Access*: Interdisciplinary Journal of Student Research and Scholarship

Volume 4 | Issue 1 Article 6

2020

Hamming Codes

Steve Mwangi
University of Washington, Tacoma, xenios@uw.edu

Sterling Quinn
University of Washington, Tacoma, stelrq@uw.edu

Abstract

We will examine the application of Matrix Algebra in forming Hamming Codes. Hamming Codes are essential not just in the detection of errors but also in the linear concurrent correction of these errors. The matrices we will use have entries that are binary units. Binary units are mathematically convenient, and their simplicity permits the representation of many open and closed circuits used in communication systems. The entries in the matrices will represent a message that is meant for transmission or reception, akin to the contemporary application of Hamming Codes in wireless communication. We will use Hamming Codes (7,4), which are linear subspaces of the 7-dimensional vector space over F_2, the base field.

Hamming Codes

Hamming Codes were named after the Mathematician Richard Wesley Hamming. Hamming, who made several contributions in Computing, Engineering and Telecommunications, realized that the detection of errors in the transmission of information, in and of itself, was not adequate. The correction of such errors was also an integral part of the transmission of the information (Hamming 2). Take for instance, the following text transmission:

Message to be transmitted: "Mr. Bob, your bags arrive at 11:00am."

Message that is processed: "Mr. Rob, your rags #$%^^& at _1:00am."

After transmission, the message transmitted in the above example will be compromised. At the time Mr. Hamming was working on problems like this, there were no error correcting codes (Hamming). As such, it was essential for the correction of codes to occur in conjunction with the detection of errors. Hence, Mr. Hamming created the codes that bear his name—Hamming Codes.

In this paper, we will assume that the standard unit of all information represented in machines is a binary digit or bit. This will enable us to employ bits as the standard unit for encoding digital information (Hamming).

Description of Terms

To begin with, we will explain the meaning of some of the terminology that we frequently use across this paper. This includes:

1. *Redundancy* is the term we will employ to calculate the efficiency of our hamming code. To determine *redundancy*, we will need to figure out how many bits to use for the entire information to be transmitted. This will be the sum of:

   $$I2 + E2$$

   where:

   I_2 = Nº bits representing information, and

   E_2 = Nº bits representing error detecting and correcting code(E_2).

   Thus, the efficiency of our hamming code is given by:

   $Redundancy = (I_2 + E_2) \div I_2$

2. *Parity Bits* (and *parity check*): These are bits added to a set of data bits. For instance, in *Single-error detecting codes*, if we have *n* bits that represent the information, we can use ($n+1$) bits for transmission. The added bit is the parity bit that helps detect an error. If the bit set containing *n* bits has an *even* occurrence of 1's, we have even parity and the added bit is 0. If we have an odd number of 1's, we have odd parity, and the added bit is 1. The data bits plus the parity bit will always have even parity (Hamming).
3. *Parity Check:* This is the method that checks the parity bits mentioned above. So, if information transmitted employed even parity, and the information received has an odd number of 1's, we know that there is an error (Hamming 3).

4. *Hamming (n, k) Code:* This is a Hamming Code Block that has *k* (= I_2) information digits, and *n* as the block length or total number of bits, including parity bits. Hence, *n = p + k*, where *p = parity bits*. If there is one error, then there are n possible locations for that error. So, $2^p = 2^{n-k}$, must be at least n+1 (with n possible errors + 1 extra for no errors). Therefore, n, the total block length, must be at most 2^p-1. Which also translates to k, the number of information bits, being at most 2^p-p-1, where p = n - k (Moon, 2005).

Parity Checking Bits

Hamming Codes' strength lies in their ability to detect and correct errors with a relatively low redundancy. For this paper, we will focus on the (7,4) Hamming Code. This code has a redundancy of 1.75, which we calculate following the formula in definition one, *Redundancy* = $(I_2 + E_2) \div I_2$. As an example, let us consider the data:

1 0 0 1

This is a Hamming Binary Code block with $k = 4$. The data we will transmit will consist of 7 bits, or n = 7.

Now we can take:

1. Take all bit positions that are powers of 2 and mark them as parity bits. Such bits would be at positions 1, 2, and 4.
2. All other bit positions will contain our data (i.e. bits at positions 3, 5, 6 and 7).
3. Now, each parity bit will be used to determine the parity for sets of bits in the code word

(Downey, 2018).

The position of the parity bit will help us know which sets of bits to check e.g.

Parity bits at Position	Sequence to check bits	Set of bit positions to check
1	check 1 bit, skip 1 bit…	{1,3,5,7}
2	check 2 bits, skip 2 bits…	{2,3,6,7}
4	check 4 bits, skip 4 bits…	{4,5,6,7}

Table 1. The following table describes, for each parity bit, its position and which bits are used to determine even parity for that bit (Downey, 2018)

4. Finally, the parity bit is set to 1 if the total number of ones in the positions it is checking, other than itself, is odd—even parity. Otherwise the parity bit is 0 if the total number of ones in the positions it checks is even (Downey, 2018).

To transmit our message, we now need to determine value for the parity bits at position:

_ _1_001

Determining parity bits:

Parity bits at Position	**Sequence to check bits**	**Parity?**	**Determine parity bit(?)**	**Set parity bit**
1	checks bits 1,3,5,7 **? _1_001**	Even parity	**? = 0**	**0_1_001**
2	checks bits 2,3,6,7 **0?1_001**	Even parity	**? = 0**	0**01_001**
4	checks bits 4,5,6,7 001**_001**	Even parity	**? = 1**	001**1001**

Table 2. Using table to explain how to determine parity bits. Complements Venn diagram (Downey, 2018).

Hamming Code word to transmit: **0011001**

Similarly, we can use Venn diagrams to find the parity bits. In our case, we will use a basic Venn diagram to showcase this implementation (Hamming Codes in TOY).

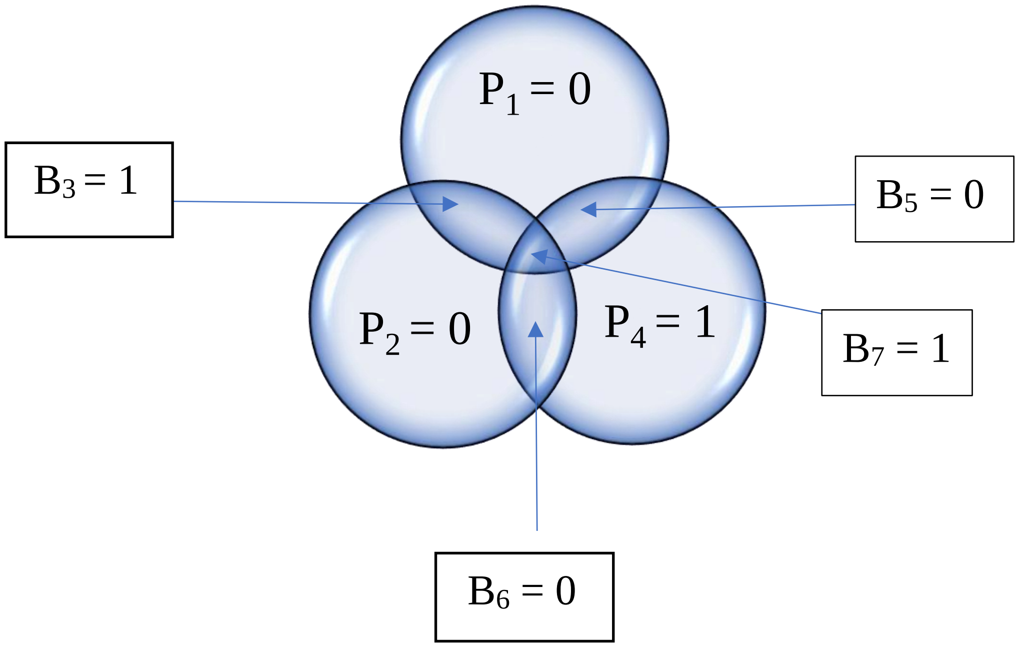

Figure 1 The B_i are the data bits and P_i are the parity bits. The value of the parity bits can be computed by making sure that the sum of the data bits encompassed by the circle of a parity bit and the parity bit itself is even (Wolf, 2017).

If for some reason the message received is **0011101,** we can detect the error by looking at the parity bits and seeing whether even parity is consistent. If not, it means the error exists at this location.

Check parity position	Sequence to bits at check bits	Parity?	Determine error
1	checks bits 1,3,5,7 **0011101**	Not even parity	**Bit 1**
2	checks bits 2,3,6,7 **0011101**	Even parity	**Ok**
4	checks bits 4,5,6,7 **0011101**	Not even parity	**Bit 4**

*Table 3. Simple way to detect and correct a single error. In this table, we see that the error bit can be determined from Bit 1 and Bit 4, which points to Bit 5. So, we flip Bit 5 to correct the error, and the information encoding, which is **0011101** becomes **0011101** (Downey, 2018).*

STEP 1: Using Venn Digrams to show the encoding process

Figure 1 Venn Diagram with message 1001

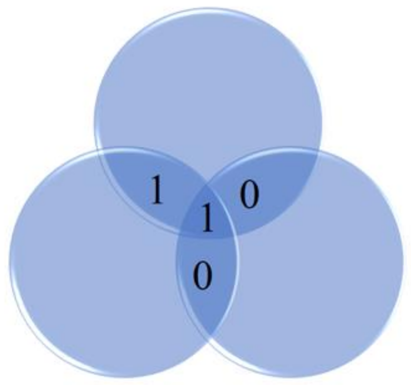

Figure 2 Venn diagram where m1, m2, m3 and m4 are our message bits.

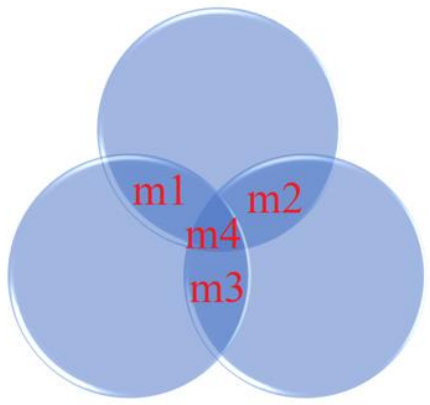

Figure 1 & 2 – Hamming Codes in TOY.

https://www.cs.princeton.edu/courses/archive/spr03/cs126/assignments/hamming.html

STEP 2: Adding parity bits

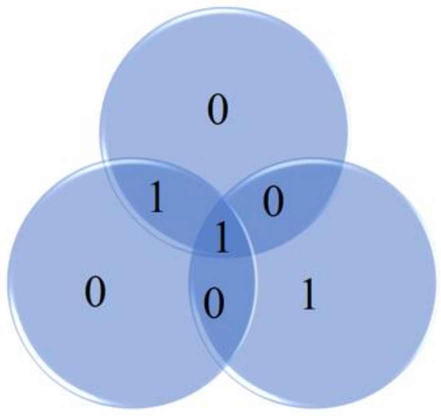

Figure 3 Hamming Code adds 3 parity bits, so each circle has even parity.

Figure 3 – Hamming Codes in TOY.

https://www.cs.princeton.edu/courses/archive/spr03/cs126/assignments/hamming.html

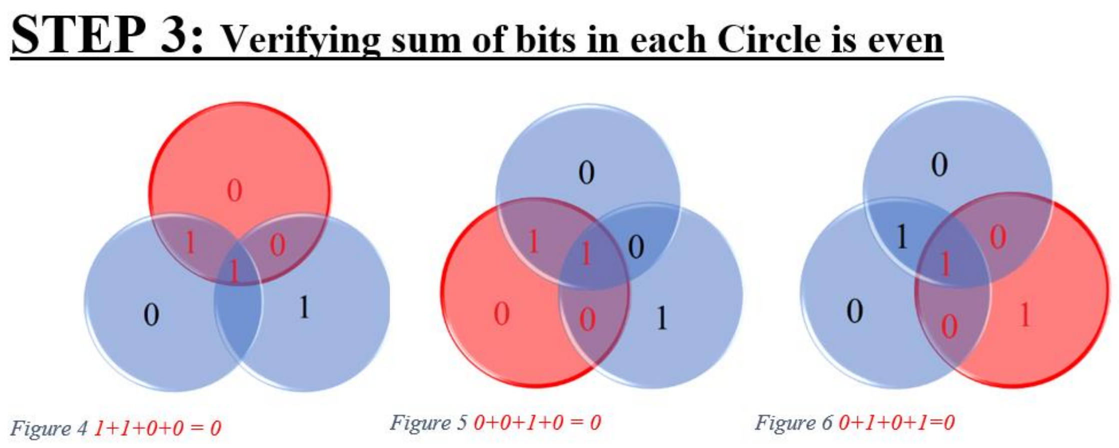

Figure 4 – *Hamming Codes in TOY.*

https://www.cs.princeton.edu/courses/archive/spr03/cs126/assignments/hamming.html

Having explained the essence of determining parity bits and checking for errors, we can now delve deeper into the parity checking matrix that uses the Hamming Code Block (7,4) for error correcting using a hamming code generator matrix.

The Hamming Code Generator Matrix

In practice, Hamming codes are generated by way of matrix multiplication. If the message has k bits, then the message matrix is a k by 1 matrix. We then multiply an n by k matrix called a generator matrix by the message matrix to produce a new matrix which represents the bits to be transmitted. An *n* by *k* generator matrix multiplies a *k* by 1 message matrix. The generator matrix transforms the message matrix into an *n* by 1 package matrix which is the original message matrix with *n*-*k* parity entries (Wolf, 2017). These parity entries describe the other entries in the package matrix, allowing for error detection and correction. We are examining the (7, 4) Hamming code. An example of a (7,4) generator matrix follows:

$$G = \begin{array}{c} \text{k columns} \\ \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \text{n rows}$$

It is worth stating that the orientation and order of matrices in Hamming code generation is not standard and entirely arbitrary. One could easily have a 1 by 4 message matrix M multiplied a 4 by 7 generator matrix G. All that matters is that the parity check matrix matches the generator matrix.

It is also possible to create new generator matrices by applying elementary row operations to G.

There are many different valid Hamming generator matrices, and one often sees deviation from the standard matrix shown here.

The generator matrix works as so: given a message matrix

$$M = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix}$$

And the above generator matrix, the output package matrix will be:

$$M_P = \begin{bmatrix} m_1 + m_2 + m_4 \\ m_1 + m_3 + m_4 \\ m_1 \\ m_2 + m_3 + m_4 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix}$$

Figure 5 - Data item 1, 2 and 4 correspond to bits 1, 3, 5, and 7 in the transmitted message.

The data being transmitted is in binary, so the above raw package matrix will not be transmittable. To remedy this, the contents of M_P are calculated using arithmetic modulo 2. This means that the first, second, and fourth entries will, rather than the straight sum, show the parity of their respective bits. Since any possible permutation of 4 bits is a possible data vector, the valid codewords, or valid package vectors, is the column space of G. A package vector can be multiplied by the parity matrix corresponding to the generator matrix to check for errors. We describe the parity matrix in the next section.

Parity Check Matrix

The parity check matrix allows us to detect an error and the position of that error in our data package. If we multiply a 4-bit data vector by our generator matrix from earlier, we end up with a 7-bit vector $(d_1, d_{2, \ldots,} d_7)^T$. We will have data bits at d_3, d_5, d_6, and d_7 and assign our parity bits variables d_1, d_2, and d_4, where d_1 is responsible for data with a 1 in the least significant place of the subscript when the subscript is converted to binary, d_2 is responsible for data with a 1 in the second least significant place, and d_4 is responsible for data with a 1 in the most significant binary place. We can organize these variables into the following equations:

$$d_3 + d_5 + d_7 = d_1$$

$$d_3 + d_6 + d_7 = d_2$$

$$d5+ d6 + d7 = d4$$

The nature of binary modular arithmetic means that the right side of these equations can be added to the left and subtracted from the right while maintaining the system. Considering the two possible cases shows us why this is a valid operation. A parity bit of 0 indicates an even number of 1s. Adding the parity bit to the left of the equation will not change the value of the equation modulus 2. If the parity bit is 1, that means that the left side of the equation has an odd number of 1s. Adding the parity bit will result in an even number of 1s and a result of 0. Adding the parity bit to the left and subtracting from the right give us:

$$d_3 + d_5 + d_7 + d_1 = 0$$

$$d_3 + d_6 + d_7 + d_2 = 0$$

$$d_5 + d_6 + d_7 + d_4 = 0$$

We can add the missing variables to each equation with a zero coefficient without invalidating the equations. After putting the subscripts in order, we get:

$$d_1 + 0d_2 + d_3 + 0d_4 + d_5 + 0d_6 + d_7 = 0$$

$$0d_1 + d_2 + d_3 + 0d_4 + 0d_5 + d_6 + d_7 = 0$$

$$0d_1 + 0d_2 + 0d_3 + d_4 + d_5 + d_6 + d_7 = 0$$

We can use these equations to create the parity check matrix P, where P is the 3x7 coefficient matrix where each row is the left side of one of the above equations.

$$\begin{matrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

We use this matrix to check for valid data and detect where any error occurred by computing P**x**, where x, the transmission matrix, is an n x 1 matrix consisting of the n transmitted bits. If P**x = 0**, the data is valid. Otherwise, the product P**x** will result in a 3x1 vector where the sum of the positions of the 1s in that vector corresponds to the position of the error in the original data (transmitted) vector.

The matching column's value in binary corresponds to the position of the error in the original data vector. This is just one possible parity check matrix. Our scheme for organizing the data and parity bits to generate P matches a certain set of codewords, or valid data packages. These valid codewords are the null space of P. For the generator parity pairing to be valid, the two matrices must satisfy the equation PG = 0. If we examine our equations used to generate our parity bits from earlier:

$$d_3 + d_5 + d_7 = d_1$$

$$d_3 + d_6 + d_7 = d_2$$

$$d5 + d6 + d7 = d4$$

We can use these equations to create our 7 by 4 generator matrix. Our data is assigned subscripts in the following fashion $(d_3, d_5, d_6, d_7)^T$. If we append the missing data bit to each of our equations with a 0 coefficient, we get the following equations:

$$1d_3+1d_5+0d_6+1d_7 = d_1$$

$$1d_3 + 0d_5 + 1d_6 + 1d_7 = d_2$$

$$0d3 + 1d5 + 1d6 + 1d7 = d4$$

The coefficients of the left side of these equations will become rows 1, 2, and 4 of our generator matrix, corresponding to the position of the parity bits they generate in the outgoing vector. The remaining rows of the matrix, 3, 5, 6, and 7 will represent data bits. Since positions 3, 5, 6, and 7 in our outgoing vector correspond to positions 1, 2, 3 and 4 in our original data vector, these rows must have a 1 in the position of the bit we want to be transmitted, and 0s elsewhere. This gives the generator matrix:

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

We can confirm this is a valid generator/parity pairing be performing the multiplication PG and verifying that we get a 3x4 zero matrix.

Conclusion

In summation, we will discuss some of the applications of Hamming Codes in today's technologies and industries. For instance, in the telecommunications industry, whereby wireless transmission of data consistently takes place, Hamming codes have been used to ensure that the integrity of the data transmitted is retained or recovered, due to the ease with which they can be implemented and the simplicity with which they facilitate error detection and correction.

Additionally, Hamming Codes' low level of redundancy makes them ideal for use in computing systems for memory access and storage (Gupta & Rashmi, 2013). The computer memory, which can be made up of thousands of silicon chips is susceptible to errors due to the nature in which binary information is represented. The bits are represented by the "presence or absence of negative electric charges" on the silicon chips, with 0 being represented by the presence of electrons in an allocated chip portion. So, when the reading of such a chip portion is higher than a predetermined value, then it is read to be a 0, and vice versa. Nonetheless, the chips can be damaged directly (physically) or indirectly (from particle bombardments), hence, "one megabyte memory consisting of 128 64K chips", might have 4 rows of 32 chips, and an additional 7 chips for error-correction (Key, 2000, p. 301). These Hamming codes are referred to as binary extended Hamming codes, since they have an n that is longer than the n = 7, in (7, 4) Hamming codes. In the extended case, for the computer memory, the k can be as big as 57, but the designed preference is k = 32 (Key, 2000).

In all, we have seen that by using linear algebra to convert packets of bits into self-correcting codewords, Hamming codes allow error detection and correction in data where errors are relatively infrequent. We have also seen how a Hamming code generator matrix takes a data packet of a prespecified length and adds redundant bits that reference the parity of the specific bits in the raw data packet. When the transmission is received, simple multiplication by a parity check matrix checks the data for errors and detects the position of an error that may have occurred. All these reasons and others, showcase how Hamming Codes' application of Matrix Algebra, renders their design simple for practical applications in many industries and technologies today.

References

Downey, T. (October 2018). *Calculating the Hamming Code*. https://users.cs.fiu.edu/~downeyt/cop3402/hamming.html

Gupta, B.K. & Rashmi, S. (April 2013). Novel Hamming code for error correction and detection of higher data bits using VHDL. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.302.7849&rep=rep1&type=p df.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell System Technical Journal* 29(2), 2.

Key, J. D. (2000). Some error-correcting codes and their applications. In D. R. Shier , K. T. Wallenius, eds., *Applied Mathematical Modeling: A Multidisciplinary Approach* (pp. 291-314). Boca Raton, FL, CRC Press.

Moon, T.K. (2005). *Error correction coding: Mathematical methods and algorithms. WileyInterscience,* John Wiley & Sons, Inc.

Wolf, J.K. (March 2017). An introduction to error correcting codes, part 2 [web pdf]. http://circuit.ucsd.edu/~yhk/ece154c-spr17/pdfs/ErrorCorrectionII.pdf.

Figures

Hamming Codes in TOY. (March 2003).

http://www.cs.princeton.edu/courses/archive/spr03/cs126/assignments/hamming.html.