of H.

Example 9.2. If $H = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ then H is a Hadamard matrix of order 2. Two Hadamard matrices of order 4 are shown below; in these matrices we write $+$ for 1 and $-$ for -1.

$$\begin{pmatrix} + & + & + & + \\ + & - & + & - \\ + & + & - & - \\ + & - & - & + \end{pmatrix}, \qquad \begin{pmatrix} + & + & + & - \\ + & + & - & + \\ + & - & + & + \\ - & + & + & + \end{pmatrix}.$$

Except for the 1×1 matrices $(+1)$ and (-1), all Hadamard matrices have even order. This result follows from the following lemma.

Lemma 9.3. *Suppose H is a Hadamard matrix of order n where $n \geq 2$. If i, $k \in \{1, 2, \ldots, n\}$ and $i \neq k$ then row i and row k of H are equal in exactly $n/2$ positions.*

The connection with coding theory is as follows.

Theorem 9.4. *Suppose that H is a Hadamard matrix of order $n \geq 2$. Let B be the $2n \times n$ matrix defined by*

$$B = \begin{pmatrix} H \\ -H \end{pmatrix}.$$

The rows of B are the codewords in a $(n, 2n, n/2)$-code over the alphabet $\{+, -\}$.

We say that any code given by the construction in Theorem 9.4 is a *Hadamard code*. These codes can be converted into binary codes over the usual alphabet of bits $\{0, 1\}$ by replacing each $+$ with 0 and each $-$ with 1.

Example 9.5. Let

$$H = \begin{pmatrix} + & + & + & - \\ + & + & - & + \\ + & - & + & + \\ - & + & + & + \end{pmatrix}.$$

The construction in Theorem 9.4 gives the binary code with codewords

$$\begin{array}{cccc} 0001 & 0010 & 0100 & 1000 \\ 1110 & 1101 & 1011 & 0111. \end{array}$$

The Singleton bound is often the strongest bound for codes over a large alphabet, but for a binary $(2d, M, d)$-code it only gives the bound $M \leq 2^{d+1}$. The following result leads to a stronger bound on $A_2(2d, d)$.

Theorem 9.6 (Plotkin bound)**.** *Let n, $d \in \mathbf{N}$ be such that $2d > n$. Then*

$$A_2(n, d) \leq \frac{2d}{2d - n}.$$

The proof of this bound is non-examinable: see the optional questions on Sheet 6 for an outline proof. For example, taking $n = 8$ and $d = 5$ we get

$$A_2(8, 5) \leq 10/(10 - 8) = 5.$$

By Theorem 6.7, $A_2(8, 5) = 4$ so the Plotkin bound comes close to the strongest possible result. In other cases the Plotkin bound is sharp.

Exercise: Use the Plotkin bound to prove that $A_2(9, 6) = 4$.

A related bound is attained by Hadamard codes.

Corollary 9.7 (Another Plotkin bound). *If $d \in \mathbf{N}$ then*

$$A_2(2d, d) \leq 4d.$$

If there is a Hadamard matrix of order $2d$ then

$$A_2(2d, d) = 4d.$$

It is quite easy to show that if there is a Hadamard matrix of order n then either $n = 1$, or $n = 2$ or n is divisible by 4. The construction in Question 2 of Sheet 6 shows that if there is a Hadamard matrix of order n then there is a Hadamard matrix of order $2^a m$ for all $a \in \mathbf{N}_0$. It is a major open problem to show that there are Hadamard matrices of all orders divisible by 4.

There is also a related 'asymptotic' Plotkin bound, which states that $A_2(n, d) \leq 2^{n-2d+1} n$ for all n and d. (See optional questions on Sheet 6. The asymptotic Plotkin bound is non-examinable.)

10. Gilbert–Varshamov Bound

Recall that, stated using the $A_2(n, d)$ notation, Hamming Packing Bound (Theorem 5.4) becomes

$$A_2(n, d) \leq \frac{2^n}{\sum_{k=0}^{e} \binom{n}{k}}$$

where $e = \lfloor (d-1)/2 \rfloor$. In the proof of this bound, we argued that if C is a binary (n, M, d)-code then the Hamming balls of radius e about codewords in C are disjoint.

A related argument using Hamming balls of radius $d-1$ gives a lower bound on $A_2(n, d)$. The idea is to construct a code of minimum distance d in the most naïve way possible: we put in new codewords until the Hamming balls of radius $(d-1)$ about codewords cover $\{0,1\}^n$, and so every word is distance $\leq (d-1)$ from some codeword.

Theorem 10.1 (Gilbert–Varshamov bound). *If $n, d \in \mathbf{N}$ then*

$$A_2(n, d) \geq \frac{2^n}{\sum_{k=0}^{d-1} \binom{n}{k}}.$$

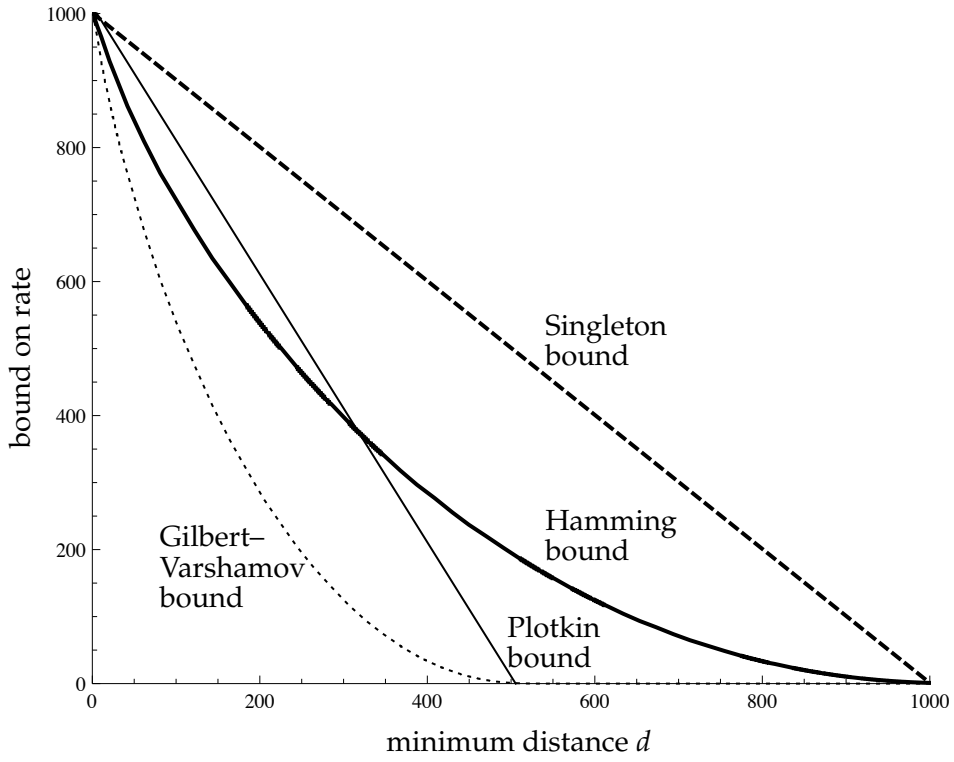
Summary of Part B. The Main Coding Theory Problem asks for codes over a given q-ary alphabet with small length n, high size M and high minimum distance d. To study these conflicting requirements, we defined $A_q(n, d)$ to be the largest size M of a q-ary code of length n and minimum distance d.

We have seen the Hamming, Plotkin and Singleton upper bounds on $A_q(n, d)$. In some cases these bounds are achieved by certain 'best possible' codes: by Remark 8.5(1) the MOLs codes in §7 achieve the Singleton bound, and by Corollary 9.7 the Hadamard codes in §9 achieve the Plotkin bound. In these cases the Main Coding Theory Problem is completely solved.

Taking $\log_2$ of a bound gives a bound on the rate of a code, as defined in Definition 1.10. Doing this makes it easier to compare different bounds. The graph below compares all the bounds seen so far for binary codes of length 1000. (The MATHEMATICA notebook used to draw this graph is available from Moodle.) The Plotkin bound used is the 'asymptotic bound' mentioned after Corollary 9.7.

The Plotkin bound is stronger than the Hamming Packing Bound for $d \geq 320$. For most d there is a wide gap between the Gilbert–Varshamov lower bound and the minimum of the Hamming and Plotkin upper bounds, and all we know is that $A_2(1000, d)$ is somewhere in between. Determining the true value of $A_2(n, d)$ for large n and d is one of the main open problems in coding theory.

Comparison of bounds for binary codes of length 1000



Part C: Linear Codes

11. Linear codes and weights

In the final part of the course we shall look at linear codes. We shall develop the theory for binary codes only; this shows all the main ideas. The extension to larger alphabets of prime or prime power degree is not difficult, and may be found in any of the recommended textbooks.

From now on the alphabet of of bits $\{0, 1\}$ should be thought of as $\mathbf{Z}_2$, that is, the integers modulo 2. So we have

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0.$$

Binary words of length n are elements of $\mathbf{Z}_2^n$. Given $u = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots, v_n) \in \mathbf{Z}_2^n$, we define

$$(u_1, u_2, \ldots, u_n) + (v_1, v_2, \ldots, v_n) = (u_1 + v_1, u_2 + v_2, \ldots, u_n + v_n).$$

Definition 11.1. Let C be a binary code of length n. We say that C is *linear* if for all $u, w \in C$ we have $u + w \in C$.

If C is a linear binary code and $u \in C$ then $u + u = (0, 0, \ldots, 0)$. So it follows from Definitions 1.4 and 11.1 that any binary code contains the all-zeros word $(0, 0, \ldots, 0)$. We will write this word as $\mathbf{0}$ (or $\underline{0}$ on the board).

We have already seen many examples of linear codes.

Example 11.2.

(1) The length 5 code $\{00000, 11100, 00111, 11011\}$ is linear.
(2) For any $n \in \mathbf{N}$, the binary repetition code of length n is a linear $(n, 2, n)$-code.
(3) For any $n \in \mathbf{N}$, the code of size 2^n consisting of all binary words of length n is a linear $(n, 2^n, 1)$-code.
(4) Let C be all binary words of length 4. As in Example 2.9, let C_{ext} be the code obtained by adding an extra bit at the end of each codeword to make the total number of 1s in each codeword even. Then, C_{ext} is a $(5, 16, 2)$-code and

$$C_{\text{ext}} = \{(u_1, u_2, u_3, u_4, u_5) \in \mathbf{Z}_2^5 : u_1 + u_2 + u_3 + u_4 + u_5 = 0\}.$$

We will show that C_{ext} is linear.

The codes in Example 11.2 have sizes 4, 2, 2^n and 16 respectively. This is explained by Theorem 12.5, which implies that any linear binary code has size a power of 2.

It is curious that many codes that meet the bounds proved in Part B are linear, or if not linear, at least equivalent to linear codes. For example, we saw in Lemma 6.5 that $A_2(5,3) = 4$, and that any binary $(5,4,3)$-code is equivalent to

$$\{00000, 11100, 00111, 11011\}.$$

By Example 11.2(1) this code is linear.

The next lemma shows that Hamming distance behaves well under addition.

Lemma 11.3. *Let u, w be binary words of length $n \in \mathbf{N}$. For any binary word $v \in \mathbf{Z}_2^n$ we have*

$$d(u,w) = d(u+v, w+v).$$

This lemma leads to an easy way to find the minimum distance of a linear code. Recall that the *weight* of a binary word u was defined just before Lemma 6.6 to be the number of positions of u equal to 1. For example, $\mathrm{wt}(11100) = 3$ and $\mathrm{wt}(11011) = 4$.

Lemma 11.4. *Let C be a linear binary code. The minimum distance of C is equal to the minimum weight of a non-zero codeword of C.*

Exercise: Use Lemma 11.4 to find the minimum distances of the codes in Example 11.2 and check that the results are as expected.

The last result in this section generalises the parity check extension codes seen in Example 2.9 and Example 11.2(2). For an optional related result see Questions 6 and 7 on Sheet 4.

Definition 11.5. Let C be a binary code of length n. The *parity check extension* of C is the code C_{ext} of length $n+1$ defined by

$$C_{\mathrm{ext}} = \{(u_1, \ldots, u_n, u_{n+1}) : (u_1, \ldots, u_n) \in C, u_1 + \cdots + u_n + u_{n+1} = 0.\}$$

Theorem 11.6. *Let C be a linear binary (n, M, d)-code. Then C_{ext} is a linear binary code of length $n+1$ and size M. The minimum distance of C_{ext} is d if d is even and $d+1$ if d is odd.*

12. Bases, generator matrices and encoding

In this section we will see an efficient way to encode using a linear binary code. The next exercise shows the basic idea.

Exercise: Suppose that $u(1), \ldots, u(k)$ are codewords in a linear binary code C. Show that if $c_1, c_2, \ldots, c_k \in \mathbf{Z}_2$, then $c_1u(1) + \cdots + c_ku(k)$ is a codeword in C and

$$c_1u(1) + \cdots + c_ku(k) = (c_1, \ldots, c_k) \begin{pmatrix} u(1) \\ \vdots \\ u(k) \end{pmatrix}.$$

This suggests an encoding strategy where we first convert messages to binary words of length k, and then encode the binary word $(c_1, \ldots, c_k)$ as the codeword $c_1u(1) + \cdots + c_ku(k) \in C$.

Example 12.1. (For interest only.) The binary $(32, 64, 16)$-Hadamard code used by Mariner 9 is linear. The encoder used the idea just outlined, with the matrix G shown below.

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

To encode a pixel whose blackness is represented by the number $m \in \{0, 1, \ldots, 63\}$, write m in binary as $b_5b_4b_3b_2b_1b_0$ and then encode it as

$$(b_5, \ldots, b_1, b_0)G.$$

For example, since 34 is 100010 in binary, 34 is encoded as the codeword $(1, 0, 0, 0, 1, 0)G = 01010101010101011010101010101010$.

To make this encoding scheme work, **it is essential** to make a careful choice of $u(1), \ldots, u(k)$. For instance, suppose there are $c_1, \ldots, c_k \in \mathbf{Z}_2$, not all equal to 0, such that

$$c_1u(1) + \cdots + c_ku(k) = \mathbf{0}$$

where $\mathbf{0}$ is the all-zeros word of length n. Then we will encode both $(c_1, \ldots, c_k)$ and $(0, \ldots, 0)$ as $\mathbf{0}$. This is clearly undesirable!

To understand how to avoid this problem, we need to think of binary words as vectors and linear binary codes as subspaces of $\mathbf{Z}_2^n$. Then the condition we need is that $u(1), \ldots, u(k)$ form a basis for C. The definition below reminds you of this and related terms.

Definition 12.2. Let C be a linear binary code of length n. We say that words $u(1), \ldots, u(k) \in \mathbf{Z}_2^n$ are

(a) *linearly independent* if the only solution to the equation

$$c_1 u(1) + \cdots + c_k u(k) = 0$$

with $c_1, \ldots, c_k \in \mathbf{Z}_2$ is $c_1 = c_2 = \ldots = c_k = 0$.

(b) *span* C if for every $w \in C$ there exist $c_1, \ldots, c_k \in \mathbf{Z}_2$ such that

$$w = c_1 u(1) + \cdots + c_k u(k).$$

(c) *a basis of* C if they are linearly independent and span C.

Example 12.3.

(1) Let $C = \{00000, 11100, 00111, 11011\}$, as in Example 11.2(1) Then a basis for C is $11100, 00111$. If we take

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

then the codewords in C are $(0,0)G$, $(0,1)G$, $(1,0)G$ and $(1,1)G$.

(2) Let C_{ext} be the parity check extension of all binary words of length 4, considered in Example 11.2(4). Then

$$10001, 01001, 00101, 00011$$

is a basis for C_{ext}. If we take

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

then we would encode the message (c_1, c_2, c_3, c_4) by

$$(c_1, c_2, c_3, c_4)G = (c_1, c_2, c_3, c_4, c_1 + c_2 + c_3 + c_4) \in C_{\text{ext}}.$$

It is **very important** to note that a linear binary code usually does not have a unique basis.

For example, another basis for the code $\{00000, 11100, 00111, 11011\}$ is 11100, 11011. So if C is a linear binary code, then it is correct to write 'a basis of C' rather than 'the basis of C'. We will see a systematic way to find a basis from a set of codewords spanning a code in Example 12.7.

Exercise: Find a different basis for the code C_{ext} in Example 12.3(2).

Definition 12.4. Suppose that C is a linear binary code of length n and minimum distance d. If $u(1), \ldots, u(k)$ is a basis of C then we say that C has *dimension* k and that C is a $[n, k, d]$-code.

Thus a linear binary $(n, 2^k, d)$-code is a $[n, k, d]$-code. The codes in Example 12.3 have parameters $[5, 2, 3]$ and $[5, 4, 2]$, respectively.

The next result connects dimension with the rate of a binary code, as defined in Definition 1.10.

Theorem 12.5. *Let C be a linear binary code having $u(1), \ldots, u(k)$ as a basis. For each $w \in C$ there exist unique $c_1, \ldots, c_k \in \mathbf{Z}_2$ such that*

$$w = c_1 u(1) + \cdots + c_k u(k).$$

Hence $|C| = 2^k$ and the rate of C is k/n.

In particular, it follows from Theorem 12.5 that any linear binary code has size 2^k for some $k \in \mathbf{N}$. Moreover, any two bases of a linear binary code C of length n and size 2^k have the same number of elements[13], namely k.

Exercise: Find a basis for the square code. (See Question 1 on Sheet 7 for a hint.)

The matrices used in Example 12.3 are generator matrices, as defined in the following definition.

Definition 12.6. Suppose that C is a linear binary code of length n having $u(1), \ldots, u(k) \in \mathbf{Z}_2^n$ as a basis. The $k \times n$ matrix

$$\begin{pmatrix} u(1) \\ \vdots \\ u(k) \end{pmatrix}$$

is said to be a *generator matrix* for C.

[13] It is a standard result from linear algebra that any two bases of a vector space have the same number of elements. The quick proof indicated here depends on the size of C being finite, so only works over finite fields, such as $\mathbf{Z}_2$.

Example 12.7. Let C be the linear code of length 7 spanned by the codewords 1100110, 1011010, 0110011, 0001111. These codewords are not linearly independent. We can demonstrate this, and find a basis and generator matrix for C, by applying row operations to the matrix

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

Suppose we want to encode a number between 0 and $2^k - 1$ using a linear binary code of dimension k with generator matrix G. To do this, write the number in binary, say as $b_{k-1} \dots b_1 b_0$, and then, as in Examples 12.1 and 12.3, encode the resulting binary word of length k as the codeword

$$(b_{k-1}, \dots, b_1, b_0)G.$$

Example 12.8. Let C be the linear code in Example 12.7. We saw that C has generator matrix

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

so C has dimension 3 and size 2^3. To encode the number 7 we write 7 in binary as 111 and take the codeword

$$(1,1,1)\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} = (1,1,0,1,0,0,1)$$

In general, the number $4b_2 + 2b_1 + b_0$, written in binary as $b_2 b_1 b_0$, is encoded as

$$(b_2, b_1, b_2 + b_1, b_0, b_2 + b_0, b_1 + b_0, b_0 + b_1 + b_2).$$

Note that if no errors occur when a codeword is transmitted through the channel, then the message can be read off from bits 1, 2 and 4 of the received word.

We end by defining a class of generator matrices that are convenient for use when encoding, and also of theoretical importance.

Definition 12.9. A generator matrix

$$\begin{pmatrix} I_k & A \end{pmatrix}$$

where I_k is the $k \times k$ identity matrix and A is a $k \times (n-k)$ matrix is said to be in *standard form*.

The generator matrix in Example 12.3(2) is in standard form. If we swap positions 3 and 4 in all the codewords in the code C in Example 12.8 we get an equivalent code C' with generator matrix

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

in standard form. This is a special case of the following theorem.

Theorem 12.10. *Let C be a linear binary code of length n and dimension k. Then C is equivalent, by a permutation of the positions in the codewords, to a code with a generator matrix in standard form*

$$\begin{pmatrix} I_k & A \end{pmatrix}$$

where A is an $k \times (n-k)$-matrix.

If $G = \begin{pmatrix} I_k & A \end{pmatrix}$ is a generator matrix for a code C in standard form then the binary word $(c_1, c_2, \ldots, c_k)$ is encoded as

$$(c_1, c_2, \ldots, c_k)G = (c_1, c_2, \ldots, c_k, a_1, \ldots, a_{n-k}) \in C$$

for some $a_1, \ldots, a_{n-k} \in \mathbf{Z}_2$. This is convenient because if no errors occur in transmission, then the message can be easily read off from the first k positions in the received word.

13. Decoding by standard arrays

In this section we shall see a way to implement nearest neighbour decoding for linear codes that exploits their special structure.

Definition 13.1. Let C be a linear binary code of length n. A *coset* of C is a set of the form

$$C + v = \{u + v : u \in C\}$$

where $v \in \mathbf{Z}_2^n$.

Note that if $v \in \mathbf{Z}_2^n$ then, since $\mathbf{0} \in C$, we have $v = \mathbf{0} + v$ and so $v \in C + v$. Hence the coset containing v is $C + v$.

Example 13.2. Let C be the linear binary code

$$C = \{0000, 1110, 0011, 1101\}$$

obtained by puncturing (see Definition 8.1) the code in Example 12.3(1) in its final position. If we send the codewords through a channel that corrupts position 1 every time, then the received words are

$$C + 1000 = \{1000, 0110, 1011, 0101\}.$$

The other possible one bit errors give cosets

$$C + 0100 = \{0100, 1010, 0111, 1001\},$$
$$C + 0010 = \{0010, 1100, 0001, 1111\},$$
$$C + 0001 = \{0001, 1111, 0010, 1100\}.$$

We also have the coset $C + 0000 = C$.

Note that the cosets $C + 0010$ and $C + 0001$ are equal. Each word v in this coset is distance 1 from two codewords, namely $v + 0010$ and $v + 0001$, so nearest neighbour decoding fails whenever such a word is received. For example, if we receive 1111, the transmitted word could be either 1101 or 1110.

Exercise: Taking C as in Example 13.2, show that

$$C + 1001 = C + 0100 = \{0100, 1010, 0111, 1001\}.$$

Show that if v is a word in this coset then using nearest neighbour decoding, v is decoded as $v + 0100 \in C$.

It is **very important** to bear in mind that cosets are sets, and that the same coset can be written as $C + v$ for many different words v.

Exercise: Let C be a linear binary code of length n. Show that if $v \in \mathbf{Z}_2^n$ then $C + v = C + (u + v)$ for all $u \in C$.

Lemma 13.3. *Let C be a linear binary code of length n. If $C + v$ and $C + v'$ are cosets of C then either $C + v = C + v'$ or the cosets $C + v$ and $C + v'$ are disjoint.*

Exercise: Check that each binary word of length 4 is in a unique coset of the code in Example 13.2.

Definition 13.4. Let C be a linear binary code of length n. A *standard array* for C is a table in which each row consists of the codewords in a coset of C, arranged so that

(i) the first row is C;

(ii) if the word x appears in the first column then $\mathrm{wt}(x) \leq \mathrm{wt}(v)$ for all v in the row of x.

The first word in each row is said to be a *coset leader*. To decode a received word $v \in \mathbf{Z}_2^n$ by *standard array decoding*, decode v as $v + x$ where x is the coset leader for the row containing v.

Example 13.5. A standard array for the code C in Example 13.2 is

$$\begin{array}{cccc} 0000 & 1110 & 0011 & 1101 \\ 1000 & 0110 & 1011 & 0101 \\ 0100 & 1010 & 0111 & 1001 \\ 0010 & 1100 & 0001 & 1111 \end{array}$$

Note that we could also taken the fourth row to be

$$0001 \quad 1111 \quad 0010 \quad 1100$$

with 0001 as the coset leader, since both 0010 and 0001 have weight 1. The other coset leaders 0000, 1000 and 0100 are uniquely determined by their cosets.

Exercise: Decode the received words 0011, 0111 and 1111 using standard array decoding with the standard array in Example 13.5.

There are 2^n binary words of length n, and each row in a standard array for a linear binary code of dimension k has 2^k words in it. So a standard array for a linear binary $[n, k, d]$-code should have 2^{n-k} rows. If you check this holds, you will avoid two common errors: putting in too many rows, or too few.

The next theorem shows that standard array decoding always gives one of the closest codewords to a received word.

Theorem 13.6. *Let C be a linear binary code of length n. Let $v \in \mathbf{Z}_2^n$. Suppose that the row containing v has coset leader x. Then $v + x \in C$ and*

$$d(v + x, v) \leq d(u, v)$$

for all $u \in C$.

In the proof we used that $d(v + x, v) = \mathrm{wt}(x)$ and $d(u, v) = \mathrm{wt}(u + v)$. So $v + x$ is the *unique* nearest codeword to v if and only if x is the unique word of minimum weight in the coset $C + v$.

This gives a way to use a standard array to perform nearest neighbour decoding. Suppose $v \in \mathbf{Z}_2^n$ is received and that x is the chosen coset leader in the row of the standard array containing v. If x is the unique word of minimum weight in its row then x is decoded to $x + v$ using nearest neighbour decoding. If there are other words of the same weight as x in the row of x then nearest neighbour decoding fails.

In the latter case, we can either use standard array decoding and decode v as $v + x$, or request retransmission. Any decoding strategy in which some received words are decoded, but for others retransmission is requested, is called *incomplete decoding*.

14. Parity check matrices and syndrome decoding

All the linear codes we have seen so far can be defined by linear equations. For instance, the code C_{ext} consisting of all binary words of length 5 with evenly many 1s can be defined by

$$C_{ext} = \{(u_1, u_2, u_3, u_4, u_5) \in \mathbf{Z}_2^5 : u_1 + u_2 + u_3 + u_4 + u_5 = 0\}$$

and the square code can be defined by

$$S = \left\{ (u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8) \in \mathbf{Z}_2^8 : \begin{array}{l} u_1 + u_2 = u_5,\ u_3 + u_4 = u_6 \\ u_1 + u_3 = u_7,\ u_2 + u_4 = u_8 \end{array} \right\}$$

To perform the decoding algorithm for the square code seen earlier in the course, we record which linear equations are not satisfied, and then try to flip a single bit to make all of them hold.[14]

Exercise: For each of the following received words, decide which of the four defining equations for the square code fail to hold. Decode each word using nearest neighbour decoding.

(i) 10001100 (ii) 11001011 (iii) 11000000 (iv) 10000001

Observe that C_{ext} has length 5, dimension 4 and is defined by one equation, and S has length 8, dimension 4 and is defined by four equations. In Theorem 14.3 we will prove that any linear binary code of length n and dimension k can be defined by $n - k$ linear equations.

Definition 14.1. Let C be a linear binary code of length n and dimension k. A *parity check matrix* for C is an $(n-k) \times n$ matrix H with linearly independent rows such that for each $u \in \mathbf{Z}_2^n$ we have

$$u \in C \iff uH^{tr} = \mathbf{0}.$$

Here $\mathbf{0}$ is the all-zeros word of length $n - k$.

Example 14.2.

(1) The code C_{ext} defined above has parity check matrix

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

[14] If two or more errors occur then we might not decode to the sent word, or it might not be possible to satisfy all the equations by flipping a single bit. Since the square code has minimum distance 3 it is only 1-error correcting, and so we do not expect to be able to decode reliably if two or more errors occur.

(2) Let S be the square code. Then S has as a parity check matrix

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

In fact the condition in Definition 14.1 that H has linearly independent rows is logically unnecessary: it is implied by the other conditions in the definition. So it is no accident that just writing down the right number of equations gave us a parity check matrix in both examples above.

The next theorem gives a more systematic way to find a parity check matrix.

Theorem 14.3. *Let C be a linear binary code of length n and dimension k. Then C has a parity check matrix. Moreover, if C has a generator matrix G in standard form $G = \begin{pmatrix} I_k & A \end{pmatrix}$ then*

$$\begin{pmatrix} A^{tr} & I_{n-k} \end{pmatrix}$$

is a parity check matrix for C.

For example, if $G = \begin{pmatrix} I_k & A \end{pmatrix}$ is the standard form generator matrix for the square code, then applying Theorem 14.3 to G, we get the parity check matrix already found in Example 14.2(2).

Definition 14.4. Let C be a linear binary code of length n and dimension k and let H be a parity check matrix for C. The *dual code* $C^{\perp}$ is the linear binary code of length n and dimension $n - k$ with generator matrix H.

Note that, by Definition 14.1, H has linearly independent rows, so we can use H as a generator matrix.

We will assume that the dual code is well-defined, i.e. that it does not depend on the choice of parity check matrix H. For a proof of this, and some further (non-examinable) results on parity check matrices, see the optional questions on Sheet 9.

Example 14.5. Let C_{ext} be as in Example 14.2(1). Then

$$C_{\text{ext}}^{\perp} = \{00000, 11111\}$$

is the binary repetition code of length 5, and

$$\{00000, 11111\}^{\perp} = C_{\text{ext}}.$$

We can also start with an $(n-k) \times n$ matrix H with linearly independent rows, and use it to define a code C having H as its parity check matrix. In the following extended example we use this idea to construct the $[7,4,3]$-Hamming code.

Example 14.6. Let

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

and let $C = \{u \in \mathbf{Z}_2^7 : uH^{tr} = 0\}$. Then C is a linear binary code with parity check matrix H and generator matrix

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

By Lemma 11.4, the minimum distance of C is equal to the minimum weight of a non-zero codeword. Clearly there are codewords of weight 3 in C, so to show C has minimum distance 3, it suffices to show there are no codewords of weight 1 or 2. This can be done using H.

The method used in Example 14.6 to find the minimum distance of C has an important generalization. This theorem is non-examinable, and will be skipped if time is pressing.

Theorem 14.7. *Let C be a linear binary code of length n and dimension k. Let H be a parity check matrix for C. The minimum distance of C is equal to the minimum $r \in \mathbf{N}$ such that there exist r linearly dependent columns of H.*

In standard array decoding one has to hunt through the entire standard array to find the coset of the code in which a received word lies. We end with an improved method that uses parity check matrices.[15]

Theorem 14.8. *Let C be a linear binary code of length n and dimension k with parity check matrix H and let $v, v' \in \mathbf{Z}_2^n$. Then v and v' are in the same coset of $C \iff vH^{tr} = v'H^{tr}$.*

This theorem motivates the following definition.

[15] Algebraically inclined readers will notice that since $C = \ker H^{tr}$, Theorem 14.8 follows from the first isomorphism theorem for the linear map $\mathbf{Z}_2^n \to \mathbf{Z}_2^{n-k}$ defined by $v \mapsto vH^{tr}$. Everyone else should ignore this remark.

Definition 14.9. Let C be a linear binary code of length n and dimension k with parity check matrix H. The *syndrome* of a word $v \in \mathbf{Z}_2^n$ is defined to be $vH^{tr} \in \mathbf{Z}_2^{n-k}$.

By Theorem 14.8 we can identify the coset of C containing a word $v \in \mathbf{Z}_2^n$ from its syndrome vH^{tr}. So to decode a received word v, calculate its syndrome vH^{tr}, and then decode v as $v + x$ where x is the chosen coset leader for the coset $C + v$ containing v.

Example 14.10. Let $C = \{0000, 1110, 0011, 1101\}$ be the code used in Examples 13.2 and 13.5. Then C has parity check matrix

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

By Theorem 14.8, any two words in the same coset of C have the same syndrome. The map from cosets of C to syndromes is

$$\begin{aligned} C &\mapsto (0,0,0,0)H^{tr} = (0,0) \\ C + (1,0,0,0) &\mapsto (1,0,0,0)H^{tr} = (1,0) \\ C + (0,1,0,0) &\mapsto (0,1,0,0)H^{tr} = (1,1) \\ C + (0,0,1,0) &\mapsto (0,0,1,0)H^{tr} = (0,1). \end{aligned}$$

Thus all words in $C + 1000 = \{1000, 0110, 1011, 0101\}$ have syndrome $(1,0)$, and if any of the words 1000, 0110, 1011, 0101 is received, it will be decoded by adding 1000, since this is the unique coset leader in the coset $C + 1000$.

Using syndrome decoding we can replace the standard array in Example 13.5 with the more concise table below.

syndrome	chosen coset leader
00	0000
10	1000
01	0010
11	0100

Syndrome decoding is ideally suited to the Hamming $[7,4,3]$-code seen in Example 14.6.

Example 14.11. Let C, G and H be as in Example 14.6. Let $e(i)$ be the word with a 1 in position i and 0 in all other positions. The syndrome of $e(i)$ is $e(i)H^{tr}$, which is the ith row of H^{tr}.

The columns of H are distinct and non-zero, so by Lemma 13.3 and Theorem 14.8 we have

$$\mathbf{Z}_2^7 = C \cup \big(C + e(1)\big) \cup \cdots \cup \big(C + e(7)\big)$$

where the union is disjoint.

To decode a received word v, we calculate its syndrome vH^{tr}. If vH^{tr} is the ith row of H^{tr} then $vH = e(i)H^{tr}$ and, by Theorem 14.8, $v \in C + e(i)$. So we decode v as $v + e(i)$.

For example, to use C to send the number 13, we would write 13 as 1101 in binary, and encode it as

$$(1,1,0,1)G = (1,0,1,0,1,0,1).$$

Suppose that when we transmit 1010101, an error occurs in position 6, so 1010111 is received. Then the syndrome of the received word is

$$(1,0,1,0,1,1,1)H^{tr} = (0,1,1)$$

which is row 6 of H^{tr}. So we decode by flipping the bit in position 6 to get 1010101. We then read off 1101 from positions 3, 5, 6 and 7.

More generally, let C be a linear binary code of length n and dimension k. To use syndrome decoding on C we need to choose a coset leader for each coset. This can be done by constructing a standard array. But it is more efficient to take a parity check matrix H for C and compute vH^{tr} for words of low weight until all elements of $\mathbf{Z}_2^{n-k}$ have appeared. Then, by Theorem 14.8, we have a coset leader for every coset.

Exercise: Make a table of syndromes and chosen coset leaders for the square code using the parity check matrix given in Example 14.2(2).

A suitable table for incomplete decoding, where we request retransmission if it appears that two or more errors have occurred will have 9 rows: one corresponding to the code C, and one for each of 8 possible one bit errors. Since $|C| = 16$ and $|\mathbf{Z}_2^8| = 2^8 = 256$, there are 16 distinct cosets of C and so the full table has 16 rows, one for each possible syndrome in $\mathbf{Z}_2^4$.

Summary of Part C. In this section we looked at codes satisfying the linearity property that if u, w are codewords then $u + w$ is also a codeword. In Lemma 11.4 we saw that the minimum distance of a linear binary code is the minimum weight of a non-zero codeword.

In §12 we saw that a linear binary code of length n and dimension k has a $k \times n$ generator matrix. This gives a concise way to specify the code: rather than having to write down 2^k codewords we can just specify k basis elements. We also saw how to use a generator matrix to encode.

In §13 we saw standard array decoding, and in §14 we saw a more efficient way to implement standard array decoding using syndromes and parity check matrices. In Example 14.6 we defined the Hamming $[7, 4, 3]$ code and saw how to use syndrome decoding to correct a single error in a sent word.

Hamming's construction generalises (see the optional questions on Sheet 9) to gives a linear binary $[2^r - 1, 2^r - r - 1, 3]$-code for any $r \in \mathbf{N}$. These codes achieve Hamming's packing bound, and so the ideas in Part C give a complete solution to the Main Coding Theory Problem for 1-error correcting codes. The problem of finding linear binary codes of large size that can correct more errors has motivated much of the subsequent work in this subject.