8 Error-Correcting Codes

7 December 2022

Sebastian Wild

COMP526 (Fall 2022)
University of Liverpool

version `2022-12-06 00:49`

Learning Outcomes

1. Understand the context of *error-prone communication*.
2. Understand concepts of *error-detecting codes* and *error-correcting codes*.
3. Know and understand the *terminology of block codes*.
4. Know and understand *Hamming codes*, in particular (7,4) Hamming code.
5. Reason about the *suitability of a code* for an application.

Unit 8: *Error-Correcting Codes*

Outline

8.1 Introduction

Noisy Communication

- most forms of communication are “noisy”
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

Noisy Communication

- most forms of communication are "noisy"
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

- How do humans cope with that?
  - slow down and/or speak up
  - ask to repeat if necessary

Noisy Communication

- most forms of communication are “noisy”
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

- How do humans cope with that?
  - slow down and/or speak up
  - ask to repeat if necessary

- But how is it possible (for us) to decode a message in the presence of noise & errors?

Bcaesue it semes taht ntaurul lanaguge has a lots fo ***redundancy*** *bilt itno it!*

Noisy Communication

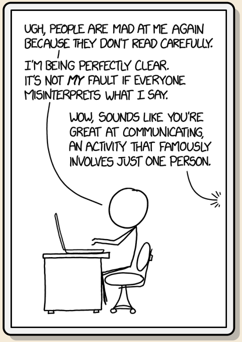
- most forms of communication are "noisy"
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

- How do humans cope with that?
  - slow down and/or speak up
  - ask to repeat if necessary
- But how is it possible (for us) to decode a message in the presence of noise & errors?

Bcaesue it semes taht ntaurul lanaguge has a lots fo ***redundancy*** *bilt itno it!*

⇝ We can

1. **detect errors** "This sentence has aao pi dgsdho gioasghds."
2. **correct** (some) **errors** "Tiny errs ar corrrected automaticly."
   (sometimes too eagerly as in the Chinese Whispers / Telephone)



Noisy Channels

- computers: copper cables &
  electromagnetic interference
- transmit a binary string
- but occasionally bits can "flip"

⇝ want a robust code

Noisy Channels

- computers: copper cables &
  electromagnetic interference
- transmit a binary string
- but occasionally bits can “flip”

⇝ want a robust code

- We can aim at
  1. **error detection** ⇝ can request a re-transmit
  2. **error correction** ⇝ avoid re-transmit for common types of errors

Noisy Channels

- computers: copper cables & electromagnetic interference
- transmit a binary string
- but occasionally bits can "flip"

⇝ want a robust code

- We can aim at
  1. **error detection** ⇝ can request a re-transmit
  2. **error correction** ⇝ avoid re-transmit for common types of errors
- This will require *redundancy*: sending *more* bits than plain message

  ⇝ **goal:** robust code with lowest redundancy

that's the opposite of compression!

Clicker Question

What do you think, how many extra bits do we need to **detect** a **single bit error** in a message of 100 bits?

→ sli.do/comp526

Clicker Question

What do you think, how many extra bits do we need to **correct** a **single bit error** in a message of 100 bits?

→ sli.do/comp526

8.2 Lower Bounds

Block codes

- **model:**
  - want to send message $S \in \{0,1\}^\star$ (bitstream) across a *(communication) channel*
  - any bit transmitted through the channel might ***flip*** ($0 \to 1$ resp. $1 \to 0$)
    no other errors occur (no bits lost, duplicated, inserted, etc.)
  - instead of S, we send *encoded bitstream* $C \in \{0,1\}^\star$
    sender *encodes* S to C, receiver *decodes* C to S (hopefully)

$\leadsto$ what errors can be detected and/or corrected?

Block codes

- **model:**
  - want to send message $S \in \{0,1\}^\star$ (bitstream) across a *(communication) channel*
  - any bit transmitted through the channel might ***flip*** ($0 \to 1$ resp. $1 \to 0$)
    no other errors occur (no bits lost, duplicated, inserted, etc.)
  - instead of S, we send *encoded bitstream* $C \in \{0,1\}^\star$
    sender *encodes* S to C, receiver *decodes* C to S (hopefully)
  - ⇝ what errors can be detected and/or corrected?
- all codes discussed here are ***block codes***
  - divide S into *messages* $m \in \{0,1\}^k$ of k bits each (k = *message length*)
  - encode each message (separately) as $C(m) \in \{0,1\}^n$ (n = *block length*, $n \geq k$)
  - ⇝ can analyze everything block-wise

Block codes

- **model:**
  - want to send message $S \in \{0,1\}^\star$ (bitstream) across a *(communication) channel*
  - any bit transmitted through the channel might ***flip*** ($0 \to 1$ resp. $1 \to 0$)
    no other errors occur (no bits lost, duplicated, inserted, etc.)
  - instead of S, we send *encoded bitstream* $C \in \{0,1\}^\star$
    sender *encodes* S to C, receiver *decodes* C to S (hopefully)
  - ⇝ what errors can be detected and/or corrected?
- all codes discussed here are ***block codes***
  - divide S into *messages* $m \in \{0,1\}^k$ of k bits each (k = *message length*)
  - encode each message (separately) as $C(m) \in \{0,1\}^n$ (n = *block length*, $n \geq k$)
  - ⇝ can analyze everything block-wise
- between 0 and n bits might be flipped
  - how many flipped bits can we definitely **detect**? (invalid code)
  - how many flipped bits can we **correct** without retransmit? (i. e. decoding m still possible)

Code distance

$m \neq m' \implies C(m) \neq C(m')$

- each block code is an *injective* function $C : \{\mathtt{0},\mathtt{1}\}^k \to \{\mathtt{0},\mathtt{1}\}^n$

Code distance

$m \neq m' \implies C(m) \neq C(m')$

- each block code is an *injective* function $C : \{0,1\}^k \to \{0,1\}^n$
- define $\mathcal{C}$ = set of all codewords = $C(\{0,1\}^k)$

$\rightsquigarrow \mathcal{C} \subseteq \{0,1\}^n$ — $|\mathcal{C}| = 2^k$ out of 2^n n-bit strings are valid codewords

- decoding = finding closest valid codeword

Code distance

$m \neq m' \implies C(m) \neq C(m')$

- each block code is an *injective* function $C : \{0,1\}^k \to \{0,1\}^n$
- define $\mathcal{C}$ = set of all codewords = $C(\{0,1\}^k)$

$\rightsquigarrow \mathcal{C} \subseteq \{0,1\}^n$ | $|\mathcal{C}| = 2^k$ out of 2^n n-bit strings are valid codewords

- decoding = finding closest valid codeword

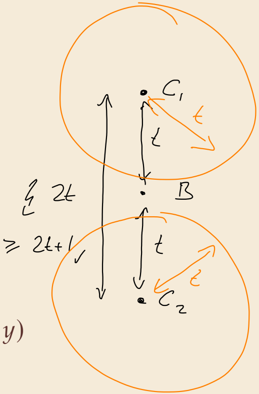
- ***distance of code:***
  d = minimal Hamming distance of any two codewords = $\min_{x,y \in \mathcal{C}} d_H(x, y)$

Code distance

- each block code is an *injective* function $C : \{0,1\}^k \to \{0,1\}^n$ ($m \neq m' \implies C(m) \neq C(m')$)
- define $\mathcal{C}$ = set of all codewords = $C(\{0,1\}^k)$

$\rightsquigarrow$ $\mathcal{C} \subseteq \{0,1\}^n$ $\quad$ $|\mathcal{C}| = 2^k$ out of 2^n n-bit strings are valid codewords

- decoding = finding closest valid codeword
- ***distance of code:***
  d = minimal Hamming distance of any two codewords = $\min_{x,y \in \mathcal{C}} d_H(x,y)$



Implications for codes

1. Need distance d to **detect** all errors flipping up to $d-1$ bits.
2. Need distance d to **correct** all errors flipping up to $\left\lfloor \frac{d-1}{2} \right\rfloor$ bits.

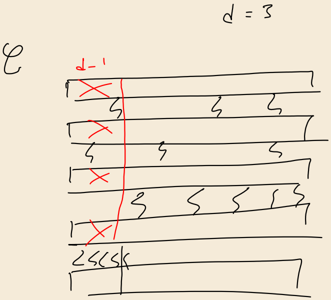
Lower Bounds

- Main advantage of concept of code distance:
  can *prove* lower bounds on block length

Lower Bounds

- Main advantage of concept of code distance:
  can *prove* lower bounds on block length

- ***Singleton bound:*** $2^k \le 2^{n-(d-1)} \rightsquigarrow \boxed{n \ge k + d - 1}$
  - *proof sketch:* We have 2^k codeswords with distance d
    after deleting the first $d-1$ bits, all are still distinct
    but there are only $2^{n-(d-1)}$ such shorter bitstrings.

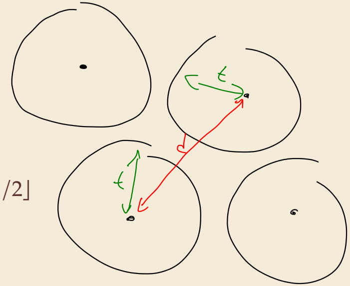


Lower Bounds

- Main advantage of concept of code distance:
  can *prove* lower bounds on block length

- ***Singleton bound:*** $2^k \le 2^{n-(d-1)} \rightsquigarrow n \ge k + d - 1$
  - *proof sketch:* We have 2^k codeswords with distance d
    after deleting the first $d-1$ bits, all are still distinct
    but there are only $2^{n-(d-1)}$ such shorter bitstrings.

- ***Hamming bound:*** $2^k \le \dfrac{2^n}{\sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}}$ ← size of ball
  - *proof idea:* consider "balls" of bitstrings around codewords
    count bitstrings with Hamming-distance $\le t = \lfloor (d-1)/2 \rfloor$
    correcting t errors means all these balls are disjoint
    so $2^k \cdot \text{ball size} \le 2^n$

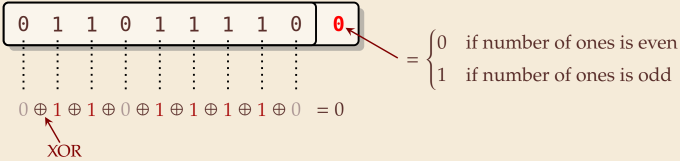


$\rightsquigarrow$ We will come back to these.

8.3 Hamming Codes

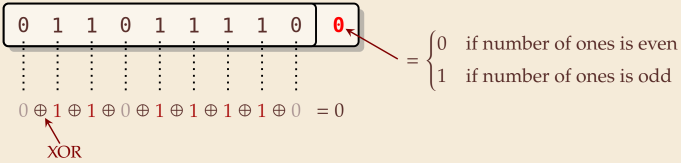
Parity Bit

- simplest possible error-detecting code: add a **parity bit**



Parity Bit

- simplest possible error-detecting code: add a **parity bit**



⇝ code distance 2

- can detect any single-bit error (actually, any odd number of flipped bits)
- used in many hardware (communication) protocols
  - PCI buses, serial buses
  - caches
  - early forms of main memory

Parity Bit

- simplest possible error-detecting code: add a **parity bit**

`0 1 1 0 1 1 1 1 0 | 0`

$$= \begin{cases} 0 & \text{if number of ones is even} \\ 1 & \text{if number of ones is odd} \end{cases}$$

$$0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

XOR

⇝ code distance 2

- can detect any single-bit error (actually, any odd number of flipped bits)
- used in many hardware (communication) protocols
  - PCI buses, serial buses
  - caches
  - early forms of main memory

very simple and cheap

cannot correct any errors

Clicker Question

What do you think, how many extra bits do we need to **detect** a **single bit error** in a message of 100 bits?

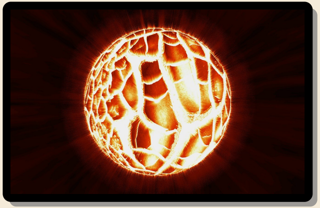
→ sli.do/comp526

Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!

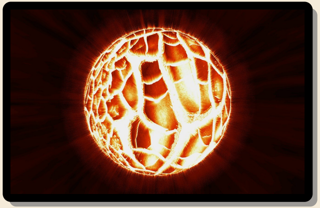
    https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2

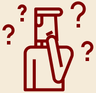


Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!
    `https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2`



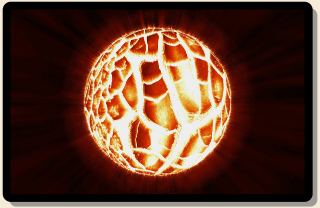


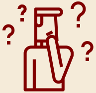
Can we **correct** a bit error without knowing where it occurred? How?

Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!
    https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2





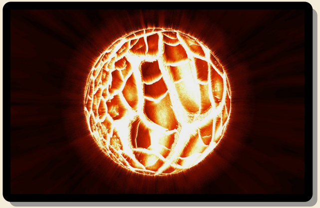
Can we **correct** a bit error without knowing where it occurred? How?

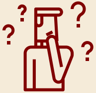
- Yes! store every bit *three times!*
  - upon read, do majority vote
  - if only one bit flipped, the other two (correct) will still win

Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!
    `https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2`





Can we **correct** a bit error without knowing where it occurred? How?

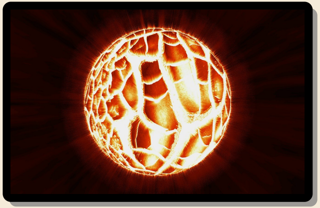
- Yes! store every bit *three times!*
  - upon read, do majority vote
  - if only one bit flipped, the other two (correct) will still win

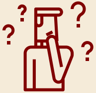
triples the cost!

You want WHAT!?!

Error-correcting codes

- typical application: heavy-duty server RAM (any downtime is expensive!)
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!
    https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2





Can we **correct** a bit error without knowing where it occurred? How?

- Yes! store every bit *three times!*
  - upon read, do majority vote
  - if only one bit flipped, the other two (correct) will still win
- *triples* the cost!

You want WHAT!?!

Can do it with 11% extra memory! (instead of 200% (!))

How to locate errors?

- **Idea:** Use several parity bits
  - each covers a **subset** of bits
  - clever subsets $\rightsquigarrow$ violated/valid parity bit pattern narrows down error

How to locate errors?

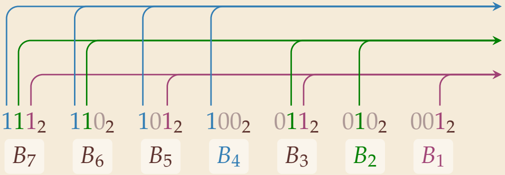
- **Idea:** Use several parity bits
  - each covers a **subset** of bits
  - clever subsets $\rightsquigarrow$ violated/valid parity bit pattern narrows down error

⚠ flipped bit can be one of the parity bits!

How to locate errors?

- **Idea:** Use several parity bits
  - each covers a **subset** of bits
  - clever subsets ⇝ violated/valid parity bit pattern narrows down error
  - ⚠ flipped bit can be one of the parity bits!

- Consider $n = 7$ bits $B_1, \ldots, B_7$ with the following constraints:



$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$

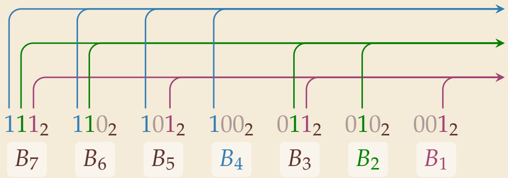
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$

$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \stackrel{!}{=} 0$$

How to locate errors?

- **Idea:** Use several parity bits
  - each covers a **subset** of bits
  - clever subsets $\rightsquigarrow$ violated/valid parity bit pattern narrows down error
  - ⚠ flipped bit can be one of the parity bits!

- Consider $n = 7$ bits $B_1, \ldots, B_7$ with the following constraints:



$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \stackrel{!}{=} 0$$

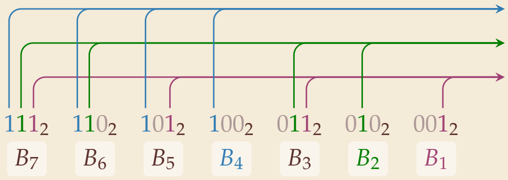
Observe:

- No error (all 7 bits correct) $\rightsquigarrow$ $C = C_2C_1C_0 = 000_2 = 0$ ✓
- What happens if (exactly) 1 bit, say B_i flips?

How to locate errors?

- **Idea:** Use several parity bits
  - each covers a **subset** of bits
  - clever subsets $\rightsquigarrow$ violated/valid parity bit pattern narrows down error
  - ⚠ flipped bit can be one of the parity bits!

- Consider $n = 7$ bits $B_1, \ldots, B_7$ with the following constraints:



$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \stackrel{!}{=} 0$$

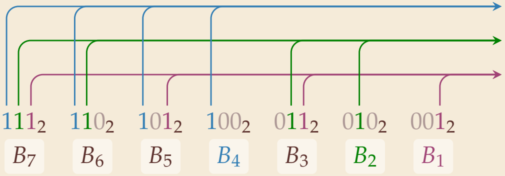
Observe:

- No error (all 7 bits correct) $\rightsquigarrow$ $C = C_2C_1C_0 = 000_2 = 0$ ✓
- What happens if (exactly) 1 bit, say B_i flips?

$C_j = 1$ iff jth bit in binary representation of i is 1

How to locate errors?

- **Idea:** Use several parity bits
  - each covers a **subset** of bits
  - clever subsets $\rightsquigarrow$ violated/valid parity bit pattern narrows down error
  - ⚠ flipped bit can be one of the parity bits!
- Consider $n = 7$ bits $B_1, \ldots, B_7$ with the following constraints:



$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \overset{!}{=} 0$$

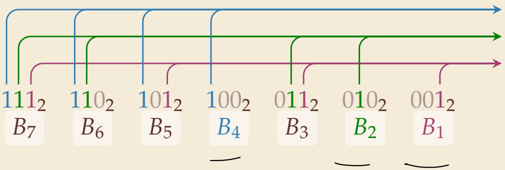
Observe:

- No error (all 7 bits correct) $\rightsquigarrow$ $C = C_2C_1C_0 = 000_2 = 0$ ✓
- What happens if (exactly) 1 bit, say B_i flips?

$C_j = 1$ iff jth bit in binary representation of i is 1 $\rightsquigarrow$ *C encodes **position of error**!*

(7, 4) Hamming Code

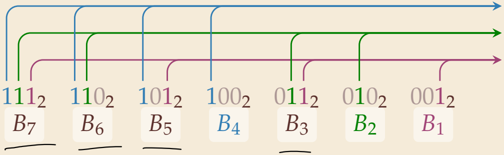
- *How can we turn this into a code?*



$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \stackrel{!}{=} 0$$

(7, 4) Hamming Code

- *How can we turn this into a code?*

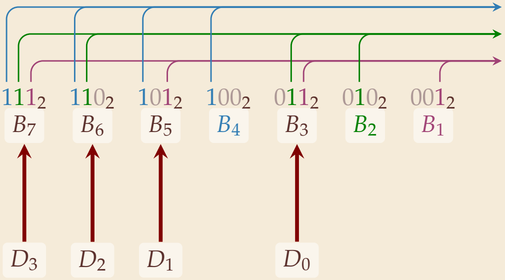


$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \overset{!}{=} 0$$

- B_4, B_2 and B_1 occur only in one constraint each $\rightsquigarrow$ **define** them based on rest!
- **(7, 4)** ***Hamming Code*** **– Encoding**
  1. **Given:** message $D_3D_2D_1D_0$ of length $k = 4$

(7, 4) Hamming Code

- *How can we turn this into a code?*

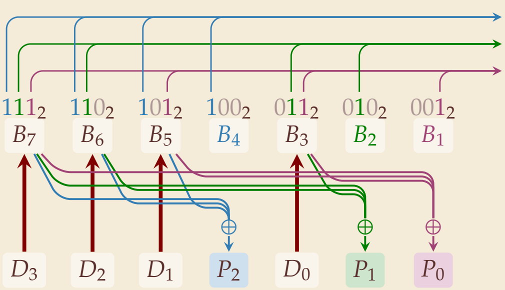


$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \overset{!}{=} 0$$

- B_4, B_2 and B_1 occur only in one constraint each $\rightsquigarrow$ **define** them based on rest!
- **(7, 4) *Hamming Code* – Encoding**
  1. **Given:** message $D_3D_2D_1D_0$ of length $k = 4$
  2. copy $D_3D_2D_1D_0$ to $B_7B_6B_5B_3$

(7, 4) Hamming Code

- *How can we turn this into a code?*



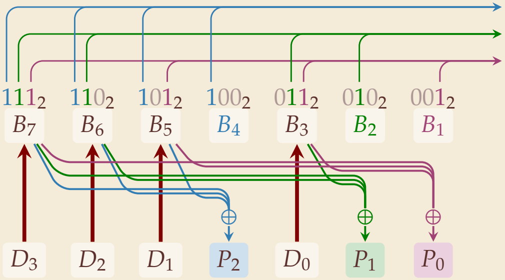
$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \stackrel{!}{=} 0$$

$$P_2 = D_3 \oplus D_2 \oplus D_1$$
$$P_1 = D_3 \oplus D_2 \oplus D_0$$
$$P_0 = D_3 \oplus D_1 \oplus D_0$$

- B_4, B_2 and B_1 occur only in one constraint each $\rightsquigarrow$ **define** them based on rest!
- **(7, 4) *Hamming Code* – Encoding**
  1. **Given:** message $D_3D_2D_1D_0$ of length $k = 4$
  2. copy $D_3D_2D_1D_0$ to $B_7B_6B_5B_3$
  3. compute $P_2P_1P_0 = B_4B_2B_1$ so that $C = 0$

(7, 4) Hamming Code

- *How can we turn this into a code?*



$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \stackrel{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \stackrel{!}{=} 0$$

$$P_2 = D_3 \oplus D_2 \oplus D_1$$
$$P_1 = D_3 \oplus D_2 \oplus D_0$$
$$P_0 = D_3 \oplus D_1 \oplus D_0$$

- B_4, B_2 and B_1 occur only in one constraint each $\rightsquigarrow$ **define** them based on rest!
- **(7, 4) *Hamming Code* – Encoding**
  1. **Given:** message $D_3D_2D_1D_0$ of length $k = 4$
  2. copy $D_3D_2D_1D_0$ to $B_7B_6B_5B_3$
  3. compute $P_2P_1P_0 = B_4B_2B_1$ so that $C = 0$
  4. send $D_3D_2D_1P_2D_0P_1P_0$

(7, 4) Hamming Code – Decoding

- $(7,4)$ ***Hamming Code*** **– Decoding**
  1. **Given:** block $B_7B_6B_5B_4B_3B_2B_1$ of length $n = 7$
  2. compute C (as above)
  3. if $C = 0$ no (detectable) error occurred
     otherwise, flip B_C (the Cth bit was twisted)
  4. return 4-bit message $B_7B_6B_5B_3$

Clicker Question

to correct t bit (1 bit)
need $2t+1$ distance

What is the code distance of $(7,4)$ Hamming code?

3

→ sli.do/comp526

(7, 4) Hamming Code – Properties

- **Hamming bound:**
  - 2^4 valid 7-bit codewords (on per message)
  - any of the 7 single-bit errors corrected towards valid codeword
  - $\rightsquigarrow$ each codeword covers 8 of all possible 7-bit strings
  - $2^4 \cdot 2^3 = 2^7 \quad \rightsquigarrow \quad$ exactly cover space of 7-bit strings

(7, 4) Hamming Code – Properties

- **Hamming bound:**
  - 2^4 valid 7-bit codewords (on per message)
  - any of the 7 single-bit errors corrected towards valid codeword
  - ⇝ each codeword covers 8 of all possible 7-bit strings
  - $2^4 \cdot 2^3 = 2^7$ ⇝ exactly cover space of 7-bit strings

- distance $d = 3$
- can *correct* any 1-bit error

(7, 4) Hamming Code – Properties

- **Hamming bound:**
  - 2^4 valid 7-bit codewords (on per message)
  - any of the 7 single-bit errors corrected towards valid codeword

  $\rightsquigarrow$ each codeword covers 8 of all possible 7-bit strings
  - $2^4 \cdot 2^3 = 2^7$ $\rightsquigarrow$ exactly cover space of 7-bit strings

- distance $d = 3$
- can *correct* any 1-bit error
- How about 2-bit errors?
  - We can *detect* that *something* went wrong.
  - **But:** above decoder mistakes it for a (different!) 1-bit error and "corrects" that
  - Variant: store one additional parity bit for entire block

  $\rightsquigarrow$ Can *detect* any 2-bit error, but *not correct* it.

Hamming Codes – General recipe

- construction can be generalized:
  - Start with $n = 2^\ell - 1$ bits for $\ell \in \mathbb{N}$ (we had $\ell = 3$)
  - use the ℓ bits whose index is a power of 2 as parity bits
  - the other $n - \ell$ are data bits

Hamming Codes – General recipe

- construction can be generalized:
  - Start with $n = 2^\ell - 1$ bits for $\ell \in \mathbb{N}$ (we had $\ell = 3$)
  - use the ℓ bits whose index is a power of 2 as parity bits
  - the other $n - \ell$ are data bits
- Choosing $\ell = 7$ we can encode entire word of memory (64 bit) with 11% overhead (using only 64 out of the 120 possible data bits)

Hamming Codes – General recipe

- construction can be generalized:
  - Start with $n = 2^\ell - 1$ bits for $\ell \in \mathbb{N}$ (we had $\ell = 3$)
  - use the ℓ bits whose index is a power of 2 as parity bits
  - the other $n - \ell$ are data bits
- Choosing $\ell = 7$ we can encode entire word of memory (64 bit) with 11% overhead (using only 64 out of the 120 possible data bits)

👍 simple and efficient coding / decoding

👍 fairly space-efficient

Outlook

- Indeed: $(2^\ell - \ell - 1) + \ell$ Hamming Code is *"perfect"*

= matches Hamming lower bound

$\rightsquigarrow$ cannot use fewer bits . . .

  - **if** message length is $2^\ell - \ell - 1$ for $\ell \in \mathbb{N}_{\geq 2}$
    i. e., one of $1, 4, 11, 26, 57, 120, 247, 502, 1013, \ldots$
  - **and** we want to correct 1-bit errors

Outlook

- Indeed: $(2^\ell - \ell - 1) + \ell$ Hamming Code is *"perfect"*

= matches Hamming lower bound

$\rightsquigarrow$ cannot use fewer bits . . .

  - **if** message length is $2^\ell - \ell - 1$ for $\ell \in \mathbb{N}_{\geq 2}$
    i. e., one of $1, 4, 11, 26, 57, 120, 247, 502, 1013, \ldots$
  - **and** we want to correct 1-bit errors

- For other scenarios, finding good codes is an active research area
  - information theory predicts that *almost all* randomly chosen codes are good(!)
  - but these are inefficient to decode

  $\rightsquigarrow$ clever tricks and constructions needed