Combinatorial Constructions in Error-Correcting Codes

Howard Qian

December 16, 2024

1. Introduction

Through the transmission or storage of data, bits in data can easily become corrupted due to transmission channel noise, distortion, interference, and even the smallest of errors can distort the information. Although these errors are able to be detected by EDCs (Error Detection Codes), they are unable to be fixed, thus rendering the transmitted data unreliable and encouraging miscommunication and system malfunctions. To address this, error-correcting codes were developed, with the ability not only to locate transmission errors but also to correct them, enabling dependable communication even over imperfect channels.

In this project, I will first examine several several simple linear codes in [§1], with important definitions explained in [§2]. Next, I will explore the intuition behind them, and their clever connections to combinatorial game theory, including impartial games such as NIM in [§3] and finally, Turning Turtles in [§4].

1.1. Motivations

Suppose than Alice and Bob are trying to send a message to each other, using a series of 0's and 1's. However, one bit, or digit, in the message may be corrupted, or flipped. More specifically, say that Alice wants to send a message to Bob, telling him that she is doing well; so, she sends the message 10100101. However, along the way, the third bit gets corrupted, turning from a 1 to a 0. Her message then becomes 10000101. Now, when Bob receives this message, the meaning of the message has changed, with the altering of this one bit, and he is left confused and baffled.

In order to solve this detrimental problem, we will explore progression of error-correcting codes.

We will first take a look at the most basic type of error-correcting code. In this code, every bit in a message is sent three times. For instance, if the original message is 10110, then the message sent will be 111000111111000. Thus, if a bit is flipped, we are able to compare every three bits of the message and just take the majority bit. For example, if we wished to send the message 10, then we would actually send 111000. Say the message is corrupted in a way where the third bit gets flipped; the message becomes 110000. Comparing every three digits, we know that the third bit must be the corrupted bit, as 1 is the majority in those three digits.

However, although this solution works, it is super inefficient, as we must send $3n$ bits instead of the original n bits.

Another error-correcting code works by appending, or adding, a parity-check digit to the end of the message. In this code, we must send the original message twice, then add on the parity-check digit. This parity-check digit is either 1, if the intended message has an odd number of 1's, or 0 if the intended message has an even number of 1's. Let's use the example of Alice and Bob again. Say that Alice intends to send the message 1011 to Bob. The actual message sent would then be 101110111. The parity-check digit, in this case, is 1 because there is an odd number of 1s in the original message 1011. When Bob receives this message, he reads the message to be 100110111. With this code, Bob now knows that there must be an error in one of the third digits, so the message is either 1011 or 1001. This is where the parity-check digit comes into play - since the last digit is a 1, Bob knows that the intended message has an odd number of 1's, and so he deduces that the correct message is 1011. The issue that this method requires sending $2n + 1$ bits if the original message was n bits, which is still inefficient.

2. Terminology

Definition 2.1 (Codes and Codewords). A *codeword* is the transmitted message and the set of all codewords is called a *code*.

Definition 2.2. The number of digits in a message is known as the length of the message, and is labeled with the symbol n. Each message is represented in a certain base, which dictates the number of unique digits used to represent a message, and the base is labeled with the symbol B.

For every message with length n digits, assuming that only one digit is flipped (corrupted), there are $n + 1$ possible received messages. Since the length of each message is n, and each digit can be either 0 or 1, there are a total of 2^n messages that can be sent. Thus, there can only be at most $\frac{2^n}{n+1}$ codewords.

Definition 2.3 (Perfect Code). If, for a code of length n correcting at most k errors, there is exactly one code word for every string such that they differ in at most k places, then the code is said to be a perfect code.

For our case where $k = 1$, $n + 1$ must evenly divide 2^n to achieve a perfect code. This leads us to see that $n + 1$ must be a factor of 2^n, or that $n + 1$ must be a power of 2.

Definition 2.4 (Hamming distance). The *Hamming distance* between two codewords a and b where $a, b \in C$ is the number of positions, or digits, at which a and b differ. The Hamming distance then, for code C, is the minimum Hamming distance between two distinct codewords.

As an example, let codeword $a = 101100$ and let codeword $b = 100001$. Then, the Hamming distance between the two codewords is 3.

Since a corrupted message can only correspond to one specific codeword, Hamming distances of 1 and 2 between codewords are not possible. For instance, let's say codewords $a = 0000000$ and $b = 0000011$. If we were given the message 0000001, is this 0000001 with an error in the seventh position, or sixth position? Thus, the minimum Hamming distance is at least 3.

3. Hamming Codes, Nim, and Game Theory

We now introduce a new group of codes called the *Hamming codes*, whose codewords have a Hamming distance of at least 3 and can correct up to 1 flipped bit. A *Hamming code* of length $2^n - 1$ is called a *perfect Hamming code.*

In order to generate a perfect code (see Definition 2.3) of 16 codewords of length $7 = (2^3 - 1)$, we use a greedy algorithm. This greedy algorithm takes the lexicographically first string, then writes down strings of length 7 which each differ from the previous ones in at least three positions. Here are the results:

0000000	0000111	0011001	0011110
0101010	0101101	0110011	0110100
1001011	1001100	1010010	1010101
1100001	1100110	1111000	1111111

This array of strings relates to Nim, or more specifically, binary Nim.

Definition 3.1. Nim is an *impartial game*, which are games where both players have the same moves available to them. On the other hand, games like CHESS are considered *partizan games*, which are games where different players have different moves. The game of Nim involves two players, where both players take turns moving. There are several piles of stones, and a move consists of removing some of the stones from a single pile. The player who removes the last stone is the winner.

Interestingly, Nim games are able to be represented using a string of 0's and 1's. In binary numbers $A = \dots a_3a_2a_1$, where $n \in \{1, 2, 3 \dots\}$ and $a_n \in \{0, 1\}$, the digits with $a_n = 1$ represents piles of stones with size a_n. As an example, the binary number 1000101 represents the Nim game with piles of size 1, 3, and 7. We then remove any leading zeros.

In Nim, players are allowed to make two different types of moves: they either remove all of the stones from a pile, or they remove some of the stones. Let's focus on each case individually:

- When a player removes all the stones from a pile, we just flip the bit in the position corresponding to the pile's size from a 1 to a 0.

- When a player removes some (not all) of the stones from a pile, we flip the bit in the position corresponding to the pile's size. Then, we flip the bit in the position corresponding to the pile's new size. For instance, let's say we have the game 1001010, and we want to remove 3 stones from the pile with 7 stones. We would then have the game 0000010, or 10.

Notice that we produced a second pile of size 4, but then we immediately removed both of them, leaving 0 piles of size 4. More generally, we are always allowed to remove two piles of the same size. The idea is that whenever one player plays a move in one of the piles, the other player mirrors the first player's move in the other pile, and continues to do so, until the first player takes the last stone from the pile, in which case the other player then removes the last stone from the other pile. Thus, having two piles of the same size does not change the outcome of the game whatsoever.

By viewing the binary strings in the frame of Nim, we are able to arrive at this astonishing conclusion:

Theorem 3.2. *The codewords in a Hamming code are the $\mathcal{P}$ positions in the binary Nim game.*

In order to prove this, we will rely on the partition theorem, which is necessary to determine the $\mathcal{P}$ positions of impartial games, and is central to the theory of impartial games. The partition theorem is defined and proved in [Rub].

Proof: Let $\mathscr{P}$ be the set of codewords, which have a Hamming distance of at least 3, and we let $\mathscr{N}$ be the set of binary strings that are not the codewords. We need to show that, for every $G \in \mathscr{N}$, there is a move to a game in $\mathscr{P}$; and for every $G \in \mathscr{P}$, every move is to a binary string in $\mathscr{N}$.

- Every element in the set $\mathscr{N}$ differs by either 1 or 2 digits from an element in $\mathscr{P}$, by the definition of a perfect code and codewords. Thus, it is possible for an element in $\mathscr{N}$ to be changed to an element in $\mathscr{P}$ per the rules of binary Nim.

- All the elements in $\mathscr{P}$ are codewords, meaning that each element differs from another element by at least 3 digits. The rules of binary Nim dictate that only at most 2 digits can be changed in a move, and thus it is impossible for an element in $\mathscr{P}$ to go to another element in $\mathscr{P}$, meaning that the only possible move from $\mathscr{P}$ is to $\mathscr{N}$.

The $\mathscr{P}$ positions and $\mathscr{N}$ positions in the partition theorem are just the $\mathcal{P}$ and $\mathcal{N}$ positions of the Nim game, respectively. ■

3.1. Nim Sum and Grundy Values

In order to evaluate a game of Nim in general, the *num sum* operation must first be understood. Denoted by the symbol $\oplus$, it is known by the **xor** operation in computer programming. In order to compute the nim sum of two nonnegative integers a and b, we must first write a and b in binary, then add them without carrying. For instance, $5 \oplus 9$ is equal to 12:

$$\begin{array}{cccccc}
5 & & 0 & 1 & 0 & 1 \\
9 & \oplus & 1 & 0 & 0 & 1 \\
\hline
12 & & 1 & 1 & 0 & 0
\end{array}$$

We can also apply the nim sum operation to more than two numbers:

$$\begin{array}{ccccc}
4 & & 1 & 0 & 0 \\
3 & & 0 & 1 & 1 \\
1 & \oplus & 0 & 0 & 1 \\
\hline
6 & & 1 & 1 & 0
\end{array}$$

Remark 3.3. Notice that the resulting digit is 0 if the number of 1's in a column is even, and the resulting digit is 1 if the number of 1's in a column is odd.

Remark 3.4. Also, notice that the nim sum of numbers $a_1, a_2, \cdots a_k$, $a_1 \oplus a_2 \oplus \cdots \oplus a_k$, is always less than or equal to the sum of the numbers, $a_1 + a_2 + \cdots + a_k$, or symbolically, $a_1 \oplus a_2 \oplus \cdots \oplus a_k \leq a_1 + a_2 + \cdots + a_k$.

One of the properties of nim sum is that $a \oplus a = 0$, which we have already seen in the previous example of nullifying two piles with the same number of stones. Since the nim sum of the two equal numbers is 0, this means that two piles of the same number of stones would not affect the overall value of the binary nim position.

Theorem 3.5. [Bou02] *The Nim position with piles of size $a_1, \cdots, a_n$ is a $\mathcal{P}$ position if and only if $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$.*

Example 3.6. 1001011 is a $\mathcal{P}$ position since $1 \oplus 2 \oplus 4 \oplus 7 = 0$:

$$\begin{array}{ccccc}
1 & & 0 & 0 & 1 \\
2 & & 0 & 1 & 0 \\
4 & & 1 & 0 & 0 \\
7 & \oplus & 1 & 1 & 1 \\
\hline
0 & & 0 & 0 & 0
\end{array}$$

Indeed, for each of the 16 codewords listed above, the nim sum of the pile sizes is equal to 0.

Theorem 3.7. *Let C be the Hamming code of $\mathcal{P}$ positions in Nim, constructed in §3. Adding two code words $c, d \in C$ without carrying results in another element of C.*

Proof: Let the codeword c be the Nim game with piles of size $a_1, a_2, \cdots, a_n$, and codeword d be the Nim game with piles of size $b_1, b_2, \cdots, b_k$. The addition of codewords c and d is equal to $c \oplus d = (a_1 \oplus a_2 \oplus \cdots \oplus a_n) \oplus (b_1 \oplus b_2 \oplus \cdots \oplus b_k)$. Because $c, d \in C$, $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ and $b_1 \oplus b_2 \oplus \cdots \oplus b_k = 0$, by Theorem 3.5. Thus, $c \oplus d = (a_1 \oplus a_2 \oplus \cdots \oplus a_n) \oplus (b_1 \oplus b_2 \oplus \cdots \oplus b_k) = 0 \oplus 0 = 0$, which indicates a $\mathcal{P}$ position, again by Theorem 3.5. ■

The code mentioned previously, with a Hamming distance of 3, can correct up to 1 error. We will look at a more general idea about codes.

Theorem 3.8. *If C is a code with Hamming distance d, then it is possible to correct $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors.*

Proof: We will first look at the case where d is a positive, odd integer, then the case where d is a positive, even integer.

- Say that the code C is able to correct $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors. Since d is odd, $\left\lfloor \frac{d-1}{2} \right\rfloor = \frac{d-1}{2}$. Then, any message differs from some codeword A by $\frac{d-1}{2}$ digits. Also, it must then differ from some other codeword B by at least $\frac{d-1}{2} + 1$ digits. Thus, the minimum distance between two codewords, or the Hamming distance, is $\frac{d-1}{2} + 1 + \frac{d-1}{2} = d$.

- Again, say that the code C is able to correct $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors. Since d is even, $\left\lfloor \frac{d-1}{2} \right\rfloor = \frac{d}{2} - 1$. Then, any message differs from some codeword A by $\frac{d}{2} - 1$ digits, and thus, it must differ from some other codeword B by at least $\frac{d}{2} + 1$ digits. Notice that the message cannot differ from codeword B by $\frac{d}{2}$ digits because then it would differ from another codeword also by $\frac{d}{2}$ digits, meaning that both codewords could match to the same message, preventing error correction. Finally, the minimum distance between two codewords is then $\frac{d}{2} - 1 + \frac{d}{2} + 1 = d$.

Additionally, considering just the case where d is odd, notice that if C is a code with Hamming distance d, then any message must differ from some codeword A by at most $\frac{d-1}{2}$ positions and from another codeword B by at least $\frac{d-1}{2} + 1$ positions. In this case, a message is able to be matched to A without confusion for codeword B, proving that the code can correct up to $\frac{d-1}{2}$ errors. ■

The idea that, for even values of d, a code with Hamming distance d is unable to correct a message if that message differs from some codeword in $\frac{d}{2}$ positions leads to another theorem:

Theorem 3.9. *If C is a code with Hamming distance d, where d is a positive, even integer, then the code can simultaneously correct $\frac{d-2}{2}$ errors and detect $\frac{d}{2}$ errors.*

To see a geometric-based proof for Theorem 3.8, see [MS77].

4. Turning Turtles

Definition 4.1. The game of *Turning Turtles* is an impartial game. There are n turtles in a row, some of which are on their backs (upside down) and the others are rightside up. A player, on their turn, chooses a positive integer k and must flip at least 1 turtle and at most k turtles, guaranteeing that the leftmost turtle flipped goes from being upside down to rightside up.

Theorem 4.2. *Nim is the game of Turning Turtles with $k = 2$.*

Proof: For Turning Turtles with $k = 2$, a player can either flip over 1 or 2 turtles. Consider that an upside down turtle is equivalent to a 0 digit in Nim, while a rightside up turtle is a 1 digit. Flipping over 1 turtle, with that turtle being originally upside down, is equivalent to flipping a single digit from a 1 to a 0, similar to the first of the two legal moves in binary Nim. Flipping over 2 turtles is equivalent to removing some of the stones from a larger pile, consequently changing the digit corresponding to the pile from a 1 to a 0 and changing the digit in the position corresponding to the remaining pile size (see §3). ■

Additionally, the $\mathcal{P}$ positions of the Turning Turtles game with $k = 6$ and $n = 23$ forms the famous *binary Golay code*, a perfect code that corrects 3 errors in strings of length 23.

References

[Bou02] C. L. Bouton. Nim, a game with a complete mathematical theory. *Ann. of Math (2)*, 3(1-4): 35–39, 1901/02.

[MS77] F. MacWilliams and N. Sloane. *The Theory of Error-Correcting Codes.* Vol. 16. North-Holland Publishing Company, 1977, pp. 10–11.

[Rub] S. Rubinstein-Salzedo. *Combinatorial Game Theory.* Chapters 3 and 4.