Davis Math Circle: Error Control Coding Handout
Presenter: Michael Ragone
June 1, Spring 2019

Question 1

For our repetition code (with message length 5 bits and encoded message length 15 bits): how many bit flips can we correct with 100% accuracy? What's the maximal number of bit flips we can correct?

■

Finite Fields: Calculating in $\mathbb{Z}_2$

It will be convenient for us to define the finite field $\mathbb{Z}_2$. For us, a **finite field** is a finite set of symbols with some nice properties. Think about real numbers: we can add, subtract, multiply, and divide them. But there are infinitely many of them–we really just need two symbols for our calculations. Two is definitely finite.

So, $\mathbb{Z}_2$ is a set containing two symbols $0, 1$ (In math, $\mathbb{Z}_2 = \{0, 1\}$), and these symbols can be added $(+)$ and multiplied $(\cdot)$ in the following way:

$$\begin{aligned} 0+0&=0 & 0\cdot 0&=0\\ 0+1&=1 & 0\cdot 1&=0\\ 1+0&=1 & 1\cdot 0&=0\\ 1+1&=0 & 1\cdot 1&=1 \end{aligned}$$

and addition/multiplication follow these rules:

Let a, b, c be in $\mathbb{Z}_2$ (meaning each one is either a 0 or 1)

- *Commutativity*: (Ex: $0 + 1 = 1 = 1 + 0$)

  $a + b = b + a$

  $a \cdot b = b \cdot a$

- *Associativity*: (Ex: $0 + (1 + 1) = 0 + (0) = 0 = (0 + 1) + 1$)

  $a + (b + c) = (a + b) + c$

  $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

- *Distributivity*: (Ex: $1 \cdot (1+0) = 1 \cdot 1 + 1 \cdot 0 = 1 + 0 = 1$)

  $a \cdot (b + c) = a \cdot b + a \cdot c$

  $(a + b) \cdot c = a \cdot c + b \cdot c$

- *Identity*: (Ex: $1 \cdot 1 = 1$, $0 + 1 = 1$)

  $a + 0 = a$

  $a \cdot 1 = a$

- *Inverses*: (Ex: $1 + (-1) = 1 + 1 = 0$)

  $a + (-a) = 0$

  $a \cdot a^{-1} = 1$, where $a \neq 0$

  note: in $\mathbb{Z}_2$, $1 = -1$ *since* $1+1 = 0$. *Also, the rule for multiplication isn't very useful here since* 0 *doesn't have a multiplicative inverse and since* $1 = 1^{-1}$.

Back to bitstrings

Now that we have a tool for calculation in $\mathbb{Z}_2$, we can start to think about bitstrings again. It will be convenient to think of bitstrings of length n as ordered lists (called *vectors* in math), like this:

$$a_1 a_2 a_3 \ldots a_n = (a_1, a_2, a_3, \ldots a_n)$$

For example,

$$10110 = (1, 0, 1, 1, 0)$$

This is useful because now, we can refer to certain elements in this vector: above, $a_1 = 1, a_2 = 0, a_3 = 1, a_4 = 1, a_5 = 0$.

Question 2

Let's practice our $\mathbb{Z}_2$ computations and learn a little about vectors. Let $a = (0, 1, 0, 1, 0), b = (1, 1, 0, 0, 1)$, $c = (1, 0, 0, 0, 0), d = (0, 1, 1, 1, 1)$.

(a) For a, b, c, d, calculate the sum of all elements using $\mathbb{Z}_2$ arithmetic (e.g. calculate $a_1 + a_2 + a_3 + a_4 + a_5$). What does this tell you about the original vector?

(b) Calculate three sums: $a_1 + a_2 = ?$, $a_1 + a_3 = ?$, and $a_1 + a_2 + a_3 = ?$, and likewise for b.

Now, let's say I had another, unknown vector $m = (m_1, m_2, m_3, m_4, m_5)$, and let's say I told you that $m_1 + m_2 = 0$, $m_1 + m_3 = 1$, and $m_1 + m_2 + m_3 = 1$. Is it possible to "work backwards" and figure out what m_1, m_2, m_3 are? What about if $m_1 + m_2 = 1$, $m_2 + m_3 = 0$, and $m_1 + m_2 + m_3 = 0$? Could you figure it out if I only gave you $m_1 + m_2 = 1$ and $m_2 + m_3 = 0$?

(c) In problem b), we saw that sometimes, you can take knowledge of the sum and work backwards to find elements of the vector that solve some given equations. But what if your information contradicts itself?

(i) Think about the vector $m = (1, 0, 1, 1, 0)$. Does it satisfy the equation $m_1 + m_2 + m_3 + m_4 + m_5 = 0$?

(ii) Now, imagine I claimed that I have a vector m that satisfies all three of these equations: $m_1 + m_2 = 0$, $m_1 + m_3 = 1$, and $m_2 + m_3 = 0$. Is this possible?

(d) We can define another type of addition–now, on vectors, in the following way:

$$\begin{aligned} a + b &= (a_1, a_2, a_3, a_4, a_5) + (b_1, b_2, b_3, b_4, b_5) \\ &= (a_1 + b_1, a_2 + b_2, a_3 + b_3, a_4 + b_4, a_5 + b_5) \end{aligned}$$

Calculate...

(i) $a + b = ?$

(ii) $a + a = ?$, $b + b = ?$. What do you notice? What happens when you add a vector to itself?

(iii) $c + d = ?$. What do you notice?

■

Question 3 We designed a code that has $m = 5$ message bits and $k = 1$ parity bits. How many errors can it detect? Can it correct any errors? Does it matter where the error occurs?

Question 4 Using this parity bit idea, how can we design codes that detect more errors?

Exercise 5 Design your own parity-check code for messages of length 4. Add k parity bits using different equations to the end of your encoded message.

Exercise 6 Write your parity-check code as a graph.

Exercise 7 Test your parity-check decoder. Can your parity-check decoder detect errors? What kinds of errors? Test it with these messages: $a = (0, 0, 0, 0)$, $b = (1, 0, 1, 0)$, $c = (1, 1, 1, 1)$. We can write noise as vectors of length $m + k$: $n_1 = (1, 0, \ldots, 0)$ (the first bit flips), and $n_2 = (0, \ldots 0, 1, 1)$ (the last two bits flip)