ERROR CORRECTION: HAMMING CODES

Robert P. Webber, Longwood University

It is nice to be able to detect that a transmission error occurred. It would be nicer to be able to find and correct the error. There are several schemes to do this. A standard one is calling **Hamming code**.

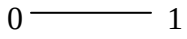
A Hamming code is a combination of 0s and 1s, but not all combinations of 0s and 1s are valid codes. The **Hamming distance** between two binary numbers of the same length is the number of positions in the numbers that have different values. For example, the Hamming distance between 1101 and 1000 is 1, since they differ in only one position. The Hamming distance between 1101 and 1011 is 2, since they differ in two positions.

Consider a Hamming code of length 1. That is, there are only two possible codes, 0 and 1. If both represent valid codes, there is no way to detect a transmission error. If an error occurs, than a 0 become a 1, or a 1 becomes a 0, and both are valid codes. All possible received words are valid, and no detection or correction is possible.

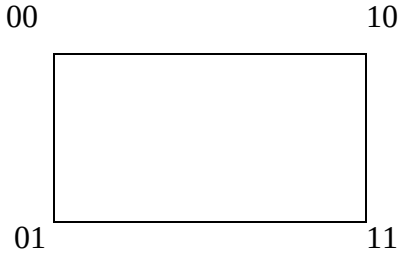
Now suppose we use a two bit code. This provides four possible values: 00, 10, 10, and 11. Suppose we specify that only 00 and 11 are valid codes. If 01 or 10 are received, we know an error has occurred. That is, we have error detection capability.

Here's a graphical way to look at what we have so far.

One bit codes: 0 and 1 are possible codes; both are valid. Put the possible codes at the ends of a line segment.



Two bit codes: 00, 01, 10, 11 are possible codes; 00 and 11 are valid. Put the possible codes at the corners of a rectangle.

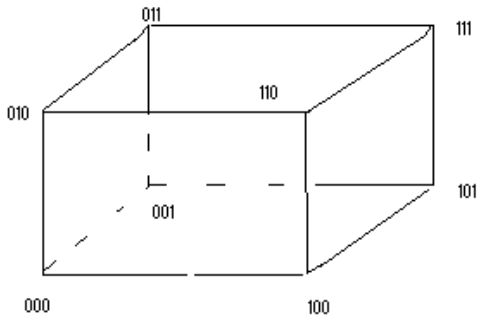


Notice that we arranged the possible codes so that the Hamming distance between any two adjacent vertices is 1, and the Hamming distance between any two non-adjacent vertices is 2.

This provides error detection, but problems remain.

- We can't detect which bit is wrong if we detect an error. For example, suppose we receive 01. It is not a legal code, so an error has occurred. But we don't know whether it came from 00 (with the second bit changed to 1) or from 11 (with the first bit changed to 0).
- A double error would not be detected at all. If we receive 11, it could have originally been 00 or 11.

Next, suppose we use three bit values. There are eight possible values, which can be represented as vertices of a cube. As before, label the vertices so that any two adjacent vertices differ in only one position.



Now there are Hamming distances of 1, 2, and 3. For instance, 000 and 111 differ in three positions. So do the pairs 100 , 011; 101 , 010; and 110 , 001. Choose a pair of codes that are at a Hamming distance of 3 apart to be the valid codes. All others represent errors.

Suppose we choose 000 and 111 as the valid codes. A single error in transmission produces a word that is a distance of 1 from the correct value and a distance of 2 from the other value. For example, suppose 001 is received. It is invalid, so an error occurred. What could the correct word have been? It could not have been 111, because that would have been two errors. The values 001 and 111 differ in two places. Therefore, it must have been 000. We have detected and corrected the error!

Similarly, suppose 011 is received. Could the original number have been 000? No, because that would have been two errors. So the original value must have been 111.

Indeed, if an invalid code is received, and we assume only one transmission error, we can correct the error by substituting the closest legal code word from the diagram. The invalid codes 001, 010 and 100 correct to 000; while 011, 101, and 110 correct to 111.

This example illustrates a **majority logic** code. Triple redundancy is used to encode the data (three bits are used to encode a single data bit), and the data decoded is that corresponding to two of the three received symbols that are alike, if not all three are the same.

This error correcting code works well for single transmission errors, but there are still some problems.

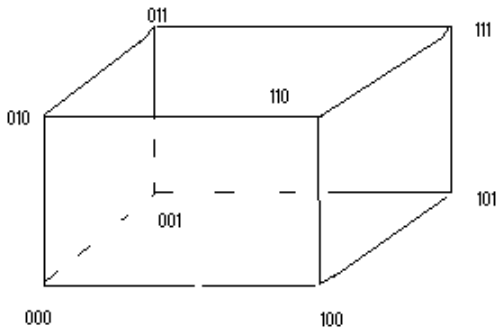
- If a double error occurs, it will be corrected erroneously! For example, suppose 011 is received. It will be corrected to 111. But if the original was 000, and the second and third bits got switched in transmission, the correction would be wrong.
- A triple error would not be detected at all.

Here are some general results.

- If the minimum distance between any two valid codes is 1, no detection or correction is possible.
- If the minimum distance between any two valid codes is 2, detection of single errors is possible, but not correction. A double error will not be detected.
- If the minimum distance between any two valid codes is 3, detection and correction of single errors is possible. A double error will be corrected erroneously, and a triple error will not be detected.

Notice the phrase, "minimum error." We do not require that all valid codes be the same distance apart, but only that no two valid codes be closer than the minimum distance. Also note that by providing sufficient minimum distance between valid codes, any desired number of errors can be corrected. Of course, this would require a lot of extra bits. If we allowed more possible codes to be legal, then we could send more data, but we might lose some error detection or correction capability.

For example, suppose we are using three bit codes, and the legal codes are 001, 010, 100, and 111. Can we detect errors? Can we correct them?



Since the Hamming distance between any two legal codes is 2, we can detect single errors in transmission. We cannot correct them, however. For example, suppose 000 arrives. It is incorrect, but we don't know whether it came from 010 or 100 .

Notice in this example that the legal codes all have odd parity. We could think of it this way: We are using two bit codes with a parity bit.

Code	*Code + parity*
00	001
01	010
10	100
11	111

If a transmission arrives with even parity, we know an error has occurred. This certainly makes it easy to detect errors.

There are some general principles here.

- For single bit error detection, we need add only one bit to the standard word size. For instance, if we are using 16 bits words, adding a 17^{th} bit will give single bit error detection. Adding the extra bit guarantees that the minimum distance between any two valid words will be 2. The extra bit can be considered as a parity bit, and setting it to even or odd parity achieves the Hamming distance of 2 between any two valid words. That is, if we are using odd parity, and two words have odd parity, there is a distance of at least 2 between them.
- Each time we increase the Hamming distance, we pick up an increased ability to detect or correct errors.

Min distance	*Detect*	*Correct*
2	Single errors	None
3	Single errors	Single errors
4	Single, double	Single
5	Single, double	Single, double
……..	……	……
2*n*	Single, double, …, *n*	Single, double, …, *n*-1
2*n*+1	Single, double, …, *n*	Single, double, …, *n*

For example, if we add five bits, giving a minimum Hamming distance of 6, we can detect single, double, and triple errors, and correct single and double ones.

Here's an example of an error-correcting Hamming code.

Symbol	*Code*
A	110100
B	111111
C	000111
D	001100
E	010010
F	011001
G	100001
H	101010

You can verify that the minimum distance between any two legal codes is 3.

If a single bit is modified in transmission, the result will not be a legal pattern. Moreover, we can correct the transmission error by figuring out the original bit pattern. The incorrect pattern will be a Hamming distance of 1 from its original form, but at least 2 from any of the other legal codes. For example, suppose we receive the code 010100 . This is not one of the eight legal codes, so it is incorrect. Calculate its distance from each of the legal codes. In the table, the different bits are highlighted.

Symbol	*Code*	*Received*	*Distance*
A	110100	010100	1
B	111111	010100	4
C	000111	010100	3
D	001100	010100	2
E	010010	010100	2
F	011001	010100	3
G	100001	010100	4
H	101010	010100	5

Since the first legal code is the only code to differ from the received pattern in only one place, the original code must have been 'A'.

Suppose the following string of codes was received. What was the original message?

Received string: 010100 100001 010110

We already figured that the first pattern is incorrect, and we corrected it to the code for 'A'. The second pattern, 100001, is a valid code, the pattern for 'G'. The third pattern, 010110, is incorrect. Looking at the table, it differs in exactly one position from the code for 'E'. Therefore, the corrected message is

Original message: AGE

How do we decide on the legal bit patterns for a Hamming code? There are several schemes, and we will go through one here. It uses parity bits to provide single bit error detection and correction. Even or odd parity can be used; we will use odd parity in the example.

Suppose that we want to be able to transmit three data bits and to be able to detect and correct single bit errors in transmission. In other words, we want to be able to reliably transmit any of the six data strings 000, 001, 010, 100, 101, 110, and 111. Let us devise a Hamming code to accomplish this.

Use a six bit code, with bits 3, 5 and 6 holding the data bits.

___	___	*d*	___	*d*	*d*
1	2	3	4	5	6

d denotes a data bit
These numbers are bit positions

Place 0 or 1 in bit position 1 so that positions 1, 3, and 5 are in odd parity.
Place 0 or 1 in bit position 2 so that positions 2, 3, and 6 are in odd parity.
Place 0 or 1 in bit position 4 so that positions 4, 5, and 6 are in odd partity.

Data 000:

___	___	0	___	0	0
1	2	3	4	5	6

Bits 3 and 5 are 00, so set bit 1 to 1 for odd parity.
Bits 3 and 6 are 00, so set bit 2 to 1 for odd parity.
Bits 5 and 6 are 00, so set bit 4 to 1 for odd parity.
The resulting Hamming code string is 110100

Data 001:

___	___	0	___	0	1
1	2	3	4	5	6

Bits 3 and 5 are 00, so set bit 1 to 1 for odd parity.
Bits 3 and 6 are 01, so set bit 2 to 0 for odd parity.
Bits 5 and 6 are 01, so set bit 4 to 0 for odd parity.
The resulting Hamming code string is 100001.

You can derive the Hamming code strings for the remaining data values. Here are the results.

Data	*Hamming code*
000	110100
001	100001
010	010010
011	000111
100	001100
101	011001
110	101010
111	111111

Notice that this is the same Hamming code, except for order, as the previous example.

There is a quick way to correct an error in a Hamming code that was derived using parity as above. Suppose 001110 is received. Find the parity bits that are wrong. Remember that bits 3, 5, and 6 are data bits and the remaining bits are parity bits, and that we are using odd parity.

0	0	1	1	1	0
1	2	3	4	5	6

Bits 3 and 5 are 11, so bit 1 should be 1 for odd parity.
Bits 3 and 6 are 10, so bit 2 is correct
Bits 5 and 6 are 10, so bit 4 should be 0.

Add the numbers of the parity bit positions that are incorrect. 1 + 4 = 5, so data bit 5 is incorrect. The correct code is 001100, corresponding to data 100 . In general, find the bad parity bits and add their locations to find the location of the bad data bit.

Here is the general method to obtain a Hamming code using parity bits. First, decide on a length for the codes. Above, the length was 6 bits. Then do the following.

- All positions that are powers of 2 are parity bits (positions 1, 2, 4, 8, …)
- The remaining positions are data bits (positions 3, 5, 6, 7, 9, …)
- Each parity bit checks the parity of some of the data bits. The position of the parity bit determines the bits that it checks. Alternately check and skip bits as follows.
    - Bit 1: Check and skip bits in groups of 1 (bits 1, 3, 5, 7, 9, …);
    - Bit 2: Check and skip bits in groups of 2 (bits 2, 3, 6, 7, 10, 11, …);
    - Bit 4: Check and skip bits in groups of 4 (bits 4, 5, 6, 7, 12, 13, 14, 15, 20, 21, 22, 23, …), and so on.

Set each parity bit to 0 or 1 to produce the proper parity for the bits it checks.

Suppose all three parity bits are wrong. For instance, suppose 000000 is received. All three parity bits should be 1, and 1 + 2 + 4 = 7 > 6 ! This indicates that at least two bits were changed. We can detect double errors, but we can't correct them. Even worse, it is possible that a double error might be corrected erroneously! Suppose that 100001 is transmitted, but 000011 is received.

0	0	0	0	1	1
1	2	3	4	5	6

Bits 3 and 5 are 01, so bit 1 is correct.
Bits 3 and 6 are 01, so bit 2 is correct.
Bits 5 and 6 are 11, so bit 4 is wrong.

Changing bit 4 produces 000111, which actually has two incorrect bits.

Now consider a 15 bit Hamming code. Bits 1, 2, 4, and 8 will be parity bits, and the remaining 11 bits will hold data.

p	*p*		*p*				*p*							
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Set bit 1 so that bits 1, 3, 5, 7, 9, 11, 13, and 15 will be in odd parity.
Set bit 2 so that bits 2, 3, 6, 7, 10, 11, 14, and 15 will be in odd parity.
Set bit 4 so that bits 4, 5, 6, 7, 12, 13, 14, and 15 will be in odd parity.
Set bit 8 so that bits 8, 9, 10, 11, 12, 13, 14, and 15 will be in odd parity.

For instance, suppose the data bits are 10001011001. What will be the code for this word?

p	*p*	1	*p*	0	0	0	*p*	1	0	1	1	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Bits 3, 5, 7, 9, 11, 13, and 15 are 1001101, so set bit 1 to 1 for odd parity.
Bits 3, 6, 7, 10, 11, 14, and 15 are 1000101, so set bit 2 to 0 for odd parity.
Bits 5, 6, 7, 12, 13, 14, and 15 are 0001001, so set bit 4 to 1 for odd parity.
Bits 9, 10, 11, 12, 13, 14, and 15 are 1011001, so set bit 8 to 1 for odd parity.

The result is the Hamming code 101100011011001. Similarly, we could get the Hamming code for any of the other possible combinations of 11 data bits.

EXERCISES

In problems 1 and 2, suppose we are using three bit codes with the legal codes being 000 and 111. Assume one bit transmission errors.

1. Which of the following received words are incorrect? Correct the incorrect words. 010, 011, 111, 101

2. Which of the following received words are incorrect? Correct the incorrect words. 110, 000, 100, 110

In problems 3 and 4, suppose we are using three bit codes with the legal codes being 010 and 101. Assume one bit transmission errors.

3. Which of the following received words are incorrect? Correct the incorrect words. 100, 001, 000, 101

4. Which of the following received words are incorrect? Correct the incorrect words. 100, 010, 011, 110

In problems 5 and 6, suppose we are using three bit codes and we can correct single transmission errors.

5. Can the following pairs be legal code words? Explain.
   a. 001, 110
   b. 111, 110
   c. 000, 011

6. Can the following pairs be legal code words? Explain.
   a. 101, 100
   b. 010, 101
   c. 110, 101

7. Suppose we are using three bit codes and the legal codes are 001, 010, 100, and 111.
   a. Can we detect single transmission errors?
   b. Can we correct single transmission errors?

8. Suppose we are using three bit codes and the legal codes are 000, 101, 110, and 111.
   a. Can we detect single transmission errors?
   b. Can we correct single transmission errors?

9. If the minimum Hamming distance between valid codes is 6,
   a. How many errors can be detected?
   b. How many errors can be corrected?

10. If the minimum Hamming distance between valid codes is 7,
    a. How many errors can be detected?
    b. How many errors can be corrected?

11. If a code can detect, but not correct, five errors, what is the minimum Hamming distance for the code?

12. If a code can detect and correct five errors, what is the minimum Hamming distance for the code?

In exercises 13 through 20, use the six bit Hamming code in the text. Use the symbols A through H in the first version of that code as needed.

13. Detect and correct the single bit transmission error in the received string 010100.

14. Detect and correct the single bit transmission error in the received string 011011.

In exercises 15 through 20, decode the following messages using the Hamming code, correcting transmission errors as necessary, and using the symbols for the results.

15. 001100 110100 101100

16. 111111 010010

17. 010010 001100

18. 111110 110100 111111 010011

19. 011001 011010 011100

20. 111001 110110 001100

21. Find a six bit, single bit error correcting and detecting Hamming code for the data values 000, 001, 010, 011, 100, 101, 111 as in the text, except use even parity.

In exercises 22 and 23, use the six bit Hamming code with odd parity derived in the text:

Data	*Hamming code*
000	110100
001	100001
010	010010
011	000111
100	001100
101	011001
110	101010
111	111111

22. Suppose the bit string 010100 is received. Find and correct the error.

23. Suppose the bit string 011011 is received. Find and correct the error.

In exercises 24 through 27, use a 15 bit Hamming code with positions 1, 2, 4, and 8 for parity, and the other bits for data.

24. Give the Hamming code for the data bits 01001000111 using odd parity.

25. Same as Exercise 24, but use even parity.

26. Give the Hamming code for the data bits 10011011100 using odd parity.

27. Same as Exercise 26, but use even parity.