COMBINATORIAL CONSTRUCTIONS AND ERROR CORRECTING CODES

SAMBHU GANESAN, JAYADEV GHANTA, ISAAC SUN

1. Introduction

A challenge arises as the world uses more data: How can we transport the data? Transferring data is challenging. Often, bits of data get altered, which can completely change the meaning of the data. To overcome this issue, we will introduce some redundancy into the data. This ensures that even if some bits are lost, the message's meaning isn't lost. For example, the simplest way to introduce redundancy to the data is to repeat each bit k times. We could construct a set like

$$[e_1, e_2, \dots e_n, e_1, e_2, \dots e_n, \dots e_1, e_2, \dots e_n]$$

or

$$[e_1, e_1, \dots e_1, e_2, e_2, \dots e_2, \dots e_n, e_n, \dots e_n].$$

The first scenario is more robust to multiple runs and hence is used in practice. This is the basis on which the theory of error-correcting codes arises. [6]

In this paper, we will go over the construction of Hamming codes and their relationship to Nim. Then we introduce Lexicodes and generalize the results from the game of Nim to any heap game. Finally, we will shift our focus from Nim to Annihilation Games and Anncodes and explore the relationships between them error-correcting codes.

2. suggestions

give me more sugegestions rto add

3. Hamming Code: Encoding, Error Detection, and Correction

The Hamming code process begins with defining two key matrices: G, the generator matrix, and H, the parity-check matrix. For the $[7, 4]$ Hamming code, G encodes 4-bit messages into 7-bit codewords, while H ensures error detection and correction. [2]

Definition 3.1. In this setup, H is a 7×3 matrix, constructed with each row as a unique binary representation:

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

Definition 3.2. The relationship $G \cdot H = 0$ ensures the code's structure, meaning that any valid codeword, when multiplied by H, yields a result of zero. From this, G is derived to encode the input message:

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

This generator matrix maps each 4-bit message into a unique 7-bit codeword by appending parity bits calculated as linear combinations of the original message bits. [7]

Example 3.3. To demonstrate, let's encode a message $m = [1, 0, 1, 1]$. Using G, the codeword c is calculated as:

$$c = m \cdot G = [1, 0, 1, 1] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Performing the matrix multiplication gives:

$$c = [1, 0, 1, 1, 0, 1, 0].$$

Definition 3.4. This 7-bit codeword is transmitted. However, during transmission, a single-bit error may occur. To identify and correct this, the parity-check matrix H is applied to the received message to compute the syndrome s:

$$s = H \cdot m.$$

Example 3.5. Suppose the received message $m = [1, 0, 1, 0, 0, 1, 0]$ differs from c, with an error in the 4th bit. Using our H, the calculation involves multiplying each row of H with the corresponding bits of m (mod 2):

$$s = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}.$$

The result $s = [0, 0, 1]$ indicates that the received message does not satisfy the parity-check constraints. The syndrome corresponds to the 4th row of H, identifying the position of the error. This process works because H was designed such that each row uniquely identifies one bit in the codeword.

Once the error is located, the 4th bit of m is flipped, correcting the error and yielding the original codeword $c = [1, 0, 1, 1, 0, 1, 0]$.

Definition 3.6. The key idea here is that the syndrome s reflects the error pattern. Mathematically, if c is the original codeword and e represents the error vector (a binary vector with a single 1 at the position of the error), then:

$$m = c + e.$$

Applying H:

$$s = H \cdot m = H \cdot (c + e) = H \cdot c + H \cdot e.$$

Since $H \cdot c = 0$ for valid codewords, we have:

$$s = H \cdot e.$$

Thus, s directly identifies the column of H corresponding to the error, enabling its correction.

By combining the encoding process, error detection through the syndrome, and error correction via flipping the identified bit, the Hamming code provides a robust method for single-bit error correction. [1]

4. Hamming Codes and their Relation to Nim

The Hamming code has many connections to the game of Nim. Before we go into the content, let's go over an important theorem.

Theorem 4.1. *A code corrects up to e errors if and only if its Hamming distance is greater than or equal to $2e + 1$.*

Proof. We can prove this in both directions. Let X and Y be any two codewords from this code.

We will prove the forward direction first. Let the code have a Hamming distance that is greater than or equal to $2e+1$. Then we know that X and Y differ in at least $2e+1$ spots. Thus any message that differs from X in at most e spots, differs from Y is at most $e + 1$ spots. So if the message was interpreted as X then it can't be interpreted as Y and vice versa.

Now we will show the other direction. Let the code be an e-error-correcting code. So the message that is interpreted as X can differ in at most e spots from X. Also, note that any message that is interpreted as Y differs from X in at least $e+1$ positions. Hence X and Y must differ in at least $2e + 1$. [5] □

This is an important result to keep in mind and we will use this for the proofs below. With this let's take a look at some definitions that will better help us understand the relationship between Hamming Codes and the game of Nim.

Definition 4.2. The game of Nim is a classical two-player game. In any position of Nim, there are multiple piles of stones and each player takes turns removing the stones. The first player without any legal moves is the loser.

Definition 4.3. Let $*n$ denote a Nim pile of size n.

Definition 4.4. Consider the Nim game $*a_1 + *a_2 + \cdots + *a_k$ where $a_k \geq a_{k-1} \geq \cdots \geq a_1$ and $a_i \in \mathbf{N}$. Take an array of size a_k populated with 0's. Then for each a_i, we flip the bit at index $a_k - a_i$. We will call this the *binNim*.

Example 4.5. The *binNim* of the game $*3 + *7 + *2 + *1 + *1 + *5$ is 1010100.

We have an interesting connection to the game of Nim using this concept.

Theorem 4.6. *The codewords in Hamming code are the P-positions when represented as a binNim string.*

In other words, we are claiming that any Hamming code of length n is a P position.

Proof. Let $\mathcal{P}$ be the set of all codewords written as a binNim string and let $\mathcal{N}$ be the set of all other messages. Note that messages in $\mathcal{N}$ differ from some codeword in less than 3 positions. This means we can go from the set of all messages excluding codewords in $\mathcal{N}$ to an element in $\mathcal{P}$.

Now we consider the set $\mathcal{P}$ of codewords. Since $\mathcal{P}$ is a set of codewords we know that it differs from another codeword in at least 3 positions. Also, note that changing for greater than or equal to 3 positions is not possible with the Nim moves. Hence we see that a move can only flip 1 or 2 of the bits. Therefore, we can go from the set of all codewords $\mathcal{P}$ to the set of all other messages $\mathcal{N}$.

By the partition theorem, we can see that $\mathcal{P}$ and $\mathcal{N}$ are just the P and N positions respectively. □

We encourage the reader to find a proof using the generator concept introduced above as a fun challenge.

5. Lexicodes

Definition 5.1. Lexicodes are a type of error-correcting code that helps detect and fix mistakes in data transmission. They use a greedy algorithm involving lexicographical ordering that ensures no two codewords are too similar. [3]

Definition 5.2. The Hamming distance of two codewords is the number of positions in which they differ. A code's Hamming distance is the minimal distance between any two of its codewords.

Construction 5.3. To construct a lexicode, begin by ordering all possible binary strings in lexicographic order, such as `0000`, `0001`, `0010`, and so on. Start with the first string `0000` and add it as the first codeword. Filter the rest of the list, selecting only those strings that are at least the required Hamming distance (e.g., 3) from every codeword already selected. Repeat this process until all valid strings are chosen.

Example 5.4. For example, suppose we're building a lexicode with a minimum Hamming distance of 3 using binary strings of length 4. After `0000`, the next valid string is `0110`, which is 3 units away from `0000`, so it is added. Continuing this process, we select strings like `1001`, `1111`, and so on. The final lexicode for this case would be {`0000, 0110, 1001, 1111`}.

Remark 5.5. This approach works as it makes sure that codewords are sufficiently spaced to detect and correct errors. If a codeword is altered slightly, it would not match any valid codeword, allowing the error to be obvious to detect. When the error is small, the receiver can determine the original codeword by finding the nearest valid one.

6. Lexicodes and their relation to heap games

Once again as last time, we will start with some definitions.

Definition 6.1. A turning set is a general form of $\{a_0, a_1, \dots a_n\}$ where $a_0 > a_1 > \cdots > a_n \geq 0$.

Definition 6.2. The base of a code is the number of unique digits used to represent messages and is usually denoted by B.

Definition 6.3. A *heap game* is an impartial game with several heaps of stones. We can use turning sets to describe the play of this game. A player may replace the pile of size a_0 with a series of piles of size $a_1, a_2, \dots a_n$.

In particular, we see that Nim, a heap game, has a turning set of $\{a_0, a_1\}$.

With the two definitions above we can write a relationship between the P-positions in heap games to lexicodes.

Theorem 6.4. *The P positions of any heap game correspond to codewords in lexicode for any turning set and base.*

Proof. We can do an inductive proof by the positions of the game. Consider a position X. Now we will consider two cases, the first being if X is not in the lexicode and the second being if X is in the lexicode.

If X is not in the lexicode this means that X' is in the lexicode where $X' < X$ and $\{X, X'\}$ is a turning set. By the inductive hypothesis, a move from X to X' is a winning move, and therefore G is not a winning position.

If X is in the lexicode and X to X' is a legal move, then we must have $X' < X$ and the turning set $\{X, X'\}$ exists. But since we have assumed that X is a winning position, we need to reject each X' which means the move from X to X' is not a winning move. Therefore X is a winning position. □

With this, we have shown a relationship between all Heap games and error-correcting codes.

7. Annihilation Games

Now we will shift our attention to a new type of game called Annihilation Games.

Definition 7.1. Annihilation games are played on finite directed graphs called ground graphs (Γ) where tokens are distributed across the vertices of the graph. Players can spend a move moving a token of their choice across a directed edge.

(1) If the destination vertex is already occupied by another token, both tokens are removed from the game.
(2) If the destination vertex is unoccupied, the token moves there.
(3) A player loses if they cannot make a move on their turn.

Example 7.2.

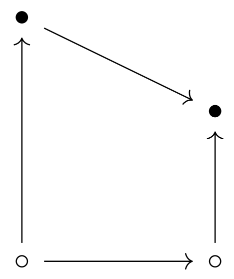

Figure 1. N-Position. Left moves top left token to the top right vertex and annihilates both tokens. Then Right has no moves left allowing Left to win the game.

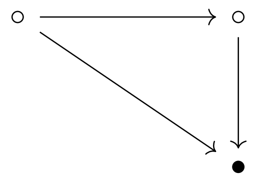

Figure 2. P-Position. Left runs out of moves as he cannot move the token to any position.

Definition 7.3. Each annihilation game can be represented as a binary vector where each bit corresponds to a vertex in a graph. When the bit is `1`, it represents an occupied vertex and when the bit is `0`, it represents an empty vertex.

Definition 7.4. The set of P-positions in the game form collections of binary vectors that satisfy the ruleset. These vectors are then used to form the codewords in an anncode.

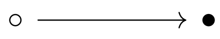

Figure 3. This game can be represented through the state (0, 1).

Example 7.5.

Definition 7.6. The binary vectors representing P-positions are used as codewords for the anncodes because P-positions form a linear subspace in the vector space over $GF(2)$. This means the XOR (bitwise addition modulo 2) of any two P-position vectors is also a P-position, and the zero vector $(0, 0, \ldots, 0)$, representing no tokens on any vertex, is always a P-position.

(1) The game rules impose constraints on P-positions, ensuring that any two P-position vectors are sufficiently far apart in terms of Hamming distance. The Hamming distance is defined as the number of positions where two vectors differ.
(2) A larger minimum Hamming distance improves the error detection and correction capabilities of the code.

Example 7.7. For example, if $(1, 0, 1, 0)$ and $(0, 1, 1, 0)$ are P-positions, their XOR, $(1 \oplus 0, 0 \oplus 1, 1 \oplus 1, 0 \oplus 0) = (1, 1, 0, 0)$, is also a P-position. More specifically, here is how it is formally derived.

Definition 7.8. Let V be a vector space over GF(2), and let $g : V \to \mathrm{GF}(2)^t$ be a function satisfying the properties of the Sprague–Grundy function. The kernel of g, defined as $V_0 = \{u \in V : g(u) = 0\}$, forms a linear subspace of V. The function g is a homomorphism, mapping V onto $\mathrm{GF}(2)^t$, where $t = \dim V - \dim V_0$. Consequently, V/V_0 is a quotient space consisting of 2^t cosets, where each coset $V_i = u \oplus V_0$ represents positions in V that share the same g-value.[4]

Remark 7.9. The subspace V_0 contains all P-positions, establishing a direct connection with the positions of the annihilation game and the algebra from V.

Theorem 7.10. *Let G be an annihilation graph. Every G has a function γ (the generalized Sprague-Grundy function).*

Proof. For the proof, see [Fraenkel 1996, p. 20] and [Fraenkel and Yesha 1986]. □

Definition 7.11. The Sprague-Grundy function assigns a nonnegative integer (Grundy value) to each position in a combinatorial game which helps identify the winning strategies. It is defined as

$$g(u) = \operatorname{mex}\{g(v) : v \in F(u)\},$$

where $F(u)$ is the set of positions reachable from u. A position $g(u) = 0$ is a losing position (P-position), while $g(u) > 0$ indicates a winning position (N-position) for the current player.

Though beyond the scope of this paper, this function and results from a study of lexicodes can help us evaluate anncodes. We will direct the interested reader to read [Fraenkel, pg 420 - 430].

8. Conclusion

In this paper, we investigated the connections between Nim and in general heap games and Hamming codes and Lexicographic codes. We also introduced the idea of an annihilation game and showed the relationships to Hamming distances.

9. Acknowledgements

We would like to thank Dr. Simon Rubinstein Salzedo for teaching us the concepts related to this project and giving us the opportunity to do this research project. We would also like to thank our TAs for all their comments and inputs.

References

References

[1] Mark Bun. *Error-Correcting Codes, Linear Codes, Lecture 19.* Lecture notes from CAS CS 599 B: Mathematical Methods for TCS, Spring 2022. 2022.

[2] Eshan Chattopadhyay and Surendra Ghentiyala. *Fundamentals of Error Correcting Codes, Lecture 8: Sep 15, 2022.* Lecture notes from CS 6815: Pseudorandomness and Combinatorial Constructions, Fall 2022. 2022.

[3] John H. Conway and N.J.A. Sloane. “Lexicographic Codes: Error-Correcting Codes from Game Theory”. In: *IEEE Transactions on Information Theory* IT-32.3 (1986), pp. 337–343.

[4] Aviezri S. Fraenkel. *Error-Correcting Codes Derived from Combinatorial Games.* MSRI Publications, Volume 29, 1996.

[5] Yuvraj Sarda. “Combinatorial constructions for error-correcting codes”. In: *An expositionary paper in Combinatorial Game Theory & Coding Theory* (2021). Paper from Euler Circle.

[6] Madhu Sudan and Piotr Mitros. *Essential Coding Theory, Lecture 1: Introduction to Error-Correcting Codes.* Lecture notes from 6.895 Essential Coding Theory, September 8, 2004. 2004.

[7] Haotian Yu. *Lecture Notes on Coding Theory, COS 521, Fall 2023.* Lecture notes from COS 521: Advanced Algorithm Design, Princeton University, Fall 2023. 2023.

Email address: `sambhu.ganesan150@gmail.com, jayadevgh@gmail.com`