25. Coding Theory

25.1 Introduction to Error Correcting Codes

In the 1940's, mathematician Richard Hamming faced a problem while working at Bell Telephone Laboratories. The laboratory's computers stored information on punch cards. The cards represented a 1 with a hole and a 0 with no hole. The trouble was that this system was highly sensitive to errors. Often cards would get bent or mispunched, which meant that unintended holes would be punched or holes missed. This resulted in a "flipped bit". These errors would cause the entire system to hault until errors could be manually found and corrected. Hamming took it upon himself to devise a system in which errors could both detect and correct themselves...

Coding theory or the field of error correcting codes is a branch of mathematics which deals with the reliable transmission and storage of data. Let's consider some means of digital data transmission. We may think of a compact disc (a "CD"), a QR code, our computer memory, or maybe even a space probe sending images hundreds of millions of kilometers back to Earth. What do all these means have in common? They are all prone to errors. A lazer beam in your CD player attempts to read the data off your favourite CD, encoded as a sequence of 1's and 0's. Scratches and fingerprints on your CD interfere with this process.

They "flip bits" and introduce errors to the data. When we take pictures of a QR code, our image may be blurry and in our computers and electronics, magnetic interference can corrupt information. This is all similar to how atmospheric conditions affect data being transmitted from deep space back to earth. You may be wondering how all these means are still able to work so well despite our data being constantly corrupted by this **channel noise**. That is all thanks to error correcting codes.

Hamming's Solution: Parity

Hamming's solution to the punch hole problem was built upon the idea of **parity**, which we explored in chapter 7. We know that the parity of an object is either "even" or "odd". We are going to let 0 represent **even parity** and 1 represent **odd parity**. We will look at a string of **bits**, or 0's and 1's. We will count the amount of 1's in the string. If this number is even, we say the string is of even parity. If the number of 1's is odd, we will say the string is of odd parity. If the string is of even parity, we will append a 0 on the end and if the string is of odd parity, we will append a 1 on the end.

Definition 25.1.1

Parity Bit The bit that we append onto a string to denote the parity of the string.

Example 25.1.2

Parity Bit Say we have the string: 01000001

Notice this string has two 1's. Two is an "even number". This string therefore has even parity. Hence, it's **parity bit** will be 0.

Let's append this bit onto the end of the string: 01000001**0**

We just extended the length of our string by one bit by adding the parity bit, but the string now has enough information to detect one error or one "bit flip"!

Example 25.1.3

Parity Bit Say we want the computer at Bell Labratories to record an "A" on the punch card, or in binary ASCII representation: 01000001

Instead of telling the computer to punch the string as is, we are going to tell it to punch the string with it's appended parity bit: 010000010

Say the computer is having a bad day. It messes up an punches an extra hole in the third position. In other words, it flips the third bit. We now have: 011000010 punched on our card.

Example 25.1.3 (cont.)

Luckily, we have programmed the computer to check the parity of the string and compare it with the parity bit:

$0\mathbf{1}\mathbf{1}00000\mathbf{1}0$ We see that this string has three 1's. That is **odd parity**.

$010000001\mathbf{0}$ But the parity bit is of **even parity**

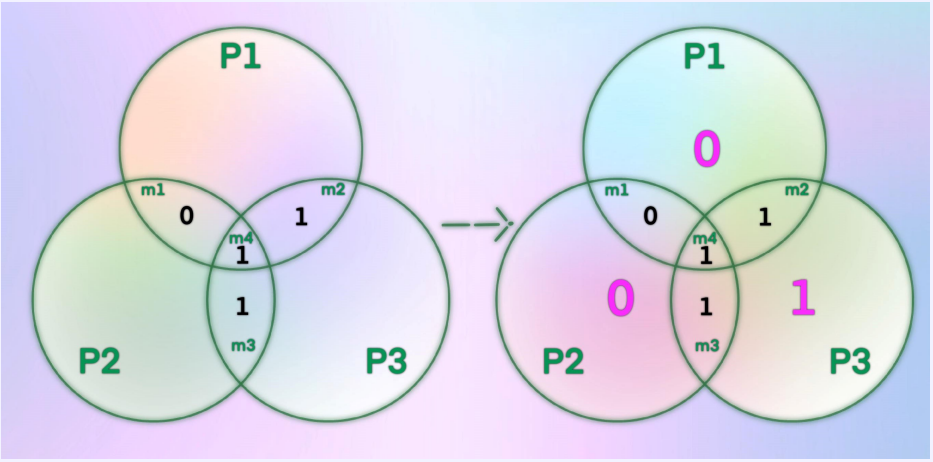
Since the parity of the string and the parity of the parity bit do not match, the computer knows that an error has occurred somewhere in the punching process!

While the process above is effective in detecting errors, it does not tell us where exactly the error occurred, which makes it difficult to correct our string back to it's intended form. This idea served as the basis for Hamming's final solution, which we now refer to as the "Hamming $(7,4)$-code". His idea was to take a string of length 4 and append three parity bits within the string. His process may be best shown by example:

Example 25.1.4

Hamming $(7,4)$-code. Say we have the string: 0111. Let's represent it as such: $(m_1\ m_2\ m_3\ m_4) = (0\ 1\ 1\ 1)$.

Now, consider the Venn diagram below. By placing the four bits into the intersections, we are able to calculate the 3 required parity bits.



We now put our string in the form: $(P_1\ m_1\ P_2\ m_2\ P_3\ m_3\ m_4)$: $(\mathbf{0}\ 0\ \mathbf{0}\ 1\ \mathbf{1}\ 1\ 1)$

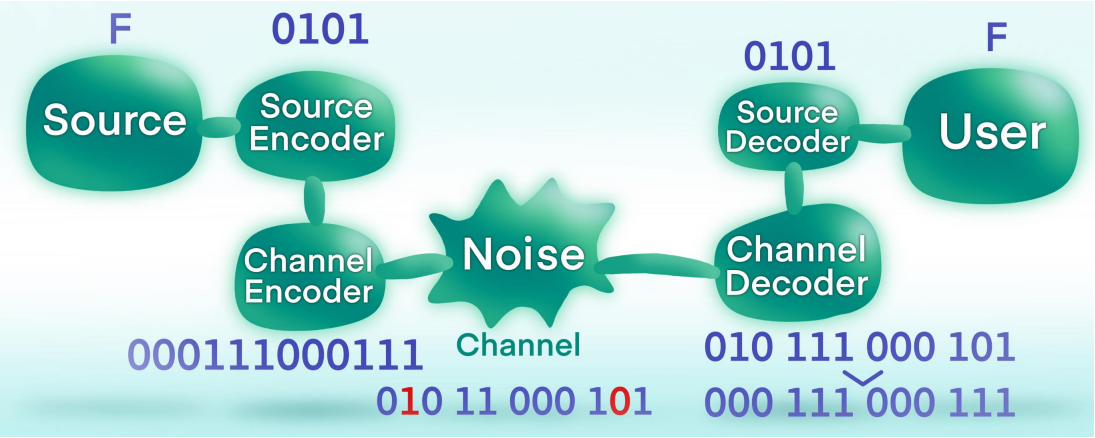
Example 25.1.4 (cont.)

We want the computer to punch our new length 7 string and then check the three parity bits. By doing so, the computer does not only detect that an error has occurred, but also has enough information to correct it.

Notice that our two techniques above required us to extend the length of our original string. Our parity bit technique for detecting errors required us to append an extra bit, turning our string of length 8 into a string of length 9. The Hamming $(7, 4)$-code required us to extend a string from length 4 to length 7. That is a lot of space used for error detection and correction! As we study Coding Theory further, we will want to study techniques that are both effective while still being slightly more space efficient. This idea of **message expansion** for the purposes of error detection and correction is the heart of coding theory!

The Father of Information Theory

This idea of message expansion is often referred to in coding theory as **redundancy**. Imagine you are at an awesome rock concert with your friend. You try to say something, but they didn't hear you the first time because there is simply too much noise. The solution is probably quite obvious: you just say it again. We are going to use this idea of repetition in our next example. Say we want to send the letter "F" and say that is represented as 0101 here. We are going to send each bit over the noisy channel three times. That way the decoder can analyze each of the three bits and take the best two. Consider the example in the diagram below:



We see that two of the bits were "flipped" by the noisy channel. Despite this, our decoder was able to uncover our original string and relay the correct information to our user. The diagram above is fundamental to coding theory and was first introduced in 1948 by mathematician Claude Shannon in his revolutionary paper, "A Mathematical Theory of Communication".

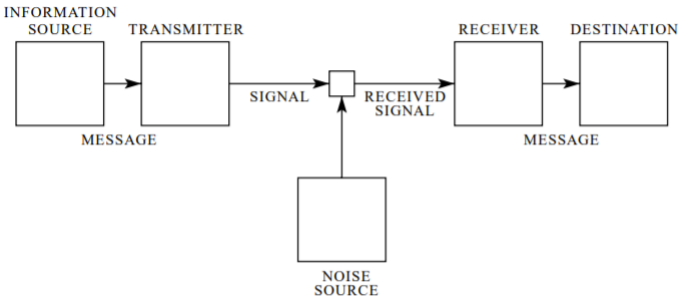
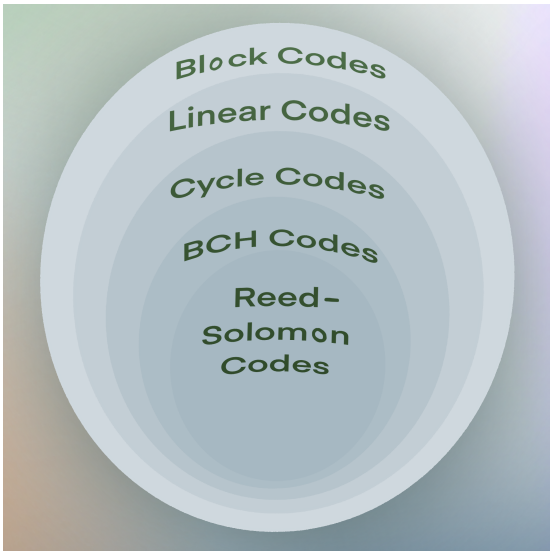


Figure 1: Taken from Shannon's 1948 paper

Claude Shannon, also known as "The Father of Information Theory" developed a mathematical theory behind digital communication, known as information theory. Note that the term coding theory and information theory are often used interchangeably and for our purposes, that is what we intend. Shannon did not invent the compact disc or any of the means of digital communication that we mentioned earlier, but his mathematical theory of communication forms the basis for all of them. Even though Shannon's work was published in the 1940's, it is more relevant today than ever. Shannon's paper promised a digital means to store and transfer data in the form of bits, exactly how it is done today. His promise would serve as a challenge for engineers. A challenge which would eventually become the reason we can download, upload, store, transfer, and playback digital data. Claude Shannon is considered one of the most influential minds of the twentieth century. If you would like to learn more about Shannon and his work, it is highly recommended that that you check out the 2018 documentary "The Bit Player".

CLICK HERE TO WATCH THE TRAILER

We will begin our study with a type of code called **block codes**, which are similar to the examples presented in this section. We will see that these do not always produce the most space efficient or effective decoding algorithms. By adding additional restraints onto our block codes and applying some results from linear algebra, we will define **linear codes**. The decoding algorithms we will present for linear codes, with emphasis on the **syndrome decoding algorithm**, are going to require us to recall many of our fundamental results on groups. Linear codes can be restricted even further into **cyclic codes**. The theory behind the structure of cyclic codes would require us to explore fundamental results of polynomial rings and galois fields and are therefore beyond the scope of this chapter. BCH codes, or Bose–Chaudhuri–Hocquenghem codes, which were invented independently in 1959 and in 1960 are further restricted cyclic codes. In 1960, Irving Reed and Gustave Solomon invented a class of codes which can be considered a BCH code with one



additional restriction on the algebra, known as Reed-Solomon codes. When studying coding theory, it can be helpful to begin with codes in the most general form and add restrictions as your study and abstract algebra knowledge progresses.

The answer to our introductory question about how our CD's, QR codes, computers, and deep space communications work despite all the errors is coding theory! We may have given this away at the beginning of the chapter, but hopefully now the answers to how this is all possible is less of a mystery. Today's CD player will be using two interwoven Reed-Solomon codes to play your favourite songs despite how dirty and scratched up your CD collection is. Error correcting codes are also built into your computers, specifically in the random access memories, as it is less expensive to correct errors than it is to build perfect circuits. In 1972, the Mariner space probe flew past Mars and transmitted pictures through noisy solar activity and Earth's atmosphere using a special type of linear code called a **Reed Muller code**. The pictures were composed of pixels ranked by 62 levels of grayscale encoded into binary strings of length 6. The strings were expanded into length 32 before transmission to compensate for expected errors! In 1979, the Voyager probes transmitted color pictures of Jupiter using another spacial type of linear codes called **Golay codes**.

CLICK HERE TO SEE THE PICTURES