

EFFICIENT ALGORITHMS \$EFFICIENT
SCIENTIALGORITHMS\$EFFI
EFFICIENTALGORITHMS\$
ENTALGORITHMS\$EFFICI
FFICIENTALGORITHMS\$E
FICIENTALGORITHMS\$EF
GORITHMS\$EFFICIENTAL
HMS\$EFFICIENTALGORIT
IENTALGORITHMS\$EFFIC
ITMS\$EFFICIENTALGOR
LORITHMS\$EFFICIENTALGO
NTALGORITHMS\$EFFICIENTALG
ORITHMS\$EFFICIENTALGO
RITHMS\$EFFICIENTALGO
S\$EFFICIENTALGORITHM
TALGORITHMS\$EFFICIENT
THMS\$EFFICIENTALGORI

6

Error-Correcting Codes

17 November 2023

Sebastian Wild

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 6: *Error-Correcting Codes*



6 Error-Correcting Codes

6.1 Introduction

6.2 Lower Bounds

6.3 Hamming Codes

6.1 Introduction

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

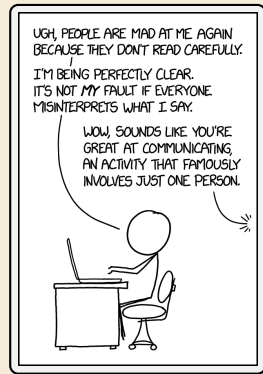


- ▶ But how is it possible (for us) to decode a message in the presence of noise & errors?

*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 corrected 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



- ▶ 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!

6.2 Lower Bounds

Block codes

► model:

- want to send message $S \in \{0, 1\}^*$ (bitstream) across a (*communication*) channel
- any bit transmitted through the channel might *flip* ($0 \rightarrow 1$ resp. $1 \rightarrow 0$)
no other errors occur (no bits lost, duplicated, inserted, etc.)
- instead of S , we send *encoded bitstream* $C \in \{0, 1\}^*$
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 ($k = \text{message length}$)
- encode each message (separately) as $C(m) \in \{0, 1\}^n$ ($n = \text{block length}$, $n \geq k$)

↪ can analyze everything block-wise

► between 0 and n bits might be flipped invalid code

- how many flipped bits can we definitely **detect**?
- 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 $C : \{0, 1\}^k \rightarrow \{0, 1\}^n$

▶ 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 = $\min_{x, y \in \mathcal{C}} d_H(x, y)$

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 $\lfloor \frac{d-1}{2} \rfloor$ bits.

Lower Bounds

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

otherwise no such code exists

Given block length n , message length k , code distance d , we must have:

- ▶ **Singleton bound:** $2^k \leq 2^{n-(d-1)} \rightsquigarrow n \geq k + d - 1$

- ▶ *proof sketch:* We have 2^k codewords 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:** $2^k \leq \frac{2^n}{\sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}}$

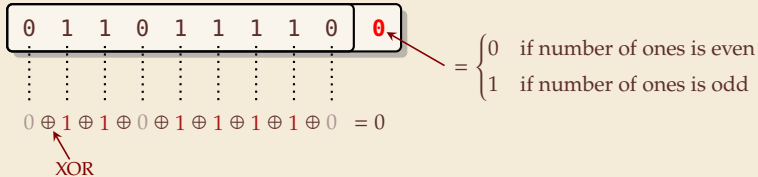
- ▶ *proof idea:* consider “balls” of bitstrings around codewords
count bitstrings with Hamming-distance $\leq t = \lfloor (d - 1)/2 \rfloor$
correcting t errors means all these balls are disjoint
so $2^k \cdot \text{ball size} \leq 2^n$

\rightsquigarrow We will come back to these.

6.3 Hamming Codes

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

👍 very simple and cheap

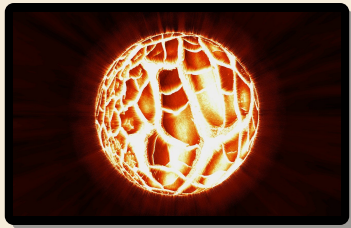
👎 cannot correct any errors

Error-correcting codes

- ▶ 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>

any downtime is expensive!



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!?!

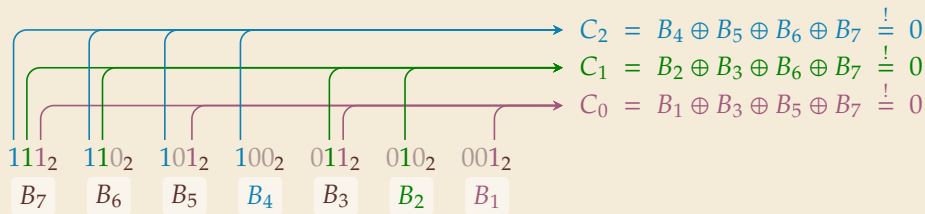


instead of 200% (!)

Can do it with 11% extra memory!

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, \dots, B_7 with the following constraints:



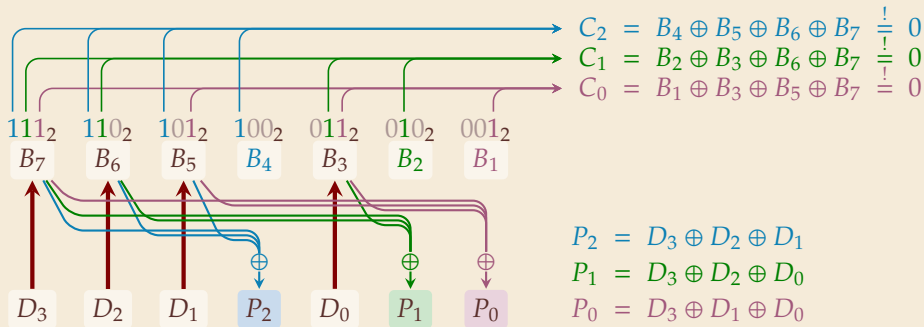
Observe:

- ▶ 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 j th 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?



► 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 C th bit was twisted)
4. return 4-bit message $B_7B_6B_5B_3$

(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


▶ 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
- ↪ 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
- ▶ 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: $(2^\ell - 1, 2^\ell - \ell - 1)$ Hamming Code is “perfect” code

↪ cannot use fewer bits ...

= matches Hamming lower bound

- ▶ 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, ...
 - ▶ **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
- ↪ clever tricks and constructions needed