25.2 Block Codes

What are Block Codes?

Every code that we will be exploring in this chapter is going to be a specific case of what we call a **block code**. Block codes are going to serve as our fundamental structure for the rest of the chapter, as every other type of code will be a block code with added restraints. Coding theory is a subject with a unique vocabulary and in this section, the aim is to acquaint the reader with block codes as well as the language which will be used throughout the chapter.

All codes begin with an **alphabet**. Let A be an alphabet of q symbols. In other words, A is a set with $q = |A|$ elements. For example $A = \{a, b, c, ..., y, z\}$ is our standard English lowercase alphabet. An important alphabet that we will be utilizing throughout this chapter is called the **binary alphabet** and is defined as such: $A = \{0, 1\}$.

Definition 25.2.1

Block Code A block code, which will also be referred to as an $[n, M]$- code over A, is a set of M n-tuples in which each n-tuple takes its components from A.

Example 25.2.2

Block Code Here is a $[5, 4]$-code over the binary alphabet:

$$C = \{00000,\ 10110,\ 01011,\ 11101\}$$

The n-tules which make up a code C are referred to as **codewords**. Note that our channel encoder is always going to be sending codewords over the channel, but once the codeword is sent and errors may be introduced by the channel noise, the n-tuple received by the decoder may not be a codeword anymore. In which case, we may refer to it as simply a **word**.

Recall that in section 25.1.1 we discussed message expansion, or **redundancy** in which we took a set of messages or k-tuples and expanded them into n-tuples for some $n \geq k$. These n-tuples are what we are referring to as our codewords. It can be useful to therefore look at the ratio between the length of our messages k and the length of our codewords n.

Definition 25.2.3

Rate The **rate** of an $[n, M]$-code that encodes k-tuples is:

$$R = \frac{k}{n}$$

Definition 25.2.4

Redundancy The **redundancy** of an $[n, M]$-code that encodes k-tuples is:

$$r = n - k$$

An important parameter of an $[n, M]$-code is a what we refer to as the **hamming distance of the code** C. First, we will define Hamming distance:

Definition 25.2.5

Hamming Distance The Hamming Distance $d(\vec{x}, \vec{y})$ between two codewords $\vec{x}$ and $\vec{y}$ is the amount of coordinate positions in which they differ

Example 25.2.6

Hamming Distance Let $\vec{x} = (2\ 1\ 0\ 0\ 2)$ and $\vec{y} = (1\ 2\ 0\ 0\ 1)$ be codewords over the alphabet $A = \{0, 1, 2\}$

Notice that these codewords differ in coordinates 1, 2, and 5

Hence, the Hamming Distance is $d(\vec{x}, \vec{y}) = 3$

In the example above, you may have noticed the use of vector notation to represent our n-tuples. It is often useful to think of codewords as vectors and we may in fact use the term "vector" interchangeably with "word" or "codeword". This analogy between codewords and vectors will become essential in the next section on **linear codes** where we wish to apply fundamental linear algebra results to our current study.

Definition 25.2.7

Hamming Distance of a Code C Let C be an $[n, M]$-code. The Hamming Distance of a code C, denoted $d(C)$ or simply d, is

$$d = min\{d(\vec{x}, \vec{y}) \mid x, y \in C, x \neq y\}$$

In other words, d is the minimum distance between two codewords in C.

In order to find the minimum distance in C, we will be required to calculate the distance between every pair of codewords. In other words, we need to check $\binom{M}{2}$ pairs of codewords. This is fine if a code is small, but for large codes this becomes inefficient. In the next section, we will restrict our attention to linear codes and this calculation will instantly become less of a chore.

Nearest neighbor Decoding

We will now look at a decoding strategy that is going to put to use our idea of Hamming distance.

Algorithm 25.2.8

Nearest Neighbor Decoding Suppose an n-tuple, $\vec{r}$, is received by the decoder.

1. If there exists a unique codeword $\vec{c} \in C$ such that $d(\vec{r}, \vec{c})$ is a minimum, then correct $\vec{r}$ to $\vec{c}$
2. If no such $\vec{c}$ exists then report that an error has been detected, but do not correct it.

In other words, nearest neighbor decoding decodes received words to the codeword which is "closest" to it with respect to Hamming distance. This procedure does not always correct the errors, as sometimes it may only detect them and on rare occasions may correct to the wrong codeword.

Example 25.2.9

Nearest Neighbor Decoding Suppose a 5-tuple, $\vec{r} = (11111)$ is received.

Suppose the code we are using here is $C = \{(00000), (10110), (01011), (11101)\}$

1. Note that
   - $d((11111), (00000)) = 5$
   - $d((11111), (10110)) = 2$
   - $d((11111), (01011)) = 2$
   - $d((11111), (11101)) = 1$

We see that the $\vec{r}$ is "closest" to $\vec{c}_4 = (11101)$. So, we decode $\vec{r}$ to $\vec{c}_4$.

The amount of "flipped bits" or **errors** that nearest neighbor decoding can decode will vary depending on our code C. The next result presents us with a way of knowing how many errors may be corrected and detected.

Theorem 25.2.10

How many errors? Let C be an $[n, M]$-code having distance d. Then C can correct $\lfloor \frac{(d-1)}{2} \rfloor$ errors. If used for detection only, C can detect $d - 1$ errors.

Example 25.2.11

How many errors? Consider the code in **Example 25.2.9**. We can verify that the minimum distance between any two codewords is 3. The distance of the code is therefore $d = 3$.

By **Theorem 25.2.10**, C can correct $\lfloor \frac{(3-1)}{2} \rfloor = 1$ error.

This explains why we were able to correct the received word in **Example 25.2.9**.

Note that C could alternatively detect $3 - 1 = 2$ errors.

Coding Spheres

A **coding sphere** is a useful geometric tool for visualizing error correction. Let $\vec{c}_i$ denote each codeword in the code C for $1 \leq i \leq M$. Let S be the set of all n-tuples over our alphabet A. Consider the set:

$$S_{c_i} = \{\vec{x} \in S \mid d(\vec{x}, \vec{c}_i) \leq e\}$$

In other words, this is the set of all n-tuples within distance e away from the codeword $\vec{c}_i$. This set is called the **sphere of radius** e. This is because we may visualize the set as a sphere of radius e with the codeword c_i at the center. Notice that since the sphere has a radius of length e and each n-tuple will be distance $d(\vec{x}, \vec{c}_i)$ away from $\vec{c}_i$, all words of distance e or less than e from $\vec{c}_i$ will lie within the sphere. For two distanct codewords, their spheres will never overlap. In other words, $S_{c_i} \cap S_{c_j} = \emptyset$ for $i \neq j$. It is this fact that is the bulk of the intuition behind **theorem 25.2.10**.