Foreword

This chapter is based on lecture notes from coding theory courses taught by Venkatesan Guruswami at University at Washington and CMU; by Atri Rudra at University at Buffalo, SUNY and by Madhu Sudan at MIT.

This version is dated **July 16, 2014**. For the latest version, please go to

http://www.cse.buffalo.edu/ atri/courses/coding-theory/book/

The material in this chapter is supported in part by the National Science Foundation under CAREER grant CCF-0844796. Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Chapter 1

The Fundamental Question

1.1 Overview

Communication is a fundamental need of our modern lives. In fact, communication is something that humans have been doing for a long time. For simplicity, let us restrict ourselves to English. It is quite remarkable that different people speaking English can be understood pretty well: even if e.g. the speaker has an accent. This is because English has some built-in redundancy, which allows for “errors" to be tolerated. This came to fore for one of the authors when his two and a half year old son, Akash, started to speak his own version of English, which we will dub “Akash English." As an example,

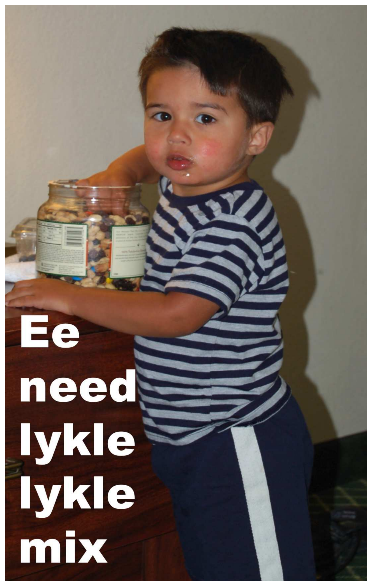

Figure 1.1: Decoding in Akash English, one gets “I need little little (trail)mix."

With some practice Akash’s parents were able to “decode" what Akash really meant. In fact, Akash could communicate even if he did not say an entire word properly and gobbled up part(s) of word(s).

The above example shows that having redundancy in a language allows for communication even in the presence of (small amounts of) differences and errors. Of course in our modern digital world, all kinds of entities communicate (and most of the entities do not communicate in English or any natural language for that matter.) Errors are also present in the digital world, so these digital communications also use redundancy.

Error-correcting codes (henceforth, just codes) are clever ways of representing data so that one can recover the original information even if parts of it are corrupted. The basic idea is to judiciously introduce redundancy so that the original information can be recovered even when parts of the (redundant) data have been corrupted.

For example, when packets are transmitted over the Internet, some of the packets get corrupted or dropped. Packet drops are resolved by the TCP layer by a combination of sequence numbers and ACKs. To deal with data corruption, multiple layers of the TCP/IP stack use a form of error correction called CRC Checksum [57]. From a theoretical point of view, the checksum is a terrible code (for that matter so is English). However, on the Internet, the current dominant mode of operation is to detect errors and if errors have occurred, then ask for retransmission. This is the reason why the use of checksum has been hugely successful in the Internet. However, there are other communication applications, where re-transmission is not an option. Codes are used when transmitting data over the telephone line or via cell phones. They are also used in deep space communication and in satellite broadcast (for example, TV signals are transmitted via satellite). Indeed, asking the Mars Rover to re-send an image just because it got corrupted during transmission is not an option– this is the reason that for such applications, the codes used have always been very sophisticated.

Codes also have applications in areas not directly related to communication. In particular, in the applications above, we want to communicate over space. Codes can also be used to communicate over time. For example, codes are used heavily in data storage. CDs and DVDs work fine even in presence of scratches precisely because they use codes. Codes are used in Redundant Array of Inexpensive Disks (RAID) [8] and error correcting memory [7]. (Sometimes in the Blue Screen of Death displayed by Microsoft Windows family of operating systems, you might see a line saying something along the lines of “parity check failed"– this happens when the code used in the error-correcting memory cannot recover from error(s). Also for certain consumers of memory, e.g. banks, do not want to suffer from even one bit flipping (this e.g. could mean someone’s bank balance either got halved or doubled– neither of which are welcome.[1]) Codes are also deployed in other applications such as paper bar codes, for example, the bar code used by UPS called MaxiCode [6]. Unlike the Internet example, in all of these applications, there is no scope for “re-transmission."

In this book, we will mainly think of codes in the communication scenario. In this framework, there is a sender who wants to send (say) k message symbols over a noisy channel. The sender first *encodes* the k message symbols into n symbols (called a *codeword*) and then sends

[1]This is a bit tongue-in-cheek: in real life banks have more mechanisms to prevent one bit flip from wreaking havoc.

it over the *channel*. The receiver gets a *received word* consisting of n symbols. The receiver then tries to *decode* and recover the original k message symbols. Thus, encoding is the process of adding redundancy and decoding is the process of removing errors.

Unless mentioned otherwise, in this book we will make the following assumption

The sender and the receiver only communicate via the channel.[a] In other words, other then some setup information about the code, the sender and the receiver do not have any other information exchange (other than of course what was transmitted over the channel). In particular, no message is more likely to be transmitted over another.

[a]The scenario where the sender and receiver have a "side-channel" is an interesting topic that has been studied but it outside the scope of this book.

The fundamental question that will occupy our attention for almost the entire book is the tradeoff between the amount of redundancy used and the number of errors that can be corrected by a code. In particular, we would like to understand

Question 1.1.1. *How much redundancy do we need to correct a given amount of errors? (We would like to correct as many errors as possible with as little redundancy as possible.)*

Intuitively, maximizing error correction and minimizing redundancy are contradictory goals: a code with higher redundancy should be able to tolerate more number of errors. By the end of this chapter, we will see a formalization of this question.

Once we determine the optimal tradeoff, we will be interested in achieving the optimal tradeoff with codes with *efficient* encoding and decoding. (A DVD player that tells its consumer that it will recover from a scratch on a DVD by tomorrow is not going to be exactly a best-seller.) In this book, we will primarily define efficient algorithms to be ones that run in polynomial time.[2]

1.2 Some definitions and codes

To formalize Question 1.1.1, we begin with the definition of a code.

Definition 1.2.1 (Code)**.** A code of *block length* n over an *alphabet* Σ is a subset of Σ^n. Typically, we will use q to denote $|\Sigma|$.[3]

Remark 1.2.1. We note that the ambient space Σ^n, viewed as a set of sequences, vectors or functions. In other words, we can think of a vector $(v_1,\ldots,v_n) \in \Sigma^n$ as just the sequence $v_1,\ldots,v_n$ (in order) or a vector tuple $(v_1,\ldots,v_n)$ or as the function $f : [n] \to \Sigma$ such that $f(i) = v_i$. Sequences

[2]We are not claiming that this is the correct notion of efficiency in practice. However, we believe that it is a good definition as the "first cut"– quadratic or cubic time algorithms are definitely more desirable than exponential time algorithms: see Section C.4 for more on this.

[3]Note that q need not be a constant and can depend on n: we'll see codes in this book where this is true.

assume least structure on Σ and hence, are most generic. Vectors work well when Σ has some structure (and in particular is what is known as a *field*, which we will see next chapter). Functions work when the set of coordinates has structure (e.g., [n] may come from a finite field of size n). In such cases functional representation will be convenient. For now however the exact representation does not matter and the reader can work with representation as sequences.

We will also frequently use the following alternate way of looking at a code. Given a code $C \subseteq \Sigma^n$, with $|C| = M$, we will think of C as a mapping of the following form:

$$C : [M] \to \Sigma^n,$$

where $[x]$ for any integer $x \geq 1$ denotes the set $\{1, 2, , \ldots, x\}$.

We will also need the notion of *dimension* of a code.

Definition 1.2.2 (Dimension of a code)**.** Given a code $C \subseteq \Sigma^n$, its *dimension* is given by

$$k \stackrel{\text{def}}{=} \log_q |C|.$$

Let us begin by looking at two specific codes. Both codes are defined over $\Sigma = \{0, 1\}$ (also known as *binary codes*). In both cases $|C| = 2^4$ and we will think of each of the 16 messages as a 4 bit vector.

We first look at the so called *parity code*, which we will denote by $C_{\oplus}$. Given a message $(x_1, x_2, x_3, x_4) \in \{0, 1\}^4$, its corresponding codeword is given by

$$C_{\oplus}(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, x_1 \oplus x_2 \oplus x_3 \oplus x_4),$$

where the $\oplus$ denotes the EXOR (also known as the XOR or Exclusive-OR) operator. In other words, the parity code appends the parity of the message bits (or taking the remainder of the sum of the message bits when divided by 2) at the end of the message. Note that such a code uses the minimum amount of non-zero redundancy.

The second code we will look at is the so called *repetition code*. This is a very natural code (and perhaps the first code one might think of). The idea is to repeat every message bit a fixed number of times. For example, we repeat each of the 4 message bits 3 times and we us use $C_{3,rep}$ to denote this code.

Let us now try to look at the tradeoff between the amount of redundancy and the number of errors each of these codes can correct. Even before we begin to answer the question, we need to define how we are going to measure the amount of redundancy. One natural way to define redundancy for a code with dimension k and block length n is by their difference $n - k$. By this definition, the parity code uses the least amount of redundancy. However, one “pitfall" of such a definition is that it does not distinguish between a code with $k = 100$ and $n = 102$ and another code with dimension and block length 2 and 4 respectively. Intuitively the latter code is using more redundancy. This motivates the following notion of measuring redundancy.

Definition 1.2.3 (Rate of a code)**.** The *rate* of a code with dimension k and block length n is given by

$$R \stackrel{\text{def}}{=} \frac{k}{n}.$$

Note that higher the rate, lesser the amount of redundancy in the code. Also note that as $k \le n$, $R \le 1$.[4] Intuitively, the rate of a code is the average amount of real information in each of the n symbols transmitted over the channel. So in some sense rate captures the complement of redundancy. However, for historical reasons we will deal with the rate R (instead of the more obvious $1-R$) as our notion of redundancy. Given the above definition, $C_{\oplus}$ and $C_{3,rep}$ have rates of $\frac{4}{5}$ and $\frac{1}{3}$. As was to be expected, the parity code has a higher rate than the repetition code.

We have formalized the notion of redundancy as the rate of a code. To formalize Question 1.1.1, we need to formally define what it means to correct errors. We do so next.

1.3 Error correction

Before we formally define error correction, we will first formally define the notion of *encoding*.

Definition 1.3.1 (Encoding function)**.** Let $C \subseteq \Sigma^n$. An equivalent description of the code C is by an injective mapping $E : [|C|] \to \Sigma^n$ called the encoding function.

Next we move to error correction. Intuitively, we can correct a received word if we can recover the transmitted codeword (or equivalently the corresponding message). This "reverse" process is called *decoding*.

Definition 1.3.2 (Decoding function)**.** Let $C \subseteq \Sigma^n$ be a code. A mapping $D : \Sigma^n \to [|C|]$ is called a decoding function for C.

The definition of a decoding function by itself does not give anything interesting. What we really need from a decoding function is that it recovers the transmitted message. This notion is captured next.

Definition 1.3.3 (Error Correction)**.** Let $C \subseteq \Sigma^n$ and let $t \ge 1$ be an integer. C is said to be *t-error-correcting* if there exists a decoding function D such that for every message $\mathbf{m} \in [|C|]$ and error pattern $\mathbf{e}$ with at most t errors, $D(C(\mathbf{m}) + \mathbf{e})) = \mathbf{m}$.

Figure 1.3 illustrates how the definitions we have examined so far interact.

We will also very briefly look at a weaker form of error recovery called *error detection*.

Definition 1.3.4 (Error detection)**.** Let $C \subseteq \Sigma^n$ and let $t \ge 1$ be an integer. C is said to be *t-error-detecting* if there exists a detecting procedure D such that for every message $\mathbf{m}$ and every error pattern $\mathbf{e}$ with at most t errors, D outputs a 1 if $(C(\mathbf{m}) + \mathbf{e}) \in C$ and 0 otherwise.

Note that a t-error correcting code is also a t-error detecting code (but not necessarily the other way round): see Exercise 1.1. Although error detection might seem like a weak error recovery model, it is useful in settings where the receiver can ask the sender to re-send the message. For example, error detection is used quite heavily in the Internet.

With the above definitions in place, we are now ready to look at the error correcting capabilities of the codes we looked at in the previous section.

[4]Further, in this book, we will always consider the case $k > 0$ and $n < \infty$ and hence, we can also assume that $R > 0$.

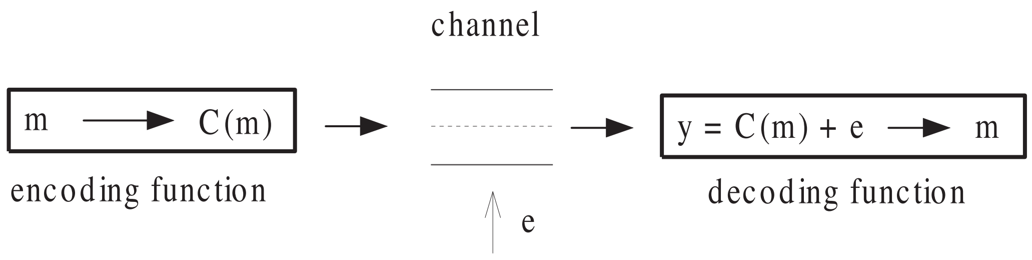

Figure 1.2: Coding process

1.3.1 Error-Correcting Capabilities of Parity and Repetition codes

In Section 1.2, we looked at examples of parity code and repetition code with the following properties:

$$C_{\oplus} : q = 2, k = 4, n = 5, R = 4/5.$$

$$C_{3,rep} : q = 2, k = 4, n = 12, R = 1/3.$$

We will start with the repetition code. To study its error-correcting capabilities, we will consider the following natural decoding function. Given a received word $\mathbf{y} \in \{0,1\}^{12}$, divide it up into four consecutive blocks (y_1, y_2, y_3, y_4) where every block consists of three bits. Then, for every block y_i $(1 \le i \le 4)$, output the majority bit as the message bit. We claim this decoding function can correct any error pattern with at most 1 error. (See Exercise 1.2.) For example, if a block of 010 is received, since there are two 0's we know the original message bit was 0. In other words, we have argued that

Proposition 1.3.1. *$C_{3,rep}$ is a 1-error correcting code.*

However, it is not too hard to see that $C_{3,rep}$ cannot correct two errors. For example, if both of the errors happen in the same block and a block in the received word is 010, then the original block in the codeword could have been either 111 or 000. Therefore in this case, no decoder can successfully recover the transmitted message.[5]

Thus, we have pin-pointed the error-correcting capabilities of the $C_{3,rep}$ code: it can correct one error, but not two or more. However, note that the argument assumed that the error positions can be located arbitrarily. In other words, we are assuming that the channel noise behaves arbitrarily (subject to a bound on the total number of errors). Obviously, we can model the noise differently. We now briefly digress to look at this issue in slightly more detail.

Digression: Channel Noise. As was mentioned above, until now we have been assuming the following noise model, which was first studied by Hamming:

[5]Recall we are assuming that the decoder has no side information about the transmitted message.

> Any error pattern can occur during transmission as long as the total number of errors is bounded. Note that this means that the location as well as the nature[6] of the errors is arbitrary.

We will frequently refer to Hamming's model as the Adversarial Noise Model. It is important to note that the atomic unit of error is a symbol from the alphabet. So for example, if the error pattern is $(1,0,1,0,0,0)$ and we consider the alphabet to be $\{0,1\}$, then the pattern has two errors. However, if our alphabet is $\{0,1\}^3$ (i.e. we think of the vector above as $((1,0,1),(0,0,0))$, with $(0,0,0)$ corresponding to the zero element in $\{0,1\}^3$), then the pattern has only one error. Thus, by increasing the alphabet size we can also change the adversarial noise model. As the book progresses, we will see how error correction over a larger alphabet is easier than error correction over a smaller alphabet.

However, the above is not the only way to model noise. For example, we could also have following error model:

> No more than 1 error can happen in any contiguous three-bit block.

First note that, for the channel model above, no more than four errors can occur when a codeword in $C_{3,rep}$ is transmitted. (Recall that in $C_{3,rep}$, each of the four bits is repeated three times.) Second, note that the decoding function that takes the majority vote of each block can successfully recover the transmitted codeword for *any* error pattern, while in the worst-case noise model it could only correct at most one error. This channel model is admittedly contrived, but it illustrates the point that the error-correcting capabilities of a code (and a decoding function) are crucially dependent on the noise model.

A popular alternate noise model is to model the channel as a stochastic process. As a concrete example, let us briefly mention the *binary symmetric channel with crossover probability* $0 \le p \le 1$, denoted by BSC_p, which was first studied by Shannon. In this model, when a (binary) codeword is transferred through the channel, every bit flips independently with probability p.

Note that the two noise models proposed by Hamming and Shannon are in some sense two extremes: Hamming's model assumes *no* knowledge about the channel (except that a bound on the total number of errors is known[7] while Shannon's noise model assumes *complete* knowledge about how noise is produced. In this book, we will consider only these two extreme noise models. In real life, the situation often is somewhere in between.

For real life applications, modeling the noise model correctly is an extremely important task, as we can tailor our codes to the noise model at hand. However, in this book we will not study this aspect of designing codes at all, and will instead mostly consider the worst-case noise model. Intuitively, if one can communicate over the worst-case noise model, then one could use the same code to communicate over pretty much every other noise model with the same amount of noise.

[6]For binary codes, there is only one kind of error: a bit flip. However, for codes over a larger alphabet, say $\{0,1,2\}$, 0 being converted to a 1 and 0 being converted into a 2 are both errors, but are different kinds of errors.

[7]A bound on the total number of errors is necessary; otherwise, error correction would be impossible: see Exercise 1.3.

We now return to $C_\oplus$ and examine its error-correcting capabilities in the worst-case noise model. We claim that $C_\oplus$ cannot correct even one error. Suppose 01000 is the received word. Then we know that an error has occurred, but we do not know which bit was flipped. This is because the two codewords 00000 and 01001 differ from the received word 01000 in exactly one bit. As we are assuming that the receiver has no side information about the transmitted codeword, no decoder can know what the transmitted codeword was.

Thus, since it cannot correct even one error, from an error-correction point of view, $C_\oplus$ is a terrible code (as it cannot correct even 1 error). However, we will now see that $C_\oplus$ can *detect* one error. Consider the following algorithm. Let $\mathbf{y} = (y_1, y_2, y_3, y_4, y_5)$ be the received word. Compute $b = y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus y_5$ and declare an error if $b = 1$. Note that when no error has occurred during transmission, $y_i = x_i$ for $1 \le i \le 4$ and $y_5 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$, in which case $b = 0$ as required. If there is a single error then either $y_i = x_i \oplus 1$ (for exactly one $1 \le i \le 4$) or $y_5 = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus 1$. It is easy to check that in this case $b = 1$. In fact, one can extend this argument to obtain the following result (see Exercise 1.4).

Proposition 1.3.2. *The parity code $C_\oplus$ can* detect *an odd number of errors.*

Let us now revisit the example that showed that one cannot correct one error using $C_\oplus$. Consider two codewords in $C_\oplus$, $\mathbf{u} = 00000$ and $\mathbf{v} = 10001$ (which are codewords corresponding to messages 0000 and 1000, respectively). Now consider the scenarios in which $\mathbf{u}$ and $\mathbf{v}$ are each transmitted and a single error occurs resulting in the received word $\mathbf{r} = 10000$. Thus, given the received word $\mathbf{r}$ and the fact that at most one error can occur, the decoder has no way of knowing whether the original transmitted codeword was $\mathbf{u}$ or $\mathbf{v}$. Looking back at the example, it is clear that the decoder is “confused" because the two codewords $\mathbf{u}$ and $\mathbf{v}$ do not differ in many positions. This notion is formalized in the next section.

1.4 Distance of a code

We now define a notion of distance that captures the concept that the two vectors $\mathbf{u}$ and $\mathbf{v}$ are “close-by."

Definition 1.4.1 (Hamming distance)**.** Given $\mathbf{u}, \mathbf{v} \in \Sigma^n$ (i.e. two vectors of length n) the *Hamming distance* between $\mathbf{u}$ and $\mathbf{v}$, denoted by $\Delta(\mathbf{u}, \mathbf{v})$, is defined to be the number of positions in which $\mathbf{u}$ and $\mathbf{v}$ differ.

The Hamming distance is a distance in a very formal mathematical sense: see Exercise 1.5. Note that the definition of Hamming distance just depends on the *number* of differences and not the nature of the difference. For example, for the vectors $\mathbf{u}$ and $\mathbf{v}$ from the previous section, $\Delta(\mathbf{u}, \mathbf{v}) = 2$, which is equal to the Hamming distance $\Delta(\mathbf{u}, \mathbf{w})$, where $\mathbf{w} = 01010$, even though the vectors $\mathbf{v}$ and $\mathbf{w}$ are not equal.

Armed with the notion of Hamming distance, we now define another important parameter of a code.

Definition 1.4.2 (Minimum distance)**.** Let $C \subseteq \Sigma^n$. The *minimum distance* (or just *distance*) of C is defined to be

$$d = \min_{\mathbf{c}_1 \neq \mathbf{c}_2 \in C} \Delta(\mathbf{c}_1, \mathbf{c}_2)$$

It is easy to check that the repetition code $C_{3,rep}$ has distance 3. (Indeed, any two distinct messages will differ in at least one of the message bits. After encoding, the difference in one message bit will translate into a difference of three bits in the corresponding codewords.) We now claim that the distance of $C_\oplus$ is 2. This is a consequence of the following observations. If two messages $\mathbf{m}_1$ and $\mathbf{m}_2$ differ in at least two places then $\Delta(C_\oplus(\mathbf{m}_1), C_\oplus(\mathbf{m}_2)) \geq 2$ (by only looking at the first four bits of the codewords). If two messages differ in exactly one place then the parity bits in the corresponding codewords are different which implies a Hamming distance of 2 between the codewords. Thus, $C_\oplus$ has smaller distance than $C_{3,rep}$ and can correct less number of errors than $C_{3,rep}$, which seems to suggest that a larger distance implies greater error-correcting capabilities. The next result formalizes this intuition.

Proposition 1.4.1. *The following statements are equivalent for a code C:*

1. *C has minimum distance $d \geq 2$,*
2. *If d is odd, C can correct $(d-1)/2$ errors.*
3. *C can detect $d-1$ errors.*
4. *C can correct $d-1$ erasures.*[8]

Remark 1.4.1. Property (2) above for even d is slightly different. In this case, one can correct up to $\frac{d}{2} - 1$ errors but cannot correct $\frac{d}{2}$ errors. (See Exercise 1.6.)

Before we prove Proposition 1.4.1, let us apply it to the codes $C_\oplus$ and $C_{3,rep}$ which have distances of 2 and 3 respectively. Proposition 1.4.1 implies the following facts that we have already proved:

- $C_{3,rep}$ can correct 1 errors (Proposition 1.3.1).
- $C_\oplus$ can detect 1 error but cannot correct 1 error (Proposition 1.3.2).

The proof of Proposition 1.4.1 will need the following decoding function. *Maximum likelihood decoding* (MLD) is a well-studied decoding method for error correcting codes, which outputs the codeword closest to the received word in Hamming distance (with ties broken arbitrarily). More formally, the MLD function denoted by $D_{MLD} : \Sigma^n \to C$ is defined as follows. For every $\mathbf{y} \in \Sigma^n$,

$$D_{MLD}(\mathbf{y}) = \arg\min_{\mathbf{c} \in C} \Delta(\mathbf{c}, \mathbf{y}).$$

Algorithm 1 is a naive implementation of the MLD.

[8]In the erasure noise model, the receiver *knows* where the errors have occurred. In this model, an erroneous symbol is denoted by "?", with the convention that any non-? symbol is a correct symbol.

Algorithm 1 Naive Maximum Likelihood Decoder

```
INPUT: Received word $\mathbf{y} \in \Sigma^n$
OUTPUT: $D_{MLD}(\mathbf{y})$

1: Pick an arbitrary $\mathbf{c} \in C$ and assign $\mathbf{z} \leftarrow \mathbf{c}$
2: FOR every $\mathbf{c}' \in C$ such that $\mathbf{c} \neq \mathbf{c}'$ DO
3:     IF $\Delta(\mathbf{c}', \mathbf{y}) < \Delta(\mathbf{z}, \mathbf{y})$ THEN
4:         $\mathbf{z} \leftarrow \mathbf{c}'$
5: RETURN $\mathbf{z}$
```

Proof of Proposition 1.4.1 We will complete the proof in two steps. First, we will show that if property 1 is satisfied then so are properties 2,3 and 4. Then we show that if property 1 is not satisfied then none of properties 2,3 or 4 hold.

1. **implies** 2. Assume C has distance d. We first prove 2 (for this case assume that $d = 2t+1$). We now need to show that there exists a decoding function such that for all error patterns with at most t errors it always outputs the transmitted message. We claim that the MLD function has this property. Assume this is not so and let $\mathbf{c}_1$ be the transmitted codeword and let $\mathbf{y}$ be the received word. Note that

$$\Delta(\mathbf{y}, \mathbf{c}_1) \leq t. \tag{1.1}$$

As we have assumed that MLD does not work, $D_{MLD}(\mathbf{y}) = \mathbf{c}_2 \neq \mathbf{c}_1$. Note that by the definition of MLD,

$$\Delta(\mathbf{y}, \mathbf{c}_2) \leq \Delta(\mathbf{y}, \mathbf{c}_1). \tag{1.2}$$

Consider the following set of inequalities:

$$\begin{align}
\Delta(\mathbf{c}_1, \mathbf{c}_2) &\leq \Delta(\mathbf{c}_2, \mathbf{y}) + \Delta(\mathbf{c}_1, \mathbf{y}) \tag{1.3}\\
&\leq 2\Delta(\mathbf{c}_1, \mathbf{y}) \tag{1.4}\\
&\leq 2t \tag{1.5}\\
&= d-1, \tag{1.6}
\end{align}$$

where (1.3) follows from the triangle inequality (see Exercise 1.5), (1.4) follows from (1.2) and (1.5) follows from (1.1). (1.6) implies that the distance of C is at most $d-1$, which is a contradiction.

1. **implies** 3. We now show that property 3 holds, that is, we need to describe an algorithm that can successfully detect whether errors have occurred during transmission (as long as the total number of errors is bounded by $d-1$). Consider the following error detection algorithm: check if the received word $\mathbf{y} = \mathbf{c}$ for some $\mathbf{c} \in C$ (this can be done via an exhaustive check). If no errors occurred during transmission, $\mathbf{y} = \mathbf{c}_1$, where $\mathbf{c}_1$ was the transmitted codeword and the algorithm above will accept (as it should). On the other hand if $1 \leq \Delta(\mathbf{y}, \mathbf{c}_1) \leq d-1$, then by the fact that the distance of C is d, $\mathbf{y} \notin C$ and hence the algorithm rejects, as required.

1. **implies** 4. Finally, we prove that property 4 holds. Let $\mathbf{y} \in (\Sigma \cup \{?\})^n$ be the received word. First we claim that there is a unique $\mathbf{c} = (c_1, \ldots, c_n) \in C$ that agrees with $\mathbf{y}$ (i.e. $y_i = c_i$ for every i such that $y_i \neq ?$). (For the sake of contradiction, assume that this is not true, i.e. there exists two distinct codewords $\mathbf{c}_1, \mathbf{c}_2 \in C$ such that both $\mathbf{c}_1$ and $\mathbf{c}_2$ agree with $\mathbf{y}$ in the unerased positions. Note that this implies that $\mathbf{c}_1$ and $\mathbf{c}_2$ agree in the positions i such that $y_i \neq ?$. Thus, $\Delta(\mathbf{c}_1, \mathbf{c}_2) \leq |\{i | y_i = ?\}| \leq d-1$, which contradicts the assumption that C has distance d.) Given the uniqueness of the codeword $\mathbf{c} \in C$ that agrees with $\mathbf{y}$ in the unerased position, here is an algorithm to find it: go through all the codewords in C and output the desired codeword.

$\neg$1. **implies** $\neg$2. For the other direction of the proof, assume that property 1 does not hold, that is, C has distance $d-1$. We now show that property 2 cannot hold, that is, for every decoding function there exists a transmitted codeword $\mathbf{c}_1$ and a received word $\mathbf{y}$ (where $\Delta(\mathbf{y}, \mathbf{c}_1) \leq (d-1)/2$) such that the decoding function cannot output $\mathbf{c}_1$. Let $\mathbf{c}_1 \neq \mathbf{c}_2 \in C$ be codewords such that $\Delta(\mathbf{c}_1, \mathbf{c}_2) = d-1$ (such a pair exists as C has distance $d-1$). Now consider a vector $\mathbf{y}$ such that $\Delta(\mathbf{y}, \mathbf{c}_1) = \Delta(\mathbf{y}, \mathbf{c}_2) = (d-1)/2$. Such a $\mathbf{y}$ exists as d is odd and by the choice of $\mathbf{c}_1$ and $\mathbf{c}_2$. Below is an illustration of such a $\mathbf{y}$ (same color implies that the vectors agree on those positions):

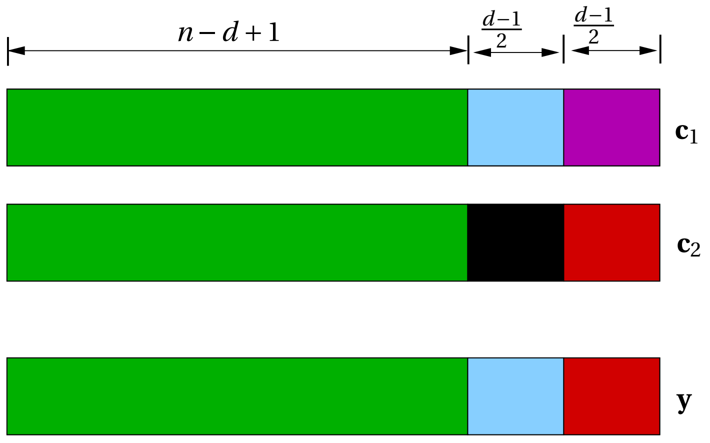

Figure 1.3: Bad example for unique decoding.

Now, since $\mathbf{y}$ could have been generated if *either* of $\mathbf{c}_1$ or $\mathbf{c}_2$ were the transmitted codeword, no decoding function can work in this case.[9]

$\neg$1. **implies** $\neg$3. For the remainder of the proof, assume that the transmitted word is $\mathbf{c}_1$ and there exists another codeword $\mathbf{c}_2$ such that $\Delta(\mathbf{c}_2, \mathbf{c}_1) = d-1$. To see why property 3 is not true, let $\mathbf{y} = \mathbf{c}_2$. In this case, either the error detecting algorithm detects no error or it declares an error when $\mathbf{c}_2$ is the transmitted codeword and no error takes place during transmission.

[9]Note that this argument is just a generalization of the argument that $C_\oplus$ cannot correct 1 error.

$\neg 1$. **implies** $\neg 4$. We finally argue that property 4 does not hold. Let $\mathbf{y}$ be the received word in which the positions that are erased are exactly those where $\mathbf{c}_1$ and $\mathbf{c}_2$ differ. Thus, given $\mathbf{y}$ both $\mathbf{c}_1$ and $\mathbf{c}_2$ could have been the transmitted codeword and no algorithm for correcting (at most $d-1$) erasures can work in this case. ■

Proposition 1.4.1 implies that Question 1.1.1 can be reframed as

Question 1.4.1. *What is the largest rate R that a code with distance d can have?*

We have seen that the repetition code $C_{3,rep}$ has distance 3 and rate 1/3. A natural follow-up question (which is a special case of Question 1.4.1) is to ask

Question 1.4.2. *Can we have a code with distance 3 and rate $R > \frac{1}{3}$?*

1.5 Hamming Code

With the above question in mind, let us consider the so called *Hamming code*, which we will denote by C_H. Given a message $(x_1, x_2, x_3, x_4) \in \{0,1\}^4$, its corresponding codeword is given by

$$C_H(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4).$$

It is easy to check that this code has the following parameters:

$$C_H : q = 2, k = 4, n = 7, R = 4/7.$$

We will show shortly that C_H has a distance of 3. We would like to point out that we could have picked the three parities differently. The reason we mention the three particular parities above is due to historical reasons. We leave it as an exercise to define alternate set of parities such that the resulting code still has a distance of 3: see Exercise 1.9.

Before we move on to determining the distance of C_H, we will need another definition.

Definition 1.5.1 (Hamming Weight)**.** Let $q \geq 2$. Given any vector $\mathbf{v} \in \{0,1,2,\ldots,q-1\}^n$, its Hamming weight, denoted by $wt(\mathbf{v})$ is the number of non-zero symbols in $\mathbf{v}$.

We now look at the distance of C_H.

Proposition 1.5.1. *C_H has distance 3.*

Proof. We will prove the claimed property by using two properties of C_H:

$$\min_{\mathbf{c} \in C_H, \mathbf{c} \neq \mathbf{0}} wt(\mathbf{c}) = 3, \tag{1.7}$$

and

$$\min_{\mathbf{c}\in C_H, \mathbf{c}\neq \mathbf{0}} wt(\mathbf{c}) = \min_{\mathbf{c}_1\neq \mathbf{c}_2 \in C_H} \Delta(\mathbf{c}_1, \mathbf{c}_2) \tag{1.8}$$

The proof of (1.7) follows from a case analysis on the Hamming weight of the message bits. Let us use $\mathbf{x} = (x_1, x_2, x_3, x_4)$ to denote the message vector.

- Case 0: If $wt(\mathbf{x}) = 0$, then $C_H(\mathbf{x}) = \mathbf{0}$, which means we do not have to consider this codeword.

- Case 1: If $wt(\mathbf{x}) = 1$ then at least two parity check bits in $(x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4)$ are 1. So in this case, $wt(C_H(\mathbf{x})) \geq 3$.

- Case 2: If $wt(\mathbf{x}) = 2$ then at least one parity check bit in $(x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4)$ is 1. So in this case, $wt(C_H(\mathbf{x})) \geq 3$.

- Case 3: If $wt(\mathbf{x}) \geq 3$ then obviously $wt(C_H(\mathbf{x})) \geq 3$.

Thus, we can conclude that $\min_{\mathbf{c}\in C_H, \mathbf{c}\neq \mathbf{0}} wt(\mathbf{c}) \geq 3$. Further, note that $wt(C_H(1,0,0,0)) = 3$, which along with the lower bound that we just obtained proves (1.7).

We now turn to the proof of (1.8). For the rest of the proof, let $\mathbf{x} = (x_1, x_2, x_3, x_4)$ and $\mathbf{y} = (y_1, y_2, y_3, y_4)$ denote the two distinct messages. Using associativity and commutativity of the $\oplus$ operator, we obtain that

$$C_H(\mathbf{x}) + C_H(\mathbf{y}) = C_H(\mathbf{x}+\mathbf{y}),$$

where the "+" operator is just the bit-wise $\oplus$ of the operand vectors. Further, it is easy to verify that for two vectors $\mathbf{u}, \mathbf{v} \in \{0,1\}^n$, $\Delta(\mathbf{u}, \mathbf{v}) = wt(\mathbf{u}+\mathbf{v})$ (see Exercise 1.10). Thus, we have

$$\begin{aligned}\min_{\mathbf{x}\neq\mathbf{y}\in\{0,1\}^4} \Delta(C_H(\mathbf{x}), C_H(\mathbf{y})) &= \min_{\mathbf{x}\neq\mathbf{y}\in\{0,1\}^4} wt(C_H(\mathbf{x}+\mathbf{y})) \\ &= \min_{\mathbf{x}\neq\mathbf{0}\in\{0,1\}^4} wt(C_H(\mathbf{x})),\end{aligned}$$

where the second equality follows from the observation that $\{\mathbf{x}+\mathbf{y} | \mathbf{x} \neq \mathbf{y} \in \{0,1\}^n\} = \{\mathbf{x} \in \{0,1\}^n | \mathbf{x} \neq \mathbf{0}\}$. Recall that $wt(C_H(\mathbf{x})) = 0$ if and only if $\mathbf{x} = \mathbf{0}$ and This completes the proof of (1.8). Combining (1.7) and (1.8), we conclude that C_H has a distance 3. □

The second part of the proof could also have been shown in the following manner. It can be verified easily that the Hamming code is the set $\{\mathbf{x} \cdot G_H | \mathbf{x} \in \{0,1\}^4\}$, where G_H is the following matrix (where we think $\mathbf{x}$ as a row vector).[10]

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

[10]Indeed $(x_1, x_2, x_3, x_4) \cdot G_H = (x_1, x_2, x_3, x_4, x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4)$, as desired.

In fact, any binary code (of dimension k and block length n) that is generated[11] by a $k \times n$ matrix is called a *binary linear code*. (Both $C_{\oplus}$ and $C_{3,rep}$ are binary linear codes: see Exercise 1.11.) This implies the following simple fact.

Lemma 1.5.2. *For any binary linear code C and any two messages $\mathbf{x}$ and $\mathbf{y}$, $C(\mathbf{x}) + C(\mathbf{y}) = C(\mathbf{x}+\mathbf{y})$.*

Proof. For any binary linear code, we have a generator matrix G. The following sequence of equalities (which follow from the distributivity and associativity properties of the boolean EXOR and AND operators) proves the lemma.

$$\begin{aligned} C(\mathbf{x}) + C(\mathbf{y}) &= \mathbf{x} \cdot G + \mathbf{y} \cdot G \\ &= (\mathbf{x}+\mathbf{y}) \cdot G \\ &= C(\mathbf{x}+\mathbf{y}) \end{aligned}$$

□

We stress that in the lemma above, $\mathbf{x}$ and $\mathbf{y}$ need *not* be distinct. Note that due to the fact that $b \oplus b = 0$ for every $b \in \{0,1\}$, $\mathbf{x}+\mathbf{x} = \mathbf{0}$, which along with the lemma above implies that $C(\mathbf{0}) = \mathbf{0}$.[12] We can infer the following result from the above lemma and the arguments used to prove (1.8) in the proof of Proposition 1.5.1.

Proposition 1.5.3. *For any binary linear code, minimum distance is equal to minimum Hamming weight of any non-zero codeword.*

Thus, we have seen that C_H has distance $d = 3$ and rate $R = \frac{4}{7}$ while $C_{3,rep}$ has distance $d = 3$ and rate $R = \frac{1}{3}$. Thus, the Hamming code is provably better than the repetition code (in terms of the tradeoff between rate and distance) and thus, answers Question 1.4.2 in the affirmative. The next natural question is

Question 1.5.1. *Can we have a distance 3 code with a rate higher than that of C_H?*

We will address this question in the next section.

1.6 Hamming Bound

Now we switch gears to present our first tradeoff between redundancy (in the form of dimension of a code) and its error-correction capability (in form of its distance). In particular, we will first prove a special case of the so called Hamming bound for a distance of 3.

We begin with another definition.

[11]That is, $C = \{\mathbf{x} \cdot G | \mathbf{x} \in \{0,1\}^k\}$, where addition is the $\oplus$ operation and multiplication is the AND operation.
[12]This of course should not be surprising as for any matrix G, we have $\mathbf{0} \cdot G = \mathbf{0}$.

Definition 1.6.1 (Hamming Ball)**.** For any vector $\mathbf{x} \in [q]^n$,

$$B(\mathbf{x}, e) = \{\mathbf{y} \in [q]^n | \Delta(\mathbf{x}, \mathbf{y}) \le e\}.$$

Next we prove an upper bound on the dimension of *every* code with distance 3.

Theorem 1.6.1 (Hamming bound for $d = 3$)**.** *Every binary code with block length n, dimension k, distance d* $= 3$ *satisfies*

$$k \le n - \log_2(n+1).$$

Proof. Given any two codewords, $\mathbf{c}_1 \neq \mathbf{c}_2 \in C$, the following is true (as C has distance[13] 3):

$$B(\mathbf{c}_1, 1) \cap B(\mathbf{c}_2, 1) = \emptyset. \tag{1.9}$$

See Figure 1.4 for an illustration. Note that for all $\mathbf{x} \in \{0,1\}^n$ (see Exercise 1.14),

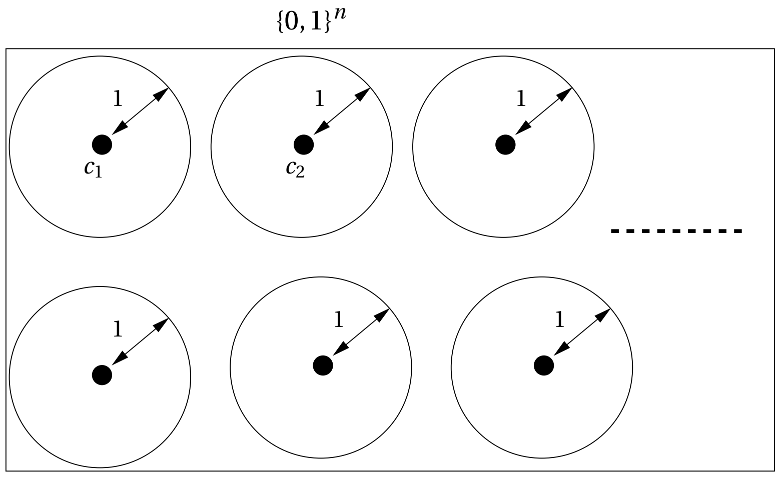

Figure 1.4: Hamming balls of radius 1 are disjoint. The figure is technically not correct: the balls above are actually balls in the Euclidean space, which is easier to visualize than the Hamming space.

$$|B(\mathbf{x}, 1)| = n + 1. \tag{1.10}$$

Now consider the union of all Hamming balls centered around some codeword. Obviously their union is a subset of $\{0,1\}^n$. In other words,

$$\left| \bigcup_{\mathbf{c} \in C} B(\mathbf{c}, 1) \right| \le 2^n. \tag{1.11}$$

[13]Assume that $\mathbf{y} \in B(\mathbf{c}_1, 1) \cap B(\mathbf{c}_2, 1)$, that is $\Delta(\mathbf{y}, \mathbf{c}_1) \le 1$ and $\Delta(\mathbf{y}, \mathbf{c}_2) \le 1$. Thus, by the triangle inequality $\Delta(\mathbf{c}_1, \mathbf{c}_2) \le 2 < 3$, which is a contradiction.

As (1.9) holds for every pair of distinct codewords,

$$\begin{aligned}\left|\bigcup_{\mathbf{c}\in C} B(\mathbf{c},1)\right| &= \sum_{\mathbf{c}\in C} |B(\mathbf{c},1)| \\ &= 2^k \cdot (n+1), \qquad (1.12)\end{aligned}$$

where (1.12) follows from (1.10) and the fact that C has dimension k. Combining (1.12) and (1.11) and taking $\log_2$ of both sides we will get the desired bound:

$$k \le n - \log_2(n+1).$$

□

Thus, Theorem 1.6.1 shows that for $n = 7$, C_H has the largest possible dimension for any binary code of block length 7 and distance 3 (as for $n = 7$, $n - \log_2(n+1) = 4$). In particular, it also answers Question 1.5.1 for $n = 7$ in the negative. Next, will present the general form of Hamming bound.

1.7 Generalized Hamming Bound

We start with a new notation.

Definition 1.7.1. A code $C \subseteq \Sigma^n$ with dimension k and distance d will be called a $(n,k,d)_\Sigma$ code. We will also refer it to as a $(n,k,d)_{|\Sigma|}$ code.

We now proceed to generalize Theorem 1.6.1 to any distance d.

Theorem 1.7.1 (Hamming Bound for any d). *For every $(n,k,d)_q$ code*

$$k \le n - \log_q\left(\sum_{i=0}^{\left\lfloor\frac{(d-1)}{2}\right\rfloor} \binom{n}{i}(q-1)^i\right).$$

Proof. The proof is a straightforward generalization of the proof of Theorem 1.6.1. For notational convenience, let $e = \left\lfloor\frac{(d-1)}{2}\right\rfloor$. Given any two codewords, $\mathbf{c}_1 \ne \mathbf{c}_2 \in C$, the following is true (as C has distance[14] d):

$$B(\mathbf{c}_1, e) \cap B(\mathbf{c}_2, e) = \emptyset. \qquad (1.13)$$

We claim that for all $\mathbf{x} \in [q]^n$,

$$|B(\mathbf{x}, e)| = \sum_{i=0}^{e} \binom{n}{i}(q-1)^i. \qquad (1.14)$$

[14]Assume that $\mathbf{y} \in B(\mathbf{c}_1, e) \cap B(\mathbf{c}_2, e)$, that is $\Delta(\mathbf{y}, \mathbf{c}_1) \le e$ and $\Delta(\mathbf{y}, \mathbf{c}_2) \le e$. Thus, by the triangle inequality, $\Delta(\mathbf{c}_1, \mathbf{c}_2) \le 2e \le d-1$, which is a contradiction.

Indeed any vector in $B(\mathbf{x}, e)$ must differ from $\mathbf{x}$ in exactly $0 \le i \le e$ positions. In the summation $\binom{n}{i}$ is the number of ways of choosing the differing i positions and in each such position, a vector can differ from $\mathbf{x}$ in $q-1$ ways.

Now consider the union of all Hamming balls centered around some codeword. Obviously their union is a subset of $[q]^n$. In other words,

$$\left|\bigcup_{\mathbf{c}\in C} B(\mathbf{c}, e)\right| \le q^n. \tag{1.15}$$

As (1.13) holds for every pair of distinct codewords,

$$\begin{aligned}\left|\bigcup_{\mathbf{c}\in C} B(\mathbf{c}, e)\right| &= \sum_{\mathbf{c}\in C} |B(\mathbf{c}, e)| \\ &= q^k \sum_{i=0}^{e} \binom{n}{i}(q-1)^i, \end{aligned} \tag{1.16}$$

where (1.16) follows from (1.14) and the fact that C has dimension k. Combining (1.16) and (1.15) and taking $\log_q$ of both sides we will get the desired bound:

$$k \le n - \log_q\left(\sum_{i=0}^{e}\binom{n}{i}(q-1)^i\right).$$

□

The Hamming bound leads to the following definition:

Definition 1.7.2. Codes that meet Hamming bound are called *perfect codes.*

Intuitively, a perfect code leads to the following perfect “packing": if one constructs Hamming balls of radius $\left\lfloor \frac{d-1}{2} \right\rfloor$ around all the codewords, then we would cover the entire ambient space, i.e. every possible vector will lie in one of these Hamming balls.

One example of perfect code is the $(7,4,3)_2$ Hamming code that we have seen in this chapter (so is the family of general Hamming codes that we will see in the next chapter). A natural question to ask is if

Question 1.7.1. *Other than the Hamming codes, are there any other perfect (binary) codes?*

We will see the answer shortly.

1.8 Exercises

Exercise 1.1. Show that any t-error correcting code is also t-error detecting but not necessarily the other way around.

Exercise 1.2. Prove Proposition 1.3.1.

Exercise 1.3. Show that for every integer n, there is no code with block length n that can handle arbitrary number of errors.

Exercise 1.4. Prove Proposition 1.3.2.

Exercise 1.5. A distance function on Σ^n (i.e. $d : \Sigma^n \times \Sigma^n \to \mathbb{R}$) is called a *metric* if the following conditions are satisfied for every $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \Sigma^n$:

1. $d(\mathbf{x}, \mathbf{y}) \geq 0$.

2. $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$.

3. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$.

4. $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$. (This property is called the *triangle inequality*.)

Prove that the Hamming distance is a metric.

Exercise 1.6. Let C be a code with distance d for even d. Then argue that C can correct up to $d/2-1$ many errors but cannot correct $d/2$ errors. Using this or otherwise, argue that if a code C is t-error correctable then it either has a distance of $2t+1$ or $2t+2$.

Exercise 1.7. In this exercise, we will see that one can convert arbitrary codes into code with slightly different parameters:

1. Let C be an $(n,k,d)_2$ code with d odd. Then it can be converted into an $(n+1,k,d+1)_2$ code.

2. Let C be an $(n,k,d)_\Sigma$ code. Then it can be converted into an $(n-1,k,d-1)_\Sigma$ code.

Note: Other than the parameters of the code C, you should not assume anything else about the code. Also your conversion should work for every $n,k,d \geq 1$.

Exercise 1.8. In this problem we will consider a noise model that has both errors and erasures. In particular, let C be an $(n,k,d)_\Sigma$ code. As usual a codeword $\mathbf{c} \in C$ is transmitted over the channel and the received word is a vector $\mathbf{y} \in (\Sigma \cup \{?\})^n$, where as before a ? denotes an erasure. We will use s to denote the number of erasures in $\mathbf{y}$ and e to denote the number of (non-erasure) errors that occurred during transmission. To decode such a vector means to output a codeword $\mathbf{c} \in C$ such that the number of positions where $\mathbf{c}$ disagree with $\mathbf{y}$ in the $n-s$ non-erased positions is at most e. For the rest of the problem assume that

$$2e + s < d. \tag{1.17}$$

1. Argue that the output of the decoder for any C under (1.17) is unique.

2. Let C be a binary code (but not necessarily linear). Assume that there exists a decoder D that can correct from $< d/2$ many errors in $T(n)$ time. Then under (1.17) one can perform decoding in time $O(T(n))$.

Exercise 1.9. Define codes other than C_H with $k = 4, n = 7$ and $d = 3$.
Hint: Refer to the proof of Proposition 1.5.1 to figure out the properties needed from the three parities.

Exercise 1.10. Prove that for any $\mathbf{u}, \mathbf{v} \in \{0,1\}^n$, $\Delta(\mathbf{u}, \mathbf{v}) = wt(\mathbf{u} + \mathbf{v})$.

Exercise 1.11. Argue that $C_{\oplus}$ and $C_{3,rep}$ are binary linear codes.

Exercise 1.12. Let G be a generator matrix of an $(n,k,d)_2$ binary linear code. Then G has at least kd ones in it.

Exercise 1.13. Argue that in any binary linear code, either all all codewords begin with a 0 of exactly half of the codewords begin with a 0.

Exercise 1.14. Prove (1.10).

Exercise 1.15. Show that there is no binary code with block length 4 that achieves the Hamming bound.

Exercise 1.16. $^{(*)}$ There are n people in a room, each of whom is given a black/white hat chosen uniformly at random (and independent of the choices of all other people). Each person can see the hat color of all other people, but not their own. Each person is asked if (s)he wishes to guess their own hat color. They can either guess, or abstain. Each person makes their choice without knowledge of what the other people are doing. They either win collectively, or lose collectively. They win if all the people who don't abstain guess their hat color correctly *and* at least one person does not abstain. They lose if all people abstain, or if some person guesses their color incorrectly. Your goal below is to come up with a strategy that will allow the n people to win with pretty high probability. We begin with a simple warmup:

(a) Argue that the n people can win with probability at least $\frac{1}{2}$.

Next we will see how one can really bump up the probability of success with some careful modeling, and some knowledge of Hamming codes. (Below are assuming knowledge of the general Hamming code (see Section 2.4). If you do not want to skip ahead, you can assume that $n = 7$ in the last part of this problem.

(b) Lets say that a directed graph G is a subgraph of the n-dimensional hypercube if its vertex set is $\{0,1\}^n$ and if $u \to v$ is an edge in G, then u and v differ in at most one coordinate. Let $K(G)$ be the number of vertices of G with in-degree at least one, and out-degree zero. Show that the probability of winning the hat problem equals the maximum, over directed subgraphs G of the n-dimensional hypercube, of $K(G)/2^n$.

(c) Using the fact that the out-degree of any vertex is at most n, show that $K(G)/2^n$ is at most $\frac{n}{n+1}$ for any directed subgraph G of the n-dimensional hypercube.

(d) Show that if $n = 2^r - 1$, then there exists a directed subgraph G of the n-dimensional hypercube with $K(G)/2^n = \frac{n}{n+1}$.
Hint: This is where the Hamming code comes in.

1.9 Bibliographic Notes

Coding theory owes its original to two remarkable papers: one by Shannon and the other by Hamming [37] both of which were published within a couple of years of each other. Shannon's paper defined the BSC$_p$ channel (among others) and defined codes in terms of its encoding function. Shannon's paper also explicitly defined the decoding function. Hamming's work defined the notion of codes as in Definition 1.2.1 as well as the notion of Hamming distance. Both the Hamming bound and the Hamming code are (not surprisingly) due to Hamming. The specific definition of Hamming code that we used in this book was the one proposed by Hamming and is also mentioned in Shannon's paper (even though Shannon's paper pre-dates Hamming's).

The notion of erasures was defined by Elias.

One hybrid model to account for the fact that in real life the noise channel is somewhere in between the extremes of the channels proposed by Hamming and Shannon is the *Arbitrary Varying Channel* (the reader is referred to the survey by Lapidoth and Narayan [48]).