University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Error Correction, Hamming Codes, and SEC-DED Codes

© 2015 Steven S. Lumetta. All rights reserved.

Error Detection May Not Be Enough

Detection of errors is not always enough.

For example, if your bank's storage server detects a bit flip in your account balance.

Say that the balance (with a bit flip) is $500.

Someone must **choose a bit to flip back**.

Choices, each flipping one bit, include:
- **$500** (change the parity bit)
- **-$500** (overdrawn! the bank's favorite)
- **$9.223×10^{21}** (your choice of bit, I'd think)

Hopefully, You Never See This (Again)

Can We Use Redundancy to Correct Errors?

Yes, but the **overhead**—the number of extra bits that we have to use—is higher.

Recall **3-bit 2's complement** with odd parity.

-4	⟷ 1000	0	⟷ 0001
-3	⟷ 1011	1	⟷ 0010
-2	⟷ 1101	2	⟷ 0100
-1	⟷ 1110	3	⟷ 0111

The Hamming distance of the code is 2.

Can the code correct an error?

With H.D. 2, Some Errors are Not Correctable

If we observe 0011 after a bit error,
what was the original bit pattern?

0 ⟷ 0001

a possible bit flip

0011

another possible bit flip

0010 ⟷ 1

Which answer should we pick?

Larger Hamming Distance Allows Error Correction

But what if we have a larger Hamming distance?

Consider the code shown here, based on **2-bit unsigned**:

Each code word consists of three copies of the original representation.

0 ⟷ 000000
1 ⟷ 010101
2 ⟷ 101010
3 ⟷ 111111

What is the Hamming distance of this code? 3

Larger Hamming Distance Allows Error Correction

What if one bit flips?

Changing one bit **can only change one of the three copies**.

The other two copies then “vote” for the right answer.

2 ⟷ 101010

a possible bit flip

100010

And **we can correct the error**!

Error Correction Fails if Too Many Bits Have Flipped

What if two bits flip?

Changing two bits **can change two of the three copies**.

The incorrect copies may outvote the right answer!

2 ⟷ 101010

two possible bit flips

100000

Correction fails if too many bits flip.
Important: You will not know that correction has failed!

Define a Neighborhood Around Each Code Word

Let's try to generalize.

Given a code word **C**, we can define a neighborhood **N_k(C)** of distance **k** around **C** as **the set of bit patterns with Hamming distance ≤ k from C**.

If **up to k bits flip** in a stored copy of **C**, the final bit pattern falls within **N_k(C)**.

We Can Correct Errors if Neighborhoods are Disjoint

When can we correct errors?

Assume that up to **k** bits flip in a stored bit pattern **C** to produce a final bit pattern **F**.

We know that **F** is in **N_k(C)**.

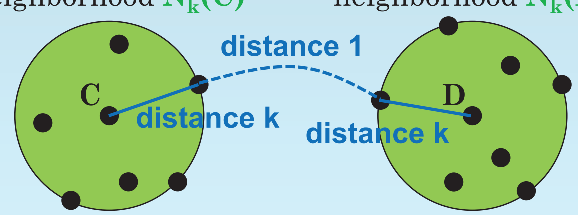
When can we identify **C**, given only **F**?

Only when N_k(C) does not overlap with neighborhood N_k(D) for any other code word **D**.

All Code Words' Neighborhoods Must be Disjoint

If we want to correct **k** errors, we need the neighborhoods **N_k(C)** and **N_k(D)** to be disjoint for any pair of code words **C** and **D**.

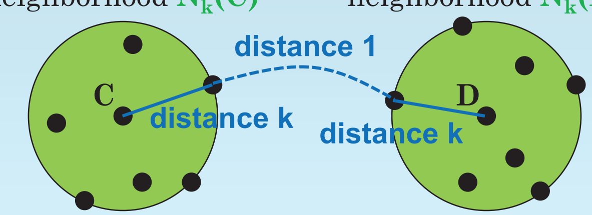
neighborhood **N_k(C)** neighborhood **N_k(D)**



Need Hamming Distance 2k+1 to Correct k Errors

In other words, to correct **k** errors, the distance between code words must be at least **2k + 1**. But that's Hamming distance!

neighborhood **N_k(C)** neighborhood **N_k(D)**



H.D. of d Allows Correction of Floor ((d-1)/2) Bit Errors

In other words, a code with Hamming distance **d** can correct **k** errors iff $\mathbf{d \geq 2k + 1}$.

Solving for **k**, we obtain $\mathbf{k \leq (d - 1) / 2}$.

Since **k** is an integer, we add a floor function for clarity.

Thus, **a code with Hamming distance d allows correction of up to $\left\lfloor \frac{d-1}{2} \right\rfloor$ bit errors**.

Hamming Codes are Good for 1-Bit Error Correction

A **Hamming code** is
- a general and efficient* code
- with Hamming distance 3.

Hamming codes also provide **simple algorithms for correcting 1-bit errors**.

*All bit patterns are part of the 1-neighborhood of some code word.

Defining and Using Hamming Codes

To define a Hamming code on **N** bits,
- number the bits from 1 upwards, and
- make all powers of two even parity bits.

Each parity bit **P** (a power of 2) is based on
- the bits with indices **k**
- for which the bit **P** appears as a **1** in **k**.
- In other words, **(k AND P) = P**.

The binary number formed by writing the parity bits in error as 1s then identifies any bit error.

(7,4) Hamming Code: Four Data Bits and Three Parity Bits

Let's do an example: **a (7,4) Hamming code**.

The **7** is the number of bits in each code word.

And the **4** represents the number of data bits.

The other **3** bits are parity bits.

We can write a code word **X** as $\mathbf{x_7x_6x_5x_4x_3x_2x_1}$.

Notice that there is no $\mathbf{x_0}$.

The **parity bits are $\mathbf{x_1}$, $\mathbf{x_2}$, and $\mathbf{x_4}$**.

Calculation of Parity Bits for a 7-Bit Hamming Code

Parity bit $\mathbf{x_1}$ is even parity on the bits with odd-numbered indices. In other words,

$$\mathbf{x_1 = x_3 \oplus x_5 \oplus x_7}.$$

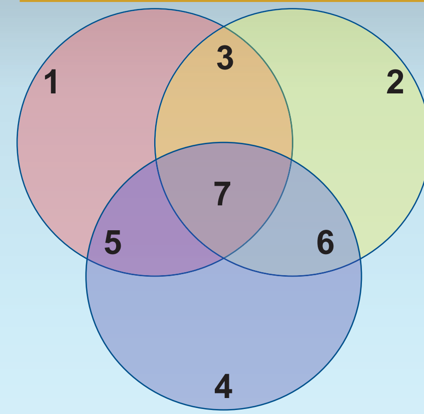
Parity bit $\mathbf{x_2}$ is parity over bits with indices in which the 2s place is a 1. In other words,

$$\mathbf{x_2 = x_3 \oplus x_6 \oplus x_7}.$$

Parity bit $\mathbf{x_4}$ is parity over bits with indices in which the 3rd place is a 1. In other words,

$$\mathbf{x_4 = x_5 \oplus x_6 \oplus x_7}.$$

Graphical View of the (7,4) Hamming Code



To find parity bits:

- Write data bits into areas 7, 6, 5, and 3.
- Choose bit for area 4 such that the blue circle has even parity.
- Do the same for the yellow and red circles.

To check parity bits:

- Check that each circle has even parity.

To correct an error:

- Find the circles with odd parity.
- Flip the bit in the area corresponding the intersection of those circles.

Can We Generalize This Approach to Error Detection?

The graphical approach generalizes,

- but one needs (**N – 1**)-dimensional hyperspheres
- for **N** parity bits.

They are hard to draw on paper when **N > 3**.

Algebraic Encoding for a 7-Bit Hamming Code

We can also work algebraically, of course.

Let's say that we want to **store the value 1001**.

We place our bits into the data bit positions.

So $\mathbf{X = x_7x_6x_5x_4x_3x_2x_1 = 100x_41x_2x_1}$, where the remaining bits must be calculated:

$$\mathbf{x_1 = x_3 \oplus x_5 \oplus x_7 = 1 \oplus 0 \oplus 1 = 0}$$

$$\mathbf{x_2 = x_3 \oplus x_6 \oplus x_7 = 1 \oplus 0 \oplus 1 = 0}$$

$$\mathbf{x_4 = x_5 \oplus x_6 \oplus x_7 = 0 \oplus 0 \oplus 1 = 1}$$

Putting the parity bits in place gives $\mathbf{X = 1001100}$.

Correcting a Bit Error is Easy with a Hamming Code

A bit flips, and we later find **Y = 1001110**.

What was **X**? To correct the error,
- we calculate an error bit for each parity bit
- by XORing the observed parity bit with the correct answer:

$$\mathbf{e_1 = x_1 \oplus x_3 \oplus x_5 \oplus x_7 = 0 \oplus 1 \oplus 0 \oplus 1 = 0}$$
$$\mathbf{e_2 = x_2 \oplus x_3 \oplus x_6 \oplus x_7 = 1 \oplus 1 \oplus 0 \oplus 1 = 1}$$
$$\mathbf{e_4 = x_4 \oplus x_5 \oplus x_6 \oplus x_7 = 1 \oplus 0 \oplus 0 \oplus 1 = 0}$$

Writing $\mathbf{e_4e_2e_1 = 010}$ identifies the error as $\mathbf{x_2}$.

An Error Syndrome of Exactly 0 Means No Error Occurred

What if no error occurs?

Can we accidentally "correct" an already correct bit?

In that case,
- all $\mathbf{e_1}$ values are 0 (all parity bits are correct),
- so $\mathbf{e_4e_2e_1 = 000}$, and **we know that no error occurred**.

Adding a Parity Bit to a Hamming Code Gives H.D. 4

What happens if we add a parity bit to a Hamming code?

In general,
- **adding a parity bit**
- to any code with **odd Hamming distance d**
- **produces** a code with **Hamming distance d + 1**.

So we obtain a code with Hamming distance 4.

What Can We Do with Hamming Distance 4?

Let's think about Hamming distance 4.

If a single bit flip occurs, we can correct it.

However, we cannot correct two bit flips ($\mathbf{\left\lfloor \frac{4-1}{2} \right\rfloor = 1}$).

Hamming Distance of 4 is a SEC-DED Code

However, **if two bit flips occur**,
- the resulting bit pattern is **not in a 1-neighborhood** of the code word
- so **we can avoid "correcting"** the errors.

In other words, we have
- Single Error Correction and
- Double Error Detection.

We call such a code a **SEC-DED code**.