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 (ABLT3), Tuesday 3pm (ABLT1), Thursday 10am (BLT2).

Extra lecture for MT461/MT5461: Thursday noon (ABLT2).

Office hours in McCrea 240: Tuesday 11am, Thursday 3pm, Friday 3pm.

Date: Second term 2011/12.

Preliminaries

Error correcting codes were invented in the late 1940s and have since become an essential part of the modern electronic world. Compact discs, spacecraft 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. (Mainly 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 `http://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 linear algebra: vector spaces, subspaces, matrices, image and kernel, rank-nullity theorem, reduced row-echelon form. I will issue a handout later in term to remind you of these ideas.

Problem sheets and exercises. There will be eight marked problem sheets 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. I will go through some 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.

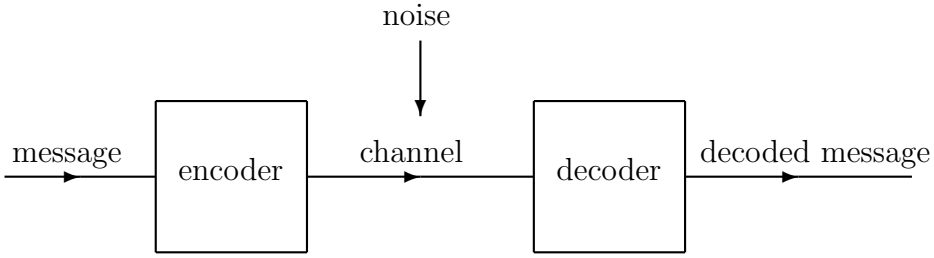
1. Introduction

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 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.

We will now see how this example is described using the technical language of coding theory.

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 we may write the word $(1, 1, 1)$ of length 3, corresponding to 'Yes' in Scheme 2, as 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 $|C|$.

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 they correspond to how computers store and send data.[2]

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

$$\text{'No'} \xrightarrow{\text{encoded as}} 00$$

$$\text{'Yes'} \xrightarrow{\text{encoded as}} 11.$$

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

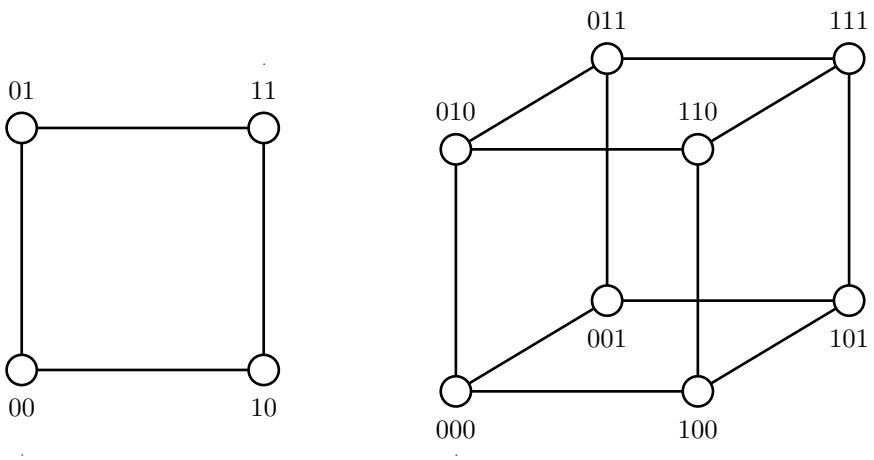
$$D = \{000, 111\}$$

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 Part A 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 $\{0, 1\}^3$ on the next page.

[2]Some early computers represented internal data by words over ternary (3-ary) or decimal (10-ary) alphabets, but they are now mere historical curiosities.



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.

Remarks 1.6. The following remarks on Definition 1.4 should be noted.

(1) By Definition 1.4, all the codewords in a code have the same length.

(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.[3]

[3]Of course there is nothing to stop Alice encrypting her message to Bob before it is encoded for the channel. In addition, 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.

(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 1.7.

Example 1.7. Suppose Alice wants to send Bob one of the messages 'Launch nukes' or 'Stand-down'. They decide to use the binary code $D = \{000, 111\}$ from Example 1.2, with the encoder

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

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

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, but with 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 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; this corresponds to noise in the channel. Repeat the game in the previous exercise: try not to use too many questions!

Example 1.8. We will convert some of the possible questioning strategies for Bob into binary codes of size 16.

We end with three examples of codes used in real life. These will not be covered fully in lectures, and you are not expected to remember any details. *Exercise:* read the examples and think about how each fits into our setup of encoder, channel and decoder.

Example 1.9. 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 purpose is to transmit information reliably through time, rather than through space.

The pits encode a long sequence of the bits 0 and 1. The encoding and decoding scheme used will always correct up to 16 errors in a block of 2048 consecutive bits. If, however, the errors occur in adjacent bits, as is usual for a scratch, then at least 128 consecutive errors may be corrected.

The code used in compact discs is a Reed–Solomon code: these codes will be treated in detail in the MSc/MSci course.

Example 1.10. The Australian railway company 'Victorian Railways' used a telegraph system and codebook. The entire codebook can be read online at `http://www.railpage.org.au/telecode/tc04.gif.html`. 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 $\{A, B, C, \ldots, X, Y, Z\}$. (This ignores 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.

(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 reader can detect and correct many errors. For example, even though the sentence

'Tae mext trxin tb Lxnton kas &e*n cznce!Hed'

has many errors, there is no difficulty in working 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, turning 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 1.11. 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 could send the two binary digits 0 and 1.

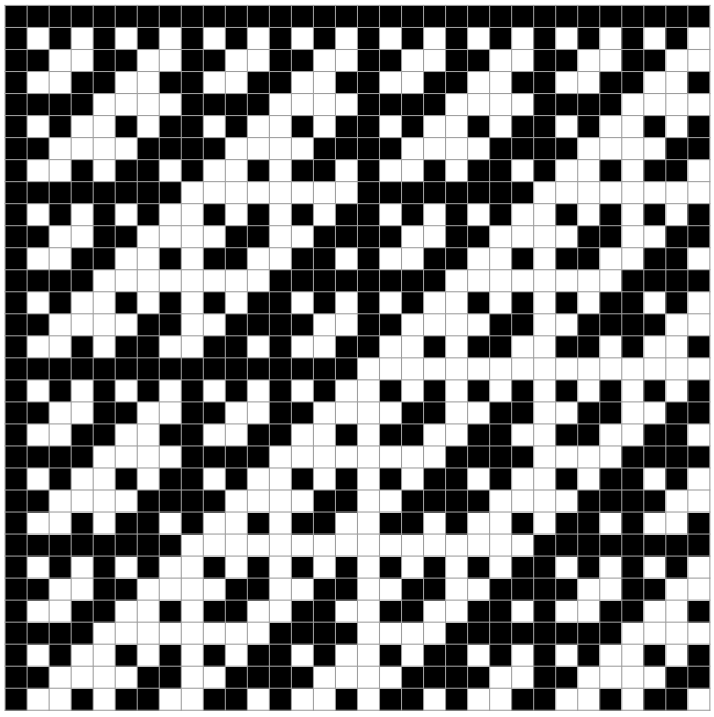
The naïve approach of just encoding each pixel as a string of 6 bits would not have worked well. One survey article[4] gives the probability as 0.05 that any particular bit will be flipped by the channel (so a 0 becomes a 1, and a 1 becomes a 0, as in the final part of Example 1.2). The probability that any particular pixel will be correctly transmitted is then $0.95^6 \approx 0.74$. So about 26% of the image will be wrong.

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. We will see 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.

The diagram overleaf 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 $11\ldots1$.

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



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. Finding fast and accurate decoders for large codes is a central problem in coding theory and has motivated much recent work in the subject.

Part A: Hamming distance and nearest neighbour decoding

2. Hamming distance and error detection/correction

In Definition 2.7 below we give a precise definition of what it means for a code to be able to detect, or correct, a given number of errors. For this we need the idea of Hamming distance between words, which makes precise our intuitive idea of when two words are close together.

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, $d(u, v) = \big|\{i \in \{1, 2, \ldots, n\} : u_i \neq v_i\}\big|$. 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 $\{\mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots, \mathrm{X}, \mathrm{Y}, \mathrm{Z}\}$ we have $d(\mathrm{TALE}, \mathrm{TAKE}) = 1$ and $d(\mathrm{TALE}, \mathrm{TILT}) = 2$.

Exercise: Check that $d(0011, 0101) = 2$. Find the number of binary words v of length 4 such that $d(0011, v) = r$ for each $r \in \{0, 1, 2, 3, 4\}$.[6]

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 Theorem 3.4 below.

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) $d(u, w) \leq d(u, v) + d(v, w)$.

Exercise: Find all English words v such that

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

Check that the triangle inequality holds when u, v, w are WARM, WALL and COLD, respectively. For the connection with Lewis Carroll's 'Doublets Game', see Question 9 on Sheet 1.

[6] You may recognise the sequence you get. If not, try typing it into Google.

To motivate Definition 2.7 we introduce two important families of codes.

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 Example 1.2, Alice and Bob used the binary repetition codes of lengths 2 and 3, with codewords $\{00, 11\}$ and $\{000, 111\}$, respectively. We saw that the length 2 repetition code could detect a single bit flip in the channel. The length 3 repetition code, with decoding done by the majority symbol in a received word, could correct a single bit flip.

Example 2.5. Consider the ternary repetition code of length 7 over the alphabet $\{0, 1, 2\}$ with codewords

$$0000000, \quad 1111111 \quad 2222222.$$

Any two distinct codewords are at distance 7. If up to 6 of the symbols in a codeword are corrupted in the channel, the received word will not be a codeword, and so the receiver will know an error has occurred. Decoding using the majority symbol gives the sent codeword when at most 3 errors occur (and may succeed when more errors occur).

Example 2.6 (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. This shows that if u, $u' \in C_{\text{ext}}$ are distinct codewords then $d(u, u') \geq 2$.

Exercise: How would you use the length 5 code C_{ext} to encode a number between 0 and 15?

Definition 2.7. 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

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

  for all codewords $u' \in C$ such that $u' \neq u$.

Remarks 2.8. We make the following remarks on Definition 2.7.

(1) Equivalently, a code C is t-error detecting if making up to t changes in a codeword does not give another codeword. It is t-error correcting if whenever v is a word within distance t of a codeword $u \in C$, then v is strictly nearer to u than to any other codeword of C. (So $d(u, v) < d(u', v)$ for all $u' \in C$ such that $u \neq u'$.)

You are welcome to state the definitions in this way, but note that we need Hamming distance to make clear what is meant by 'distance' and 'strictly nearer'.

(2) In both definitions we have $d(u, v) \leq t$, so v is obtained from u by changing **up to** t positions. Thus 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 8 on Sheet 1, there would be codes that are 2-error detecting but not 1-error detecting. The result would be theorems with long-winded hypotheses of the form 'Suppose C is a code that is t-error detecting for all $t \leq c, \ldots$'. This is undesirable.

(3) It may seem a bit odd to say that a code is 't-error correcting' when it is the decoder (be it human or computer) that has to do all the work of decoding! Moreover, we have seen in Example 1.7 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 a suitable decoder is used.*

We will now show that Definition 2.7 agrees with our findings in Examples 2.5 and 2.6.

Lemma 2.9. *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 3 of Sheet 1.

Lemma 2.10. *Let $n \in \mathbf{N}$ and let C_{ext} be the binary parity check code of length $n+1$ defined in Example 2.6. Then C_{ext} is 1-error detecting but not 2-error detecting. It is not 1-error correcting.*

In Example 1.8 we saw two different codes of size 16 that could be used to correct a single error. In the next lemma we show that the code of length 12 is indeed 1-error correcting, as this term is defined in Definition 2.7. The other code will be analysed on Sheet 2.

Lemma 2.11. *Let C be the binary code of length 12 seen in Example 1.8 whose codewords are all words of the form*

$$u_1u_2u_3u_4\, u_1u_2u_3u_4\, u_1u_2u_3u_4$$

where $u_1, u_2, u_3, u_4 \in \{0, 1\}$. Then C is 2-error detecting and one 1-error correcting. It is not 3-error detecting or 2-error correcting.

Note that, while the code C in Lemma 2.11 can be used to detect some cases where 3 or more errors occur, it **does not guarantee to do so**. The definitions of t-error detecting and t-error correcting codes follow Hamming's 'adversarial' approach, in which we aim to decode reliably even if the enemy, who wants to cause us the maximum inconvenience, is allowed to choose the positions in which the (up to t) errors occur.

Example 2.12 (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 $\{0, 1, 2, \ldots, 9, \mathrm{X}\}$.

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

We saw in the lemma that the ISBN-10 code cannot detect all double errors. Question 5 on Sheet 1 asks you to show that the word obtained by interchanging two adjacent positions in an ISBN is not an ISBN. Therefore the ISBN-10 code can detect *some* errors, of the type likely to be made by busy human beings.

3. Minimum distance

Question 4 on Sheet 1 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 3.1. 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 is always well-defined.

Exercise: Let C be a code. What do you think is the relationship between the maximum t for which C is t-error detecting / t-error correcting and the minimum distance of C?

Example 3.2. Here is an example from Hamming's original paper. 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 $d(000, 001) = 1$.

Lemma 3.3. *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 2.6 is 2.*
(iii) *The minimum distance of the square code is 3.*

To add to the results in the lemma above, Question 2 on Sheet 2 gives a step-by-step proof that the 1-error correcting code of length 9 seen in Example 1.8 has minimum distance 3.

Theorem 3.4. *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 $\iff$ $2t \leq d - 1$.* [**Misprinted as 't-error detecting' in original.**]

Corollary 3.5. *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, see reference on page 3) shows the 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

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

Notation 3.6. 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 [**Corrected 31 January from** $(n, n, 2)$], 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.

4. Nearest neighbour decoding

Let C be a code. Suppose that a codeword is sent through the channel and we receive the word v. The natural way to decode v is to look through all the codewords of C and to pick the one that is nearest, in the Hamming distance, to v. This procedure is called *nearest neighbour decoding*. It could be that there is no unique nearest codeword to v. In this case we shall say that nearest neighbour decoding *fails*.[7]

You have already used nearest neighbour decoding to decode words in the quizzes on Example 2.5 and Lemma 2.11 (see the slides for Part A on Moodle).

The example below uses the 1-error correcting code seen on Question 4 on Sheet 1.

Example 4.1. Let $C = \{00000, 11100, 00111, 11011\}$.

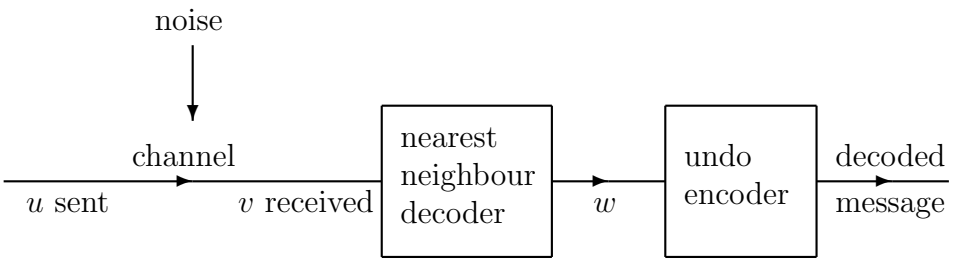
(i) Suppose $v = 11111$ is received. Then since $d(11011, 11111) = 1$ and $d(u', 11111) \geq 2$ for all codewords $u' \neq 11011$, we would decode v as 11011 using nearest neighbour decoding.

(ii) Suppose $v = 01110$ is received. Then nearest neighbour decoding fails because the nearest codewords to v are 11100 and 00111, both at distance 2.

Nearest neighbour decoding is only one step in the decoding process. To recover the sent word, the decoder must take the codeword given by nearest neighbour decoding and then undo the encoder by translating the codeword back into a message. It is therefore helpful to extend our model of decoding to a two-step process. (The quiz on Lemma 2.11 gives an example of this.)

In the diagram overleaf, the codeword u is sent, the word v is received, and nearest neighbour decoding gives the codeword w. 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', as in Definition 2.7.

[7] In practice it might be essential that the decoder always gives some result, even if it is quite likely to be wrong; then the decoder will have to choose in some arbitrary way from the nearest codewords to v, and hope for the best.



In Remark 2.8(3) we remarked that a t-error correcting code promises to correct up to t errors, *provided a suitable decoder is used*. The nearest neighbour decoder shown in the diagram above is this suitable decoder.

Exercise: Let C be a code. Show that C is t-error correcting $\iff$ whenever at most t errors occur in the channel, decoding a received word using nearest neighbour decoding gives the sent codeword.

This is a convenient place to give a more geometric way of looking at Hamming distance and t-error correcting codes.

Definition 4.2. 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

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

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.

Example 4.3. 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 $B_4(0000)$ consists of all binary words of length 4.

The next lemma is essentially a restatement of the exercise above. Recall that two sets are said to be disjoint if no element lies in both of them.

Lemma 4.4. *Let C be a code. Then C is t-error correcting $\iff$ for all distinct codewords $u, u' \in C$, the Hamming balls $B_t(u)$ and $B_t(u')$ are disjoint.*

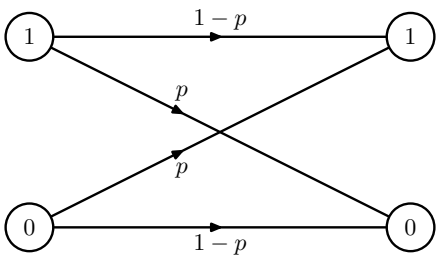
In §5 we will use Lemma 4.4 to prove the remarkable Hamming Packing Bound on the maximum size of a t-error correcting code of length n.

The next exercise shows that Lemma 4.4 can be used to speed up nearest neighbour decoding.[8]

Exercise: Let C be a t-error correcting code. Suppose that you receive a word v, and that after searching through some of the codewords in C for its nearest neighbour, you find a codeword u such that $d(u, v) \leq t$. Explain why u must be the unique nearest codeword to v.

We end Part A with the probabilistic justification for nearest neighbour decoding in the case of binary codes.

Consider the binary channel in which each transmitted bit flips, so a 0 becomes a 1 and a 1 becomes a 0, independently with probability $p > 0$. This channel is known as the *binary symmetric channel*. (It has already been seen in Example 1.2 and Question 1 on the preliminary problem sheet.) The probability p is called the *cross-over probability*. The transition probabilities in the binary symmetric channel are shown in the diagram below.



Theorem 4.5. *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 $u \in C$,*

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

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

[8]Even with this speed up, the naïve implementation of nearest neighbour decoding is too slow to be used on many codes of interest. For example the 16-error correcting Reed–Solomon code used on compact discs has size $2^{8\times 223}$, which is far too large to search through. Fortunately there are faster algorithms that will faithfully implement nearest neighbour decoding, provided at most 16 errors occur (see the MSc/MSci course).

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 4.5 may be regarded as non-examinable.

The assumption in Theorem 4.5 that each codeword is equally likely to be transmitted is critical. See the optional questions on Sheet 3 for a case where nearest neighbour decoding no longer works well because one codeword is much more likely to be sent than another.[9]

Summary of Part A. In Part A we have seen the formal definition (Definition 2.7) of t-error detecting/correcting codes. The examples in §2 and the results in Lemmas 2.9 and 2.10 show that this definition is a reasonable one.

We then saw other ways of thinking about t-error correcting codes, using minimum distance (Definition 3.1) and nearest neighbour decoding. These are summarised in the following theorem.

Theorem 4.6. *Let C be a code. The following are equivalent*

(a) *C is t-error correcting;*
(b) *The minimum distance of C is at least $2t+1$;*
(c) *Nearest neighbour decoding always gives the sent codeword, provided at most t errors occur;*
(d) *If u, $u' \in C$ are distinct codewords then the Hamming balls $B_t(u)$ and $B_t(u')$ are disjoint.*

Proof. This is just a matter of putting together results already proved:

- (a) $\iff$ (b): Theorem 3.4(ii),
- (a) $\iff$ (c): Exercise on page 18,
- (a) $\iff$ (d): Lemma 4.4. □

One might say that (c) shows the algorithmic side of the subject: it tells us how to decode a t-error correcting code (and is informed by probabilistic ideas), while (b) and (d) shows the geometric side of the subject. We saw in Lemma 3.3 that (b) is often the easier condition to work with; we will use it throughout Parts B and C.

[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 transmit 'healthy' with probability 999/1000 and 'ill' with probability 1/1000, but any transmission of 'healthy' 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.

Part B: Main Coding Theory Problem

5. Main Coding Theory Problem and Hamming's bound

Theorem 3.4 and Corollary 3.5 show that the maximum number of errors a code can detect or correct is determined by its minimum distance. In this part of the course we shall look at some codes of specified length and minimum distance that have as many codewords as possible.

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 is of 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.[10] Other bounds will be seen in the remainder of Part B.

We need the following combinatorial lemma. Recall that the binomial coefficient $\binom{n}{k}$ is the number of ways to choose k objects from a set of size n. One can compute binomial coefficients using the formula

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

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*

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

[10] This bound can be extended easily to codes over a general q-ary alphabet. For an outline see Sheet 4. Only the binary case is examinable.

We also need the result of Question 5 of Sheet 3, that if C has minimum distance at least $2t+1$ then the Hamming balls of radius t about distinct codewords are disjoint. (Alternatively, this is (b) $\Rightarrow$ (d) in Theorem 4.6.)

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*

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

We saw in Corollary 3.5 that a code is 1-error correcting if and only if its minimum distance is 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**. For example, Question 4 on Sheet 3 implies that a binary code of length 5 and minimum distance 3 has size at most 4. The exercise below gives a simpler example. (See also Question 2 on Sheet 4.)

Exercise: Show that the largest size of a binary code of length 4 and minimum distance 3 is 2.

6. Equivalences of codes and $A_q(n, d)$

The bounds we will prove in Part B can be stated very concisely using the following definition.

Definition 6.1. 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 the choice of which particular q-ary alphabet to use makes no difference to $A_q(n, d)$.

Exercise: 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 two exercises imply that $A_q(n, d)$ is well-defined.[11] 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 following lemma gives two results for general q.

Lemma 6.2. *Let $q \geq 2$ and let $n \in \mathbf{N}$. Then*

(i) $A_q(n, 1) = q^n$;
(ii) $A_q(n, n) = q$.

Question 4 on Sheet 3 implies that $A_2(5, 3) \leq 4$. We have already seen that there are binary codes of length 5 and minimum distance 3, for example $C = \{00000, 11100, 00111, 11011\}$ in Example 4.1, and so $A_2(5, 3) = 4$.

In the argument suggested for this question we supposed that C was a binary $(5, M, 3)$-code with $M \geq 4$. We then flipped bits in all the codewords to assume, without loss of generality, that 00000 was a codeword, and then shuffled bits in the codewords to assume, again without loss of generality, that 11100 was a codeword. These operations are examples of *equivalences of codes*. We shall use equivalences to solve the Main Coding Theory Problem for some other small values of n.

Definition 6.3. 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.

Question 3 of Sheet 3 implies that the following lemma holds for binary codes. (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.

[11] 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.

Lemma 6.4. *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* (a) *and/or shuffling positions* (b) *then $d(u, w) = d(u', w')$.*

In particular, Lemma 6.4 implies that equivalent codes have the same minimum distance.

Example 6.5. 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 on Sheet 4.

Exercise: Let C be the binary code with codewords

$$11001,\ 01111,\ 10100,\ 00010.$$

and let C' be the binary code with codewords

$$00000,\ 11100,\ 00111,\ 11011$$

Show that C and C' are equivalent.

We end this section by using equivalences of codes to find $A_2(8, 5)$. The following lemma isolates the 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 $\mathrm{wt}(u) \geq r$ and $\mathrm{wt}(w) \geq s$. If $r + s \geq n$ then $d(u, w) \leq 2n - (r + s)$.*

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

Note that the proof of Theorem 6.7 actually proves something stronger: up to equivalence there is a unique binary $(8, 4, 5)$-code. On Sheet 4 you are asked to use similar ideas to show that $A_2(9, 6) = 4$.

For background, the table below shows some other values of $A_2(n, d)$. 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

Note that the values of $A_2(n, d)$ are not always powers of 2. For example, $A_2(8, 3) = 20$; equivalently (by Theorem 3.4) the largest 1-error correcting of length 8 has 20 codewords. The reason why $A_2(n, d)$ often is a power of 2 is that many good codes are linear (see Part C), and all linear binary codes have size a power of 2.

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 a pair of 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 '$\Rightarrow$' direction. See Sheet 5 for the '$\Leftarrow$' 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 $1 \leq i, j \leq q-1$. 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 whenever there is a pair of MOLs of order q we have $A_q(4,3) = q^2$.

(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 6.2(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 1.11. Since the code has minimum distance 16, it follows from Theorem 3.4(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 that $M \leq 2^{d+1}$. 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 give an alternative proof of the result of Question 2(b) on Sheet 4, 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. 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: this is entirely non-examinable and mentioned for interest only.)

10. Gilbert–Varshamov Bound

Stated using the $A_2(n, d)$ notation introduced in Definition 6.1, the 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 just keep on adding codewords until the Hamming balls of radius $(d-1)$ 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*

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

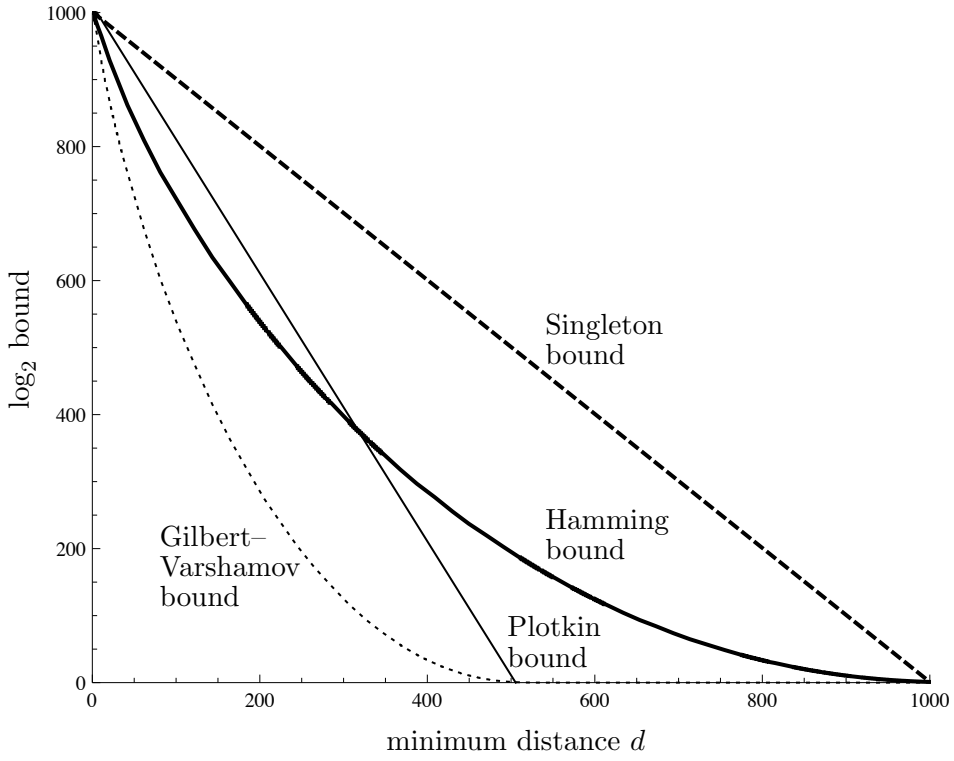
Summary of Part B. The Main Coding Theory Problem asks for codes over a given q-ary alphabet with small length n, high size M and high minimum distance d. To study these conflicting requirements, we defined $A_q(n, d)$ to be the largest size M of a q-ary code of length n and minimum distance d.

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.

These upper bounds and the Gilbert–Varshamov lower bound are compared in the diagram below 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. Note that the graph shows $\log_2$ of each bound.

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 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, which shows all the main ideas. The extension to larger alphabets of prime power degree is not too 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 $u + w \in C$.

Note that if C is a linear binary code and $u \in C$ then $u + u = (0, 0, \ldots, 0)$. So it follows from Definition 8.1 that any binary code contains the all-zeros word, written $\mathbf{0}$ (or $\underline{0}$ on the board).

We have already seen many examples of linear codes.

Example 11.2. Let $n \in \mathbf{N}$.

(1) The length 5 code $C = \{00000, 11100, 00111, 11011\}$ is linear.

(2) The binary repetition code of length n is a linear $(n, 2, n)$-code.

(3) 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 2.6, 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.

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, $A_2(5,3) = 4$, and a small extension of Question 4 on Sheet 3 shows that any binary $(5,4,3)$-code is equivalent to

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

We saw that this code was linear in Example 11.2(1).

The following exercise is strongly recommended.

Exercise: Show that the square code (see Question 2 on Preliminary Problem Sheet) 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; check that the results are as expected.

The last result in this section generalises the parity check extension codes seen in Example 2.6 and Example 11.2(2). (For a related result see Questions 5 and 6 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. *Suppose that C be a linear binary (n, M, d)-code. If d is odd then the parity check extension C_{ext} of C is a linear binary $(n+1, M, d+1)$-code.*

12. Bases, generator matrices and encoding

To proceed any further we need to think of binary words as vectors and linear binary codes as subspaces of $\mathbf{Z}_2^n$.

Exercise: Show from Definition 11.1 that if $u(1), \ldots, u(m)$ are codewords in a linear binary code C and $x(1), \ldots, x(m) \in \mathbf{Z}_2$ then

$$x(1)u(1) + \cdots + x(m)u(m) \in C.$$

Definition 12.1. Let C be a linear code. We say that codewords $u(1), \ldots, u(m) \in C$ are:

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

$$x(1)u(1) + \cdots + x(m)u(m) = 0$$

with $x(1), \ldots, x(m) \in \mathbf{Z}_2$ is $x(1) = \ldots = x(m) = 0$.

(b) *span* C if for every $w \in C$ there exist $x(1), \ldots, x(m) \in \mathbf{Z}_2$ such that

$$w = x(1)u(1) + \cdots + x(m)u(m).$$

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

Example 12.2.

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

(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.

Note that a linear binary code may have several different bases. So 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 spanning set in Example 12.6(2).

Exercise: Find another basis for the code C_{ext} in Example 12.2(2).

One reason why having a basis is useful is the following result. The proof is nothing particular to do with coding theory, so will not be covered in full in lectures (and is non-examinable).

Lemma 12.3. *Let C be a linear code. The codewords*

$$u(1), \ldots, u(m) \in C$$

are a basis for C $\iff$ for all $w \in C$ there exist unique *$x(1), \ldots, x(m)$, such that*

$$w = x(1)u(1) + \cdots + x(m)u(m).$$

Suppose that $u(1), \ldots, u(m)$, is a basis for a linear binary code C. Then, by Lemma 12.4

$$C = \{x(1)u(1) + \cdots + x(m)u(m) : x(1), \ldots, x(m) \in \mathbf{Z}_2\}$$

and C has 2^m elements, coming from the two choices for each of $x(1), \ldots, x(m)$. It follows that any two bases of C have the same size[12]. We also see that any linear binary code has size 2^m for some m.

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

Thus a linear binary $(n, 2^m, d)$-code is a $[n, m, d]$-code. The codes in Example 12.2 have parameters $[5, 2, 3]$ and $[5, 4, 2]$, respectively.

Exercise: (To appear on Sheet 7.) Find a basis for the square code.

Definition 12.5. Suppose that C is a linear binary code of length n having $u(1), \ldots, u(m) \in \mathbf{Z}_2^n$ as a basis. The $m \times n$ matrix G with rows $u(1), \ldots, u(m)$ is said to be a *generator matrix* for C.

Example 12.6.

(1) Let $C = \{00000, 11100, 00111, 11011\}$. Then a generator matrix for C is

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

(2) 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

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

[12]This is a standard result from linear algebra. The proof indicated here only works over finite fields such as $\mathbf{Z}_2$, but is much shorter than the usual proof.

(3) The parity check code C_{ext} in Example 12.2(2) has as a generator matrix

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

This is a convenient place to discuss encoding for linear codes. Let C be a linear code of dimension m with generator matrix G. We have seen that C has size 2^m, so C can encode 2^m different messages. We will assume that the messages are numbers between 0 and 2^{m-1}.

To encode a message, write its number in binary, say as $b_1 b_2, \ldots b_m$, and take the codeword

$$(b_1, b_2, \ldots, b_m)G.$$

Note that if G has rows $u(1), u(2), \ldots, u(m)$ then

$$(b_1, b_2, \ldots, b_m)G = b_1 u(1) + b_2 u(2) + \cdots + b_m u(m).$$

It follows from the exercise at the start of this section that $b_1 u(1) + b_2 u(2) + \cdots + b_m u(m)$ is a codeword in C.

Example 12.7. Let C be the linear code in Example 12.6(2). In this example 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 6 we write 6 in binary as 110 and take the codeword

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

In general, the binary number $b_1 b_2 b_3$ is encoded as

$$(b_1, b_2, b_1 + b_2, b_3, b_1 + b_3, b_2 + b_3, b_1 + b_2 + b_3).$$

Note that in Example 12.7, 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.

Definition 12.8. A generator matrix G for a linear binary $[n, m, d]$-code is said to be in *standard* form if

$$G = \begin{pmatrix} I_m & A \end{pmatrix}$$

where I_m is the $m \times m$ identity matrix and A is a $m \times (n - m)$ matrix.

The generator matrix in Example 12.6(3) is in standard form. If we swap positions 3 and 4 in all the codewords in the code C in Example 12.6(2) and Example 12.7 then 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 12.9. *Let C be a linear binary code of length n and dimension m. Then C is equivalent, by a permutation of the positions in the codewords, to a code with a generator matrix in standard form*

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

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

In general, if $G = \begin{pmatrix} I_m & A \end{pmatrix}$ is a generator matrix for a code C in standard form then the message labelled $(b_1, b_2, \ldots, b_m)$ is encoded as

$$(b_1, b_2, \ldots, b_m)G = (b_1, b_2, \ldots, b_m, c_1, \ldots, c_{n-m})$$

for some $c_1, \ldots, c_{n-m} \in \mathbf{Z}_2$. This is convenient because if no errors occur in transmission, then the message can be easily read off from the received word. We end with a related definition.

Definition 12.10. Let C be a code of length n and size M. We define the *rate* of C to be $(\log_2 M)/n$.

Thus if C is a linear binary $[n, m, d]$-code then C has 2^m codewords and the rate of C is m/n. Roughly put, the rate of a code measures the proportion of bits that give direct information about the encoded message.

13. Decoding by standard arrays

In this section we shall see a way to implement nearest neighbour decoding for linear codes.

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 $v = \mathbf{0} + v$ so $v \in C + v$.

Example 13.2. Let C be the linear binary code

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

obtained by puncturing the code in Example 12.2(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 in this coset is distance 1 from two codewords, 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 $C + w$ are cosets of C then either $C + v = C + w$ or the cosets $C + v$ and $C + w$ 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. 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*.

Example 13.5. A standard array for the code C in Example 10.2 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. (Both 0010 and 0001 have weight 1.) The other coset leaders 0000, 1000 and 0100 are uniquely determined by their cosets.

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 $u \in C$.

If C is e-error correcting and the coset leader x in the lemma above has weight $\leq e$ then

$$d(v + x, v) = d(x, 0) = \mathrm{wt}(x) \leq e.$$

So $v + x$ must be the *unique* nearest codeword to v, and decoding using the standard array finds this codeword. This shows that the standard array gives an efficient way to implement nearest neighbour 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 $\mathrm{C}_{\mathrm{ext}}$ consisting of all binary words of length 5 of even weight can be defined by

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

Similarly, the square code can be defined by

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

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.[13]

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 m can be defined by $n - m$ linear equations.

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

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

Example 14.2.

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

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

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

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

[13] 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 (by Theorem 3.4), and so we do not expect to be able to decode reliably in this case.

We now show that *any* linear binary code can be defined by a suitable parity check matrix.

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

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

is a parity check matrix for C.

For example, if $G = \begin{pmatrix} I_m & A \end{pmatrix}$ 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).

This is a convenient place to make the following definition.

Definition 14.4. Let C be a linear binary code of length n 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 $n - m$ with generator matrix H.

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

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

It is also possible to start with a $(n-m) \times n$ 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, so to show C has minimum distance 3, it suffices to show there are no codewords of weight 1 or 2: we will show this using 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.[14]

Theorem 14.7. *Let C be a linear binary code of length n and dimension m 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.

Definition 14.8. Let C be a linear binary code of length n and dimension m with parity check matrix H. The *syndrome* of a word $v \in \mathbf{Z}_2^n$ is defined to be $vH^{tr} \in \mathbf{Z}_2^{n-m}$.

By Theorem 14.7 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 14.9. Let $C = \{0000, 1110, 0011, 1101\}$ be the code used as an example in §13. Then C has parity check matrix

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

[14] Algebraically inclined readers will notice that since $C = \ker H^{tr}$, Theorem 14.7 follows from the first isomorphism theorem for the linear map $\mathbf{Z}_2^n \to \mathbf{Z}_2^{n-m}$ defined by $v \mapsto vH^{tr}$. Everyone else may ignore this remark.

By Theorem 14.7, 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 $C + 1000$.

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

A defect of the code C is that $C + 0010 = C + 0001$ and so the single bit errors 0010 and 0001 have the same syndrome. If C had to be used in practice, one possibility would be to use *incomplete decoding* and request retransmission whenever a received word has syndrome 01.

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

Example 14.10. 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, so by Lemma 13.3 and Theorem 14.7 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 so we decode v as $v + e(i)$.

For example, to use C to send the number 9, we would write 9 as 1001 in binary, and encode it as

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

Suppose that when we transmit 0011001, an error occurs in position 6, so 0011011 is received. Then the syndrome of the received word is

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

which is row 6 of H^{tr}. So we decode by changing the bit in position 6 to get 0011001.

More generally, let C be a linear binary code of length n and dimension m. 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 $\mathbf{Z}_2^{n-m}$ have appeared. Then, by Theorem 14.7, 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 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 m has a $m \times n$ generator matrix. This gives a concise way to specify the code: rather than give 2^m codewords we can specify m basis elements. Generator matrices are also useful for encoding: see page 37, and in proofs.

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.