TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Faculty of Computer Science, Institute of Systems Architecture, Chair of Systems Engineering

RoboLab Autumn Course – Hamming Codes

by M.Sc. Samuel Knobloch

What are Hamming Codes?

- Used in telecommunication and other fields for error-resilient data transmission
- Belongs to category of linear block codes of Forward Error Correcting (FEC) codes
  - ➔ Hamming Codes can detect up to two bit errors or
  - ➔ Correct one bit error without detecting other, uncorrected, errors
- Hamming Codes are considered perfect codes
  - ➔ Why? Best rate for codes for block length and minimum distance of 3
  - ➔ And why the distance of 3?
    - If t errors can be corrected: $2t+1$ = minimum distance
- Hamming Codes are invented in 1950 by Richard Wesley Hamming for punched card readers

Minimum Distance - Hamming Distance

- Hamming Distance describes variety of strings or codes

  ➔ Given two codes with a fixed length, the Hamming Distance *h* is the number of varying positions

  ```
  a = 00101, b = 01110   =>   To generate b from a, 3 bits needs to be changed
                         =>   h = 3
  ```

- General calculation:

  ```
  h = t + 1      for t errors detectable
  h = 2t + 1     for t errors correctable
  ```

- What is achievable:

h	*t errors detectable*	*t errors correctable*
1	0	0
2	1	0
3	2	1
4	3	1

Theory behind Hamming Codes

- Given a 4-bit code, we extend this code with a specific number of parity (or redundancy) bits

  ➔ General calculation: $2^d \geq d + l + 1$

  l … Length of original code

  ➔ Solution:
  $2^1 \geq 1 + 4 + 1 \Rightarrow 2 \geq 6$
  $2^2 \geq 2 + 4 + 1 \Rightarrow 4 \geq 7$
  $2^3 \geq 3 + 4 + 1 \Rightarrow 8 \geq 8$

  So, for our code we need r=*3* parity bits resulting in the standard Hamming Code $(7,4)$.

  Longer signature version: $(n,k,d) \Rightarrow (7,4,3)$

Standard Hamming Code (7,4)

- As calculated before, we now have *4 message bits + 3 parity bits = 7-bit Hamming Code*

- Now, for representing this linear block code there are two forms:

  ➔ Systematic form (e.g. *0101**101***)

  This means, the parity bits are simply added after the original code

  ➔ Non-systematic form (e.g. *01**10011***)

  This means, the parity bits are part of the whole code and cover the data bits

- For illustration we'll go through a short systematic example on the next slides

Standard Hamming Code (7,4) - Systematic

- Given an example code: *1011*
  - ➔ Codeword representation: $D_1\ D_2\ D_3\ D_4\ P_1\ P_2\ P_4$ with
    D_1, D_2, D_3, D_4 data and
    P_1, P_2, P_4 redundant bits
  - ➔ How to calculate *P* using (binary) *XOR*:

$$P_1 = D_1 \oplus D_2 \oplus D_4 \Rightarrow P_1 = 1 \oplus 0 \oplus 1 \Rightarrow P_1 = 0$$
$$P_2 = D_1 \oplus D_3 \oplus D_4 \Rightarrow P_2 = 1 \oplus 1 \oplus 1 \Rightarrow P_2 = 1$$
$$P_4 = D_2 \oplus D_3 \oplus D_4 \Rightarrow P_4 = 0 \oplus 1 \oplus 1 \Rightarrow P_4 = 0$$

  - ➔ Resulting codeword: *1011**010***

- Now, we have converted the 4-bit code into a 7-bit codeword.
  So, out of 128 (2^7) combinations only 16 are valid codewords.
  - ➔ Any other combination will result in an error!
- Hamming distance is 3, so we can *detect two-bit* errors **or** *correct* a single *one*

Standard Hamming Code (7,4) - Systematic

- How to detect and correct a single-bit error?
  - ➔ Given the received codeword *1011110*, we need to calculate the position of the error:

$$A = P_1 \oplus D_1 \oplus D_2 \oplus D_4 \quad \Rightarrow \quad A = 1 \oplus 1 \oplus 0 \oplus 1 \Rightarrow \quad A = 1$$
$$B = P_2 \oplus D_1 \oplus D_3 \oplus D_4 \quad \Rightarrow \quad B = 1 \oplus 1 \oplus 1 \oplus 1 \Rightarrow \quad B = 0$$
$$C = P_4 \oplus D_2 \oplus D_3 \oplus D_4 \quad \Rightarrow \quad C = 0 \oplus 0 \oplus 1 \oplus 1 \Rightarrow \quad C = 1$$

  - ➔ Resulting check code (also Syndrome): $101_b = 5_d$

    So, the error should be at position 5: *1011**1**10*
  - ➔ Flip the incorrect bit, we have our correct codeword: *1011010*

- Other approach: Using matrices (see Assignment 1)
  - ➔ Parity-check matrix: $H := (A \mid I_{n-k})$
    Generator matrix: $G := (I_k \mid -A^T)$

Hamming Code – Extension „SECDED“

- *SECDED*? Single Error Correction Double Error Detection
  - ➔ By adding another parity bit we can now:
    - Distinguish a double bit error in a single codeword from single-bit errors in different codewords
    - Detect two errors **and** correct one
- Two ways (adding at *most-significant-bit* if not defined otherwise):
  - ➔ Even Parity – Count all **1** in your codeword and if odd add another 1, else 0
  - ➔ Odd Parity – Count all **1** in your codeword and if even add another 1, else 0
- Taking our example *1100110* and adding an extra parity bit, we get:
  - ➔ For even parity: *10110100*
    For odd parity: *10110101*
  - ➔ Resulting signature (extended): `(8,4)`

Hamming Code – Examples

- List of possible Hamming Codes (incomplete):

Parity bits	Total bits	Data bits	Name
2	3	1	Hamming (3, 1)
3	7	4	Hamming (7, 4)
4	15	11	Hamming (15, 11)
5	31	26	Hamming (31, 26)
6	63	57	Hamming (63, 57)
7	127	120	Hamming (127, 120)
8	255	247	Hamming (255, 247)

Assignment 1 – Hamming Code (Theory)

Assignment 1 – Hamming Code (Theory)

- Task 1
  - ➔ Create both the generator and parity-check matrix for a non-systematic Hamming Code
- Task 2
  - ➔ Create both the generator and parity-check matrix for a systematic Hamming Code
- Task 3
  - ➔ Use your matrices generated in Task 2 and encode the given codewords
- Task 4
  - ➔ Use your matrices generated in Task 1 and decode given codewords
  - ➔ Correct errors if possible and write down to original source word

Assignment – First steps

- **Create** a **Gitlab account** using your **TU-Dresden login** (ZIH/Selma)
  - ➔ Check your email account for the confirmation mail (https://msx.tu-dresden.de)
- Follow „**Getting started**“ from our documentation
  - ➔ Fork the repository „***robolab-assignments***“ into your own workspace
  - ➔ Grant **Read** access to the user “**RoboLab**”
- **Commit** the assignment **solution into** the directory „**pdf-files**“