Computer Science Fundamentals

Channel Coding – Fault-tolerant Codes / Introduction

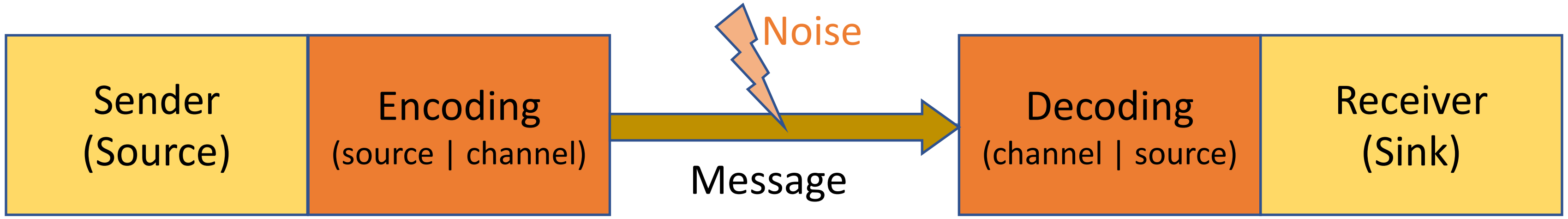
Technische Hochschule Rosenheim
Winter 2021/22
Prof. Dr. Jochen Schmidt

Overview

- Motivation
- Hamming distance
- m-out-of-n codes
- Noisy-channel coding theorem
- Parity check

Motivation

Technische Hochschule Rosenheim



- Noisy channels may randomly alter single or multiple bits during transmission
- A channel may be, e.g.,
    - cables connecting devices, e.g., USB, network, HDMI, CAN, …
    - wireless transmissions using protocols like Wifi, Bluetooth, …
    - devices storing information, e.g., hard drives, SSDs, RAM, …

⟶ We need to make messages robust to noise

This is called channel coding (*Kanalcodierung*)

Motivation

Technische Hochschule Rosenheim

- Channel coding follows source coding:
  - source coding: minimize redundancy (compression)
  - channel coding: systematically add redundancy to cope with errors

- We are looking for fault-tolerant codes
  - that allow the recipient to recognize whether an error occurred during transmission
  - and, if so, maybe even to correct it
- Error-detecting codes – errors can be detected by recipient
- Error-correcting codes – errors can be corrected by recipient

Can we measure how fault-tolerant a binary code is? YES: Hamming distance

Hamming Distance

- Hamming Distance of two strings of equal length
  - number of symbols that are different between the strings
  - we usually use the binary alphabet and compare binary code words
  - the distance between code words of variable length us undefined (is this a problem with Huffman?)

- Hamming Distance of a code
  - minimum value of all pairwise (mutually distinct) distances of all code words
  - measure for robustness against noise

Hamming Distance – Example

Decimal digits 1 to 4 coded directly in binary: 1 = `001`, 2 = `010`, 3 = `011`, 4 = `100`

	`001`	`010`	`011`	`100`
`001`	-	-	-	-
`010`	2	-	-	-
`011`	1	1	-	-
`100`	2	2	3	-

Hamming distance of code = 1

Fault-tolerance Capabilities

- For a given Hamming distance h of a code holds:
  - If a maximum of $h - 1$ bits are incorrect in a code word, this can be detected
  - If a maximum of $(h - 1)/2$ bits are incorrect, these errors can be corrected

or in other words:

- If a code has the Hamming distance h, all errors can be
  - detected that affect less than h bits
  - corrected that affect less than $h/2$ bits

  this is mutually exclusive!

- Depending on the maximum number k of incorrectly transmitted bits that can be automatically detected or corrected in a code, we call it k-error-detecting or k-error-correcting code.

Fault-tolerance Capabilities

Technische Hochschule Rosenheim

- $h = 1$
  - erroneous bits cannot be detected (e.g., ASCII)
- $h = 2$
  - 1-bit errors can be detected but not corrected (e.g., parity check)
- $h = 3$
  - 1-bit errors can be corrected (e.g., Hamming codes) OR
  - 1-bit and 2-bit errors can be detected but not corrected
- $h = 4$
  - 1-bit errors can be corrected, and 2-bit errors can be detected OR
  - 1-bit, 2-bit, and 3-bit errors can be detected but not corrected
- $h = 5$
  - 1-bit and 2-bit errors can be corrected OR
  - 1-bit, 2-bit, 3-bit, and 4-bit errors can be detected but not corrected

Hamming Distance – Example

Decimal digits 1 to 4 coded directly in binary:
1 = **001**, 2 = **010**, 3 = **011**, 4 = **100**

	001	**010**	**011**	**100**
001	-	-	-	-
010	2	-	-	-
011	1	1	-	-
100	2	2	3	-

Hamming distance of code $h = 1$
$\longrightarrow$ we cannot guarantee that
all errors will be detected

Decimal digits 1 to 4 coded differently:
1 = **000**, 2 = **011**, 3 = **101**, 4 = **110**

	000	**011**	**101**	**110**
000	-	-	-	-
011	2	-	-	-
101	2	2	-	-
110	2	2	2	-

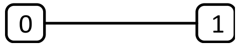
Hamming distance of code $h = 2$
$\longrightarrow$ we can guarantee that
all 1-bit errors will be detected

Linear Codes

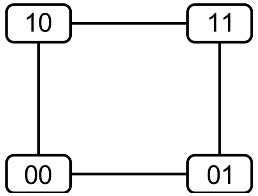
- Consider a block code of length r:
  - the code words are elements of a vector space $\mathbb{F}_q^r$ defined over a finite field $\mathbb{F}_q$
  - often (but not always), we will use binary codes, i.e., the field $\mathbb{F}_2$ containing only 0 and 1
- For binary codes, we get the vector space
  - with operations
    - AND = scalar product (“normal” multiplication)
    - XOR = vector addition (XOR = bitwise mod 2)
  - the neutral elements
    - One = 1 (scalar)
    - Zero = (0 0 … 0) (vector)
  - inverse elements regarding XOR: $-x = x$ (self-inverse, involution)
- A code $C \subseteq \mathbb{F}_q^r$ is called linear if C is a subspace of $\mathbb{F}_q^r$
  - i.e., C is itself a vector space
  - in particular this implies that C contains Zero and is algebraically closed wrt. vector addition and scalar multiplication
- Most of the codes discussed here will be linear codes
- Linear means in particular:
  - any linear combination of valid code words is also a code word
  - encoding/decoding can be formulated as matrix multiplications
  - we can use the power of linear algebra

Geometric Interpretation of linear Codes

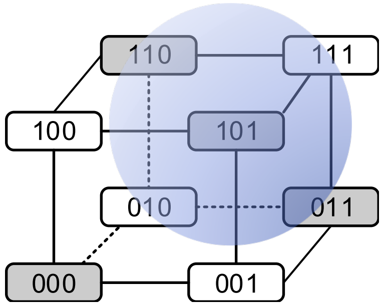
linear code of length r = vector space $\longrightarrow n = 2^r$ possible code words = corners of an r-dimensional (hyper) cube



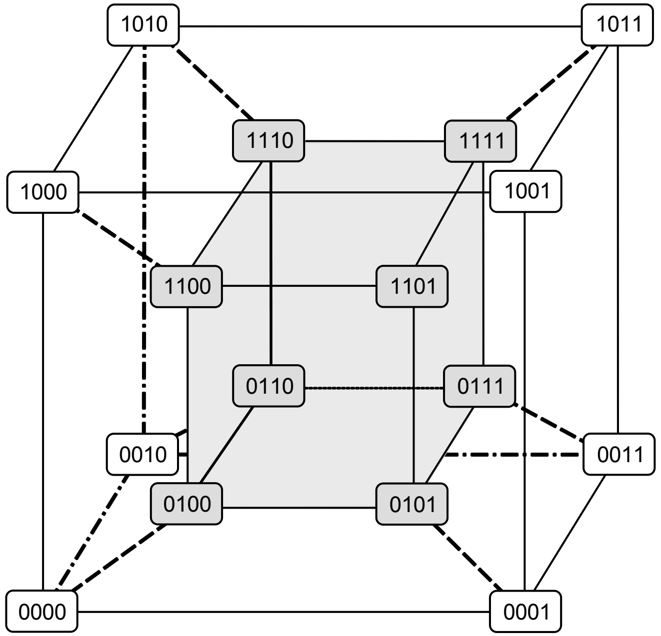
$r = 1$



$r = 2$



$r = 3$



$r = 4$
(this is the projection of a 4-D hypercube to 3-D)

k errors correctable:

- sphere of radius k centered at each code word
- such that they do not overlap
- Hamming distance $h = 2k + 1$

An upper bound n_k for the number of code words with length r in a k-error-correcting code is

$$n_k \leq \frac{2^r}{V_k} = \frac{2^r}{1 + \sum_{i=1}^{k} \binom{r}{i}}$$

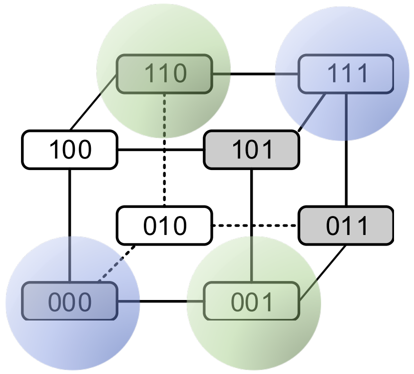
total "volume", i.e., total number of possible code words

"volume" of sphere of radius k

Codes that attain the bound are called perfect codes. They are dense and can always be decoded uniquely.

Perfect Codes – Example

Code length $r = 3$, 1-error-correction code ($k = 1$, Hamming distance $h = 3$)



$$n_k \le \frac{2^3}{V_k} = \frac{8}{1 + \binom{3}{1}} = 2$$

For example:

 {000, 111}

 {001, 110}

Are these linear codes?

Code length $r = 7$, 1-error-correction code ($k = 1$, Hamming distance $h = 3$)

$$n_k \le \frac{2^7}{V_k} = \frac{128}{1 + \binom{7}{1}} = 16$$

We will see an example for such a code soon – the (7, 4) Hamming code

Exercise

Let's consider binary block codes of length 9 bit:

1. If we use each possible combination in the resulting code:
   a) How many code words are available?
   b) What is the Hamming distance of such a code?
   c) How many erroneous bits can be detected or corrected?

2. If we want to use a 2-error-correcting code:
   a) What Hamming distance is required?
   b) What is the upper bound for the number of code words that are available in such a code?

m-out-of-n (m-oo-n) Codes

- (Nonlinear) block codes with a word length of n

- Each code word contains exactly
  - m **Ones** and
  - n – m **Zeros**
- Special case: 1-out-of-n code: „one-hot“ coding

- The code contains exactly $\binom{n}{m}$ code words

Examples:

Digit	2-oo-5 code	1-oo-10 code
0	00011	0000000001
1	00101	0000000010
2	00110	0000000100
3	01001	0000001000
4	01010	0000010000
5	01100	0000100000
6	10001	0001000000
7	10010	0010000000
8	10100	0100000000
9	11000	1000000000

Parity Codes

Technische Hochschule Rosenheim

- Parity Check (*Paritätsprüfung*)
- Widely used for error detection
- Idea
  - Add an additional bit to an existing block code (the parity bit) such that
  - the total number of ones in the code words are
    - even (even parity, *gerade Parität*) or
    - odd (odd parity, *ungerade Parität*)

Parity Codes – Example

Technische Hochschule Rosenheim

7 bit ASCII code with added even parity

A 10000010	G 10001110	M 10011010	S 10100110	Y 10110010
B 10000100	H 10010000	N 10011100	T 10101001	Z 10110100
C 10000111	I 10010011	O 10011111	U 10101010	
D 10001000	J 10010101	P 10100000	V 10101100	
E 10001011	K 10010110	Q 10100011	W 10101111	
F 10001101	L 10011001	R 10100101	X 10110001	

from: H. Herold, B. Lurz, J. Wohlrab, M. Hopf. ***Grundlagen der Informatik***

Parity Codes – Example

Technische Hochschule Rosenheim

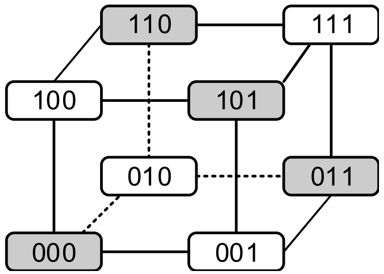
We have received the following sequence of 7 bit ASCII characters (+ even parity bit on right):

1001000011001010110110001101100011011110010000011010111011011110111001001101100011101001

10010000	H
11001010	e
11011000	l
11011000	l
11011110	o
01000001	(space)
10101110	W
11011110	o
11100100	r
11011000	l
11101001	← **Parity error!**

Even Parity Codes are Linear – Example

All elements of vector space $\mathbb{F}_2^3$:



Gray: Code words for even parity

These form a subspace $C \subset \mathbb{F}_2^3$

Basis: $\left\{ \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right\}$

(you can use any pair of linear independent non-zero vectors, but then the parity bit will get mixed in between the original code words)

⟶ combine basis vectors in matrix to get the generator matrix (*Generatormatrix*)

Generator matrix for C: $\boldsymbol{G} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$

$$\boldsymbol{c} = \boldsymbol{G}\boldsymbol{o}$$

$\boldsymbol{o}$: (original) 2-bit code word without parity bit (as column vector)

$\boldsymbol{c}$: 3-bit code word with even parity bit on the right

Example: $\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

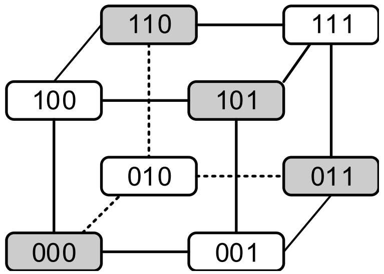
remember:
"+" = mod 2 addition
= XOR of all bits

In many cases you will see

- the transposed generator: $\boldsymbol{G} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$
- multiplication from left: $\boldsymbol{c} = \boldsymbol{o}\boldsymbol{G}$
- row vectors $\begin{pmatrix} 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$

Even Parity Codes are Linear – Example

All elements of vector space $\mathbb{F}_2^3$:



Gray: Code words for even parity

We can check whether a received code word $\boldsymbol{c}$ is correct using the (parity) check matrix $\boldsymbol{H}$.
It is built from the basis vectors of the orthogonal vector space $C^\perp$ to C

$$C^\perp = \{\boldsymbol{v} \mid \boldsymbol{v} \cdot \boldsymbol{u} = 0, \boldsymbol{u} \in C\}$$

In our example: $C^\perp = \left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\}$

This is a 1-D subspace with basis $\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

Check matrix (*Kontrollmatrix*; row vectors): $\boldsymbol{H} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$

By definition we get for correct code words: $\boldsymbol{H}\boldsymbol{c}^T = \boldsymbol{0}$

$\boldsymbol{c}$: (received) 3-bit code word with parity bit (as row vector)

Examples:

$$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = 1 + 1 + 0 = 0$$ $\longrightarrow$ no error detected

$$\begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 0 + 1 + 0 = 1$$ $\longrightarrow$ error detected

remember:
"+" = mod 2 addition
= XOR of all bits

Even Parity Codes are Linear – Example

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \qquad \xrightarrow{\text{negate \& transpose}} \qquad H = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

fill to correct dimension with identity matrix (here: 1x1)

If the first part of $\boldsymbol{G}$ is the identity matrix

- we say that $\boldsymbol{G}$ is in standard form (*systematische Form*)
- $\boldsymbol{H}$ can be determined easily directly from $\boldsymbol{G}$

In general: For an r-dimensional vector space with s-dimensional code subspace C

- $\boldsymbol{G}$ has size $s \times r$
- in standard form $\boldsymbol{G} = (\boldsymbol{I}_{s \times s} \mid \boldsymbol{M})$ ($\boldsymbol{I}_{s \times s}$: $s \times s$ identity matrix; | = concatenation)
- the orthogonal space is $(r - s)$-dimensional
- $\boldsymbol{H}$ has size $(r - s) \times r$
- $\boldsymbol{H} = \left(-\boldsymbol{M}^T \mid \boldsymbol{I}_{(r-s)\times(r-s)}\right)$
  - for binary codes (i.e., in $\mathbb{F}_2$) the “minus” has no effect and can be omitted:
  - $1 + 1 \equiv 0 \bmod 2 \;\longrightarrow\; 1 \equiv -1 \pmod 2$

2-D Parity Check

- Extension of the one-dimensional parity check
- Check 2-D blocks of data
  - a parity bit is used for each individual code word
  - after the entire block of code words is transferred another code word is transferred that contains the parity bits to all columns of the transferred block

2-D Parity Check – Example

2-D parity check

row parity bits

1	1	0	0	0	0	1	1	a
1	1	0	0	0	1	0	1	b
1	1	0	0	0	1	1	0	c
1	1	0	0	1	0	0	1	d
1	1	0	0	1	0	1	0	e
1	1	0	0	1	1	0	0	f
1	1	0	0	1	1	1	1	g
1	1	0	1	0	0	0	1	h
0	0	0	1	0	0	0	1	← column parity bits

2-D Parity Check – Example

If only a single bit changes during transmission, this can be corrected

1	1	0	0	0	0	1	1
1	1	0	0	0	1	0	1
1	1	0	1	0	1	1	0
1	1	0	0	1	0	0	1
1	1	0	0	1	0	1	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	1	1
1	1	0	1	0	0	0	1
0	0	0	1	0	0	0	1

2-D Parity Check – Example

A double error (2 incorrect bits) can be detected

1	1	0	0	0	0	1	1
1	1	0	0	0	1	0	1
1	1	0	1	0	1	1	0
1	1	0	0	1	0	0	1
1	1	0	0	1	1	1	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	1	1
1	1	0	1	0	0	0	1
0	0	0	1	0	0	0	1

4 incorrect parity bits

Correction not possible

1	1	0	0	0	0	1	1
1	1	0	0	0	1	0	1
1	1	0	1	0	1	1	0
1	1	0	0	1	0	0	1
1	1	0	0	1	0	1	0
1	1	0	1	1	1	0	0
1	1	0	0	1	1	1	1
1	1	0	1	0	0	0	1
0	0	0	1	0	0	0	1

2 incorrect parity bits

Correction not possible

2-D Parity Check – Example

A triple error (3 incorrect bits) can be detected

1	1	0	0	0	0	1	1
1	1	0	0	0	1	0	1
1	1	0	1	0	1	1	0
1	1	0	0	1	0	0	1
1	1	0	0	1	1	1	0
1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	1
1	1	0	1	0	0	0	1
0	0	0	1	0	0	0	1

6 incorrect parity bits

Correction not possible

1	1	0	0	0	0	1	1
1	1	0	0	0	1	0	1
1	1	0	1	1	1	1	0
1	1	0	1	1	0	0	1
1	1	0	0	1	0	1	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	1	1
1	1	0	1	0	0	0	1
0	0	0	1	0	0	0	1

2 incorrect parity bits

Correction not possible

Looks like a single error!

2-D Parity Check – Example

Detection of a quadruple error (4 incorrect bits) …

1	1	0	0	0	0	1	1
1	1	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	1	0	0	1	0	0	1
1	1	0	0	1	0	1	0
1	0	0	1	1	1	0	0
1	1	0	0	1	1	1	1
1	1	0	1	0	0	0	1
0	0	0	1	0	0	0	1

… cannot be guaranteed!

2-D Parity Check – Example

- Binary coding of the word INFORMATIK using 7 bit ASCII
  - the total number of ones per character is padded to an even number with a parity bit
  - we use an additional word containing parities for the block after each 4th character
  - in contrast to the previous example the **characters are arranged in columns** rather than rows
    - this is obviously arbitrary, as in reality we have a linear memory layout anyway
    - row/column parities then change their role

- Transfer is performed by splitting the word into blocks
  - 1. and 2. block contain 4 characters
  - 3. block contains only 2 characters and is padded with zeros

- During transmission, 1-bit errors occur and the word ANFORMAPIK is received

2-D Parity Check – Example

Technische Hochschule Rosenheim

Received Data	block parities	Received Data	block parities	Received Data	block parities
1111	0	1111	0	1100	0
0000	0	0000	0	0000	0
0000	0	1001	0	0000	0
0101	1	0100	1	1100	0
0111	1	0100	0	0000	0
0111	1	1000	1	0100	1
1001	0	0110	0	1000	1
1011	1	1001	0	1100	0

Parity bits of single characters

ANFO	RMAP	IK	Received text
I	T		Corrections

1000001 (A)
→ 1001001 (I)

1010000 (P)
→ 1010100 (T)

2-D Parity Check – Summary

- 2-D Parity Check is
  - 1-error-correcting (Correction of single errors and detection of double errors) OR
  - 3-error-detecting (Detection of single, double, and triple errors).
- Disadvantage: We must wait for whole blocks to be transferred before correction
- The concept can be generalized to more dimensions straightforwardly
  - the Hamming distance of a d-dimensional parity check is $d + 1$
  - therefore, a maximum of $d/2$ erroneous bits can be corrected