**Hamming Codes**

Hamming codes belong to the class of codes known as Linear Block Codes. We will discuss the generation of single error correction Hamming codes and give several mathematical descriptions of them (parity, error-bit specifying, Linear independence of the H-matrix).

First, let’s intuitively develop the single-error correcting code by considering the additional check bits as parity bits. Consider a 4-bit data space where each data word consists of the 4 bits \(d_0d_1d_2d_3\).

Consider \(c_0\) to be a parity bit across bits 0-2. That is, the bit combination \(d_0d_1d_2c_0\) has even parity (we could equally as well have chosen odd parity). If those bits are received with odd parity then it is known that an error has occurred (note that the bit in error might be the \(c_0\) bit itself.) That is, if \(s_0 = d_0 \oplus d_1 \oplus d_2 \oplus c_0 = 0\), then there is no error, otherwise \((p_0 = 1)\) an error has occurred.

Now consider \(c_1\) to be a parity bit across bits 1-3, so that \(d_1d_2d_3c_1\) has even parity. We now have a codeword of six bits, \(d_0d_1d_2d_3c_0c_1\), with the following properties:

- **d.** if \(s_0=1\) and \(p_1=0\) we know there was an error in bits 0-2 or \(c_0\), but not in bits 1-3 or \(c_1\). I.e. the error is in bit 0 or \(c_0\).
- **e.** if \(s_0=0\) and \(s_1=1\) we know there was an error in bits 1-3 or \(c_1\), but not in bits 0-2 or \(c_0\). I.e the error is in bit 3 or \(c_1\)
- **f.** If both \(s_0\) and \(s_1\) are 1, we know the error is in one of the common bits, 1 or 2
- **g.** if neither \(s_0\) or \(s_1\) are 1, there is no error.

In each case we have narrowed down the possibility of which bit is in error to just two. We can resolve this down to one bit with one more check bit, \(c_2\), which provides even parity across one of the possible error bits in each of the above three cases: 0, 2, and 3.

We now have three check bits which guarantee odd parity against three overlapping subsets of the four data bits, in such a way that when all three parities are computed at the receiver, the results can be decoded to point directly to the offending bit. The decode table looks as follows:

<table>
<thead>
<tr>
<th>(s_0)</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>Error bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No error has occurred</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>c2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>c1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>d3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>c0</td>
</tr>
</tbody>
</table>

\[E1\]
We have, in fact, generated a Hamming Code for k=4 data bits. The code requires 3 check bits, giving a codeword size of n=7. Such a code is denoted as a (7, 4) code.

The outputs of the three parity checkers taken together, \( s_0, s_1, s_2 \), is called the ‘syndrome’ of the code. The syndrome is decoded to identify the erroneous bit.

Note the following:

h. The code for 4 bit data words just developed is not unique - the parity bits could have been defined differently, so long as the overlapping ranges are properly defined.

i. This particular code is known as a ‘perfect’ Hamming code since the number of possible decodes of the syndrome is precisely equal to the number of bits in the codeword, plus 1 (for the non-error case). In practice, the number of bits in a codeword is usually significantly less than the number that could be identified by decoding the syndrome.

j. Some of the syndromes correspond to check bits. That is, if a check bit is in error it can also be corrected - check bits are treated just like any other bits in the codeword. Such a code is called a ‘self-checking’ code.

k. Since all possible syndromes decode to bits needing correction, double errors are not detected. In fact, double errors look like single errors, resulting in some bit, not necessarily one of the corrupted bits, being erroneously ‘corrected’.

In general, we need to know how many check bits, \( c \), are necessary for single error correction for \( k \) data bits. The number of check bits \( c (= n-k) \) is the same as the number of bits in the syndrome, and we need to be able to decode the syndrome into at least \( n \) different bit positions. This leads to the following very important constraint, which determines how many check bits are required:

\[
2^{n-k} \geq n+1
\]  \[E2.1\]

Alternatively, if \( c \) is the number of check bits, then \( n = c+k \) and \( c = n-k \). Then [E2] can be rewritten as

\[
2^c \geq n+1
\]  \[E2.2\]

For example, suppose that data words were 24 bits in length. How many check bits are required to provide single error correction? We need to satisfy the inequality

\[
2^{n-24} \geq n+1
\]
We know that \( n > 24 \) and that the left side of the inequality is a power of two. The next power of two above 25 (\( n+1 \)) is 32, so, using trial and error, we try

\[
2^{n-24} = 32
\]

which gives \( n-24 = 5 \) and \( n = 29 \). \( n+1 = 30 \), so we have

\[
2^{29-24} \geq 30.
\]

The inequality is satisfied and \( 29-24 = 5 \), which is the number of required check bits. The total number of bits in a codeword is \( n=24+5 = 29 \).

---

**Practice Problems - Number of Hamming Check Bits**

1. What is the minimum number of check bits required for single-error correcting codes with the following numbers of data bits?

   a. 1  c. 3  e. 11
   b. 33 d. 57 f. 64

2. Which of the codes generated above is a ‘perfect’ code?

3. What is the maximum number of data bits possible in codes with each of the following number of check bits?

   a. 1  b. 2  c. 5  d. 6

4. How many check bits are there in each of the following codes, assuming codewords of the given lengths, and assuming a minimum number of check bits?

   a. 8  b. 12  c. 32  d. 128

---

A Hamming Code is generally described by the use of a matrix, called the Hamming matrix, or H-matrix. The H-Matrix is an \( n-k \) by \( n \) matrix which defines which data bits are used to combine with each check bit for generating the syndrome. For our previous example with \( k=4 \) the H-matrix is

\[
d_0 d_1 d_2 d_3 c_0 c_1 c_2
\]
In practice, odd parity is used, but the choice in this text eliminates some complexity which would only serve to cause confusion.

We have labeled the columns with the identifiers for the data and check bits. Looking at the first row of the matrix, it says that $c_0$ provides even parity\(^{11}\) over bits $d_0$, $d_1$, and $d_2$. In other words, $c_0$ is determined from the Boolean equation

$$c_0 = d_0 \oplus d_1 \oplus d_2.$$  \[E4.1\]

Similarly

$$c_1 = d_1 \oplus d_2 \oplus d_3$$ and  \[E4.2\]
$$c_2 = d_0 \oplus d_2 \oplus d_3.$$ \[E4.3\]

Compare [E3] with [E1]. Notice that the syndrome decodes in [E1] compare exactly with the columns in [E3].

The elements of the syndrome are generated as

$$s_0 = c_0 \oplus d_0 \oplus d_1 \oplus d_2$$
$$s_1 = c_1 \oplus d_1 \oplus d_2 \oplus d_3$$
$$s_2 = c_2 \oplus d_0 \oplus d_2 \oplus d_3$$

Notice that, in the absence of errors, the above equations are equivalent to

$$s_0 = c_0 \oplus c_0 = 0$$
$$s_1 = c_1 \oplus c_1 = 0$$
$$s_2 = c_2 \oplus c_2 = 0$$

Error correction is performed as follows:

1. The check bits $c_i$ are generated according to the H-matrix and [E4].
2. The resulting codeword is stored or transmitted.
3. When data is fetched from storage, or received at the other end of the channel, a new set of check bits $c'_i$ are generated, according to the same H-matrix.
4. The old and new check bits are compared (via XOR: $s_i = c_i \oplus c'_i$), generating a syndrome.
5. The syndrome is decoded to determine which bit was in error.

\(^{11}\)In practice, odd parity is used, but the choice in this text eliminates some complexity which would only serve to cause confusion.
6. The bit in error is corrected (flipped).

Example. Suppose we are given the code defined by \([E3]\). What is the codeword corresponding to the dataword 1010?

Using \([E4]\) we have

\[
\begin{align*}
c_0 &= 1 \oplus 0 \oplus 1 = 0 \\
c_1 &= 0 \oplus 1 \oplus 0 = 1 \\
c_2 &= 1 \oplus 1 \oplus 0 = 0
\end{align*}
\]

The codeword is then: 1010 010. Suppose this code word is sent and the receiver receives 1011010. What is the syndrome and which bit does the syndrome say is in error?

A new set of check bits \((c'_i)\) is generated as above (at the receiver), using 1011 as the data:

\[
\begin{align*}
c_0' &= 1 \oplus 0 \oplus 1 = 0 & \text{Same as sent. Was not affected by the error} \\
c_1' &= 0 \oplus 1 \oplus 1 = 0 & \text{This bit is different, since } d_3 \text{ has changed} \\
c_2' &= 1 \oplus 1 \oplus 1 = 1 & \text{This bit is different, since } d_3 \text{ has changed}
\end{align*}
\]

The syndrome is generated by comparing the old and new check bits:

\[
\begin{align*}
s_0 &= c_0 \oplus c_0' = 0 \oplus 0 = 0 \\
s_1 &= c_1 \oplus c_1' = 1 \oplus 0 = 1 \\
s_2 &= c_2 \oplus c_2' = 0 \oplus 1 = 1
\end{align*}
\]

The syndrome, 011, is the same as the column under \(d_3\) in the H-matrix. Correction can be immediately be done by flipping \(d_3\), whereupon the original codeword (1010010) is obtained. By deleting the check bits the original data (1010) is retrieved.
Practice Problems - Determining Syndromes

Consider the following H-Matrix
\[
\begin{array}{cccccccc}
d_0 & d_1 & d_2 & d_3 & d_4 & c_0 & c_1 & c_2 \\[1] 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \[2] 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \[3] 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \[4] 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\end{array}
\]

1. What are the equations which determine \(c_0, c_1, c_2, \text{ and } c_3\) (refer to [E4])?

2. Given the following data words, find the corresponding codewords
   a. 00000 b. 11111 c. 00110 d. 10101

3. Given the following codewords, determine which have errors, and the error location. For those with errors, determine the original codeword.
   a. 110001110 b. 101010101 c. 000000100

How do we generate an H-matrix for an arbitrary number of data bits?

1. Determine the number of check bits (C) using [E2].
2. The H-matrix will have \(C\) rows and \(k+C = n\) columns
3. Write out the data and check bit labels as was done above [E3]
4. Under each check bit write C 1's and 0's such that there is exactly one 1 in each column, and no two columns are the same. (That is, the C x C subarray of the H-matrix below the check bits will be an identity matrix.)
5. In each of the remaining columns (i.e. under each data bit label) write any arbitrary sequence of C 0's and 1's, ensuring only that all columns are unique with respect to each other, and that every column contains at least two ones (all the sequences using one 1 have been used under the check bit labels, and the sequence of all 0's is not allowed since it is reserved for identifying the non-error case.)

The result will be a valid H-matrix for a Single-Error-Correcting (SEC) code. It will not be unique since assignment of sequences of length C to the columns is arbitrary, and, in fact, if \(2^{n-k} (= 2^C)\) is not exactly equal to \(n+1\), only a subset of all possible column sequences of 1's and 0's will be used, and which subset that happens to be is again arbitrary.
Example: Generate an H-Matrix for 6 data bits.

1. From \([E2]\), \(2^{n-6} \geq n+1\). Since \(n+1\) is greater than 6, we try \(2^{n-6} = 8\), giving \(n-6 = 3\) check bits. But \(n+1 = 3+6+1 = 10\) and \(2^3 = 8\) is not \(\geq 10\).

   So we try \(n-6 = 4\). \(2^4 = 16\) which is greater than \(4+6+1 = 11\).

2. The H-matrix will have 4 rows and 10 columns

3. \(d_0d_1d_2d_3d_4d_5c_0c_1c_2c_3\)

4. \[
1 \ 0 \ 0 \ 0  \\
0 \ 1 \ 0 \ 0  \\
0 \ 0 \ 1 \ 0  \\
0 \ 0 \ 0 \ 1 
\]

5. \[
1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \\
0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \\
1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \\
1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1
\]

**Practice Problems - Developing H-matrices**

1. Develop H-Matrices for codes with the following numbers of data bits
   a. 1  b. 2  c. 7

2. How many rows will there be in an H-Matrix, assuming
   a. There are 5 check bits
   b. There are 64 data bits

The above procedures provide a Hamming code which detects and corrects single errors. Should a double (or any even number) of errors occur, they may not even be detected. In the code given above, double errors are detected but, unfortunately, the syndromes they produce are valid syndromes for single errors. This results in the wrong bit being 'corrected'. In the above example, for instance, if both bits \(d_0\) and \(d_1\) become corrupted, the resulting syndrome will call for correcting \(c_3\). If \(d_3\) and \(d_4\) were to be both in error, \(d_1\) would be 'corrected'.
This defect can be corrected by adding an additional check bit which is basically a parity bit across all of the other bits. Thus we ensure that the total number of 1's in any codeword is always even (or odd). Then, if a double error occurs, if the parity across the control word remains even, but the syndrome is non-zero, we know that a double error has occurred. Codes thus produced are known as **Single-Error-Correction/Double-Error Detection (SEC-DED) codes**.

Example: \([E3]\) can be modified by adding a fourth (parity) check bit to the H-matrix as shown below

\[
\begin{array}{cccccc}
d_0 & d_1 & d_2 & d_3 & c_0 & c_1 & c_2 & c_3 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\textbf{[E5]}

Consider the data word 0001. Using \([E3]\) and\([E4]\), \(c_0 = 0, c_1 = 1, c_2 = 1\) and \(c_3 = 1\).

If, after storage and retrieval, say, 1000 is received (errors in bits \(d_0\) and \(d_3\)) the new check bits are \(c_0' = 1, c_1' = 0, c_2' = 1\) and \(c_3' = 1\). The syndrome \(s = 0111 \oplus 1011 = 1100\). Since the syndrome is non-zero we know that an error has occurred, but since it doesn’t correspond to any of the columns in \([E5]\) we know that it was a double error. Without the fourth check bit, the syndrome would have been \(s = 011 \oplus 101 = 110\) which would have caused bit \(d_1\) to be erroneously corrected.
**Practice Problems - SEC-DED codes**

Consider the following H-Matrix (from an earlier Practice Problem set.)

\[
\begin{array}{cccccccc}
    d_0 & d_1 & d_2 & d_3 & c_0 & c_1 & c_2 & c_3 \\
    1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
    1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\
    0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
    0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\end{array}
\]

1. Modify the matrix so that it provides double error detection as well as single error correction.

2. What is the equation which determines \( c_4 \)?

3. Given the following data words, find the corresponding codewords
   a. 00000  b. 11111  c. 00110  d. 10101

4. Given the following codewords, determine which have errors, and whether the errors are single errors or double errors
   a. 1100101000  c. 000100001  
   b. 0010101110  d. 1111101101