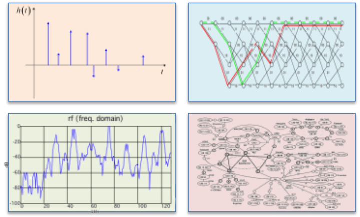


INTRODUCTION TO EECS II
DIGITAL COMMUNICATION SYSTEMS

6.02 Spring 2012
Lecture #4

- Linear block codes – Syndrome Decoding
- Handling bursts

Let's do some rectangular parity decoding

D1	D2	P1
D3	D4	P2
P3	P4	

Received codewords

1	0	1
0	1	0
0	1	

1. Decoded message bits: **1011**

0	0	0
1	1	1
1	1	

2. Decoded message bits: **0011**
P2 parity error

0	0	1
0	1	0
0	0	

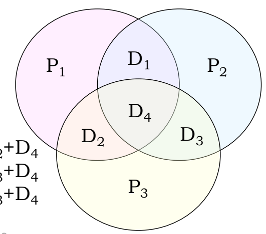
3. Decoded message bits: **0001**
"uncorrectable"

Towards More Efficient Codes: (7,4,3) Hamming Code Example

- Use multiple parity bits, each covering a subset of the data bits.
- No two message bits belong to exactly the same subsets, so a single-bit error will generate a unique set of parity check errors.

Modulo-2 addition, aka XOR

$P_1 = D_1+D_2+D_4$
$P_2 = D_1+D_3+D_4$
$P_3 = D_2+D_3+D_4$



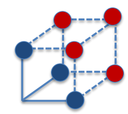
Suppose we check the parity and discover that P1 and P3 indicate an error?
bit D2 must have flipped

What if only P2 indicates an error?
P2 itself had the error!

Logic Behind Hamming Code Construction

- Idea: Use parity bits to cover each axis of the binary vector space
  - That way, all message bits will be covered with a **unique** combination of parity bits

Index	1	2	3	4	5	6	7
Binary index	001	010	011	100	101	110	111
(7,4) code	P1	P2	D1	P3	D2	D3	D4



P_1 with binary index 00**1** covers

$P_1 = D_1+D_2+D_4$
$P_2 = D_1+D_3+D_4$
$P_3 = D_2+D_3+D_4$

D_1 with binary index 01**1**
D_2 with binary index 10**1**
D_4 with binary index 11**1**

Syndrome Decoding: Idea

- After receiving the (possibly corrupted) message, compute a **syndrome** bit (E_i) for each parity bit

$$E_1 = D_1 + D_2 + D_4 + P_1$$
$$E_2 = D_1 + D_3 + D_4 + P_2$$
$$E_3 = D_2 + D_3 + D_4 + P_3$$

$$P_1 = D_1+D_2+D_4$$
$$P_2 = D_1+D_3+D_4$$
$$P_3 = D_2+D_3+D_4$$

- If all the E_i are zero: no errors
- Otherwise the particular combination of the $E_3E_2E_1$ can be used to figure out which bit to correct

Index	1	2	3	4	5	6	7
Binary index	001	010	011	100	101	110	111
(7,4) code	P1	P2	D1	P3	D2	D3	D4

$E_3E_2E_1$	Corrective Action
000	no errors
001	p_1 has an error, flip to correct
010	p_2 has an error, flip to correct
011	d_1 has an error, flip to correct
100	p_3 has an error, flip to correct
101	d_2 has an error, flip to correct
110	d_3 has an error, flip to correct
111	d_4 has an error, flip to correct

Matrix Notation for Linear Block Codes

Task: given **k-bit** message, compute **n-bit** codeword. We can use standard matrix arithmetic (modulo 2) to do the job. For example, here's how we would describe the **(9,4,4)** rectangular code that includes an overall parity bit.

D_1	D_2	P_1
D_3	D_4	P_2
P_3	P_4	P_5

$$d_{1xk} \cdot G_{kxn} = c_{1xn}$$

$$[D_1\ D_2\ D_3\ D_4] \cdot \begin{bmatrix} 1&0&0&0&1&0&1&0&1 \\ 0&1&0&0&1&0&0&1&1 \\ 0&0&1&0&0&1&1&0&1 \\ 0&0&0&1&0&1&0&1&1 \end{bmatrix} = [D_1\ D_2\ D_3\ D_4\ P_1\ P_2\ P_3\ P_4\ P_5]$$

$1 \times k$ message vector; $k \times n$ generator matrix; $1 \times n$ code word vector

The generator matrix $$G_{kxn} = \left[I_{k\times k} \mid A_{k\times(n-k)} \right]$$

Parity Check Matrix

Can restate the codeword generation process as a parity check

For (9,4,4) example

D_1	D_2	P_1
D_3	D_4	P_2
P_3	P_4	P_5

$$H_{(n-k)xn} \cdot c_{1xn}^T = 0$$

$$\begin{bmatrix} 1&1&0&0&1&0&0&0&0 \\ 0&0&1&1&0&1&0&0&0 \\ 1&0&1&0&0&0&1&0&0 \\ 0&1&0&1&0&0&0&1&0 \\ 1&1&1&1&0&0&0&0&1 \end{bmatrix} \cdot \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_5 \end{bmatrix} = 0_{5x1}$$

(n-k) x n parity check matrix; $n \times 1$ code word vector (transpose)

The parity check matrix,

$$H = A^T | I_{(n-k)\times(n-k)}$$

Syndrome Decoding - Matrix Form

Task: given **n-bit** code word, compute **(n-k)** syndrome bits. Again we can use matrix multiply to do the job.

received word $$r_{1xn} = c_{1xn} + e_{1xn}$$ ← 1 x n error vector

compute Syndromes on receive word $$H_{(n-k)xn} \cdot r_{1xn}^T = E_{(n-k)x1}$$ ← (n-k) x 1 syndrome vector

To figure out the relationship of Syndromes to errors:

$$H_{(n-k)xn} \cdot (c_{1xn} + e_{1xn})^T = E_{(n-k)x1} \quad \text{use} \quad H_{(n-k)xn} \cdot c_{1xn}^T = 0$$

$$H_{(n-k)xn} \cdot e_{1xn}^T = E_{(n-k)x1}$$ figure-out error type from Syndrome

Knowing the error patterns we want to correct for, we can compute Syndrome vectors offline and then do a lookup after the Syndrome is calculated from a received word to find the error type that occurred

Syndrome Decoding - Steps

Step 1: For a given code and error patterns $\mathbf{e_i}$, precompute Syndromes and store them

$$H \cdot e_i = E_i$$

Step 2: For each received word, compute the Syndrome

$$H \cdot r = E$$

Step 3: Find $\boldsymbol{l}$ such that $\boldsymbol{E_l} == \boldsymbol{E}$ and apply correction for error $\mathbf{e_l}$

$$c = r + e_l$$

Spot Quiz: Hamming Syndrome Decoding

Find the error in the following received codeword

[D1 D2 D3 D4 P1 P2 P3] = [1 1 1 0 1 1 1]

E_1 = 1+1+0+1 = 1
E_2 = 1+1+0+1 = 1
E_3 = 1+1+0+1 = 1

Syndrome computation:

$E_1 = D_1+D_2+D_4+P_1$
$E_2 = D_1+D_3+D_4+P_2$
$E_3 = D_2+D_3+D_4+P_3$

$E_3E_2E_1$	Corrective Action
000	no errors
001	p_1 has an error, flip to correct
010	p_2 has an error, flip to correct
011	d_1 has an error, flip to correct
100	p_3 has an error, flip to correct
101	d_2 has an error, flip to correct
110	d_3 has an error, flip to correct
111	d_4 has an error, flip to correct

Syndrome Decoding - Steps (9,4,4) example

Codeword generation:

$$\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix} \cdot \begin{bmatrix}1&0&0&0&1&0&1&0&1\\0&1&0&0&1&0&0&1&1\\0&0&1&0&0&1&1&0&1\\0&0&0&1&0&1&0&1&1\end{bmatrix} = \begin{bmatrix}1&1&1&1&0&0&0&0&0\end{bmatrix}$$

Received word in error:

$$\begin{bmatrix}1&0&1&1&0&0&0&0&0\end{bmatrix} = \begin{bmatrix}1&1&1&1&0&0&0&0&0\end{bmatrix} + \begin{bmatrix}0&1&0&0&0&0&0&0&0\end{bmatrix}$$

Syndrome computation for received word

$$\begin{bmatrix}1&1&0&0&1&0&0&0&0\\0&0&1&1&0&1&0&0&0\\1&0&1&0&0&0&1&0&0\\0&1&0&1&0&0&0&1&0\\1&1&1&1&0&0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}1\\0\\1\\1\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\1\\1\end{bmatrix}$$

Precomputed Syndrome for a given error pattern

$$\begin{bmatrix}1&1&0&0&1&0&0&0&0\\0&0&1&1&0&1&0&0&0\\1&0&1&0&0&0&1&0&0\\0&1&0&1&0&0&0&1&0\\1&1&1&1&0&0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}1\\0\\0\\1\\1\end{bmatrix}$$

Syndrome Decoding - Steps (9,4,4) example

Correction:

Since received word Syndrome $[1\ 0\ 0\ 1\ 1]^T$ matches the precomputed Syndrome of the error [0 1 0 0 0 0 0 0 0],
apply this error to the received word to recover the original codeword

Received word

$$\begin{bmatrix}1&1&1&1&0&0&0&0&0\end{bmatrix} = \begin{bmatrix}1&0&1&1&0&0&0&0&0\end{bmatrix} + \begin{bmatrix}0&1&0&0&0&0&0&0&0\end{bmatrix}$$

Corrected codeword

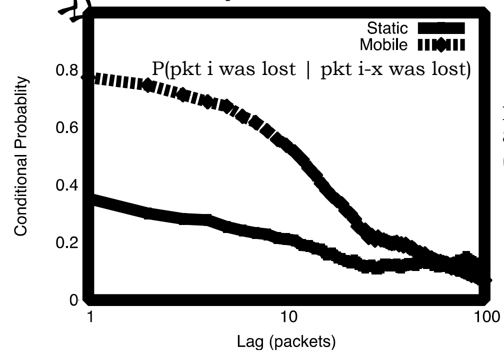
Error pattern from matching Syndrome

Linear Block Codes: Wrap-Up

- (n,k,d) codes have rate k/n and can correct up to floor((d-1)/2) bit errors
- Code words are linear operations over message bits: sum of any two code words is a code word
  - Message + 1 parity bit: (n+1,n,2) code
    - Good code rate, but only 1-bit error detection
  - Replicating each bit c times is a (c,1,c) code
    - Simple way to get great error correction; poor code rate
  - Hamming single-error correcting codes are (n, n-m, 3) where n = 2^m - 1 for m > 1
    - Adding an overall parity bit makes the code (n+1,n-p,4)
  - Rectangular parity codes are (rc+r+c, rc, 3) codes
    - Rate not as good as Hamming codes
- Syndrome decoding: general efficient approach for decoding linear block codes

Burst Errors

Correcting single-bit errors is nice, but in many situations errors come in bursts many bits long (e.g., fading or burst of interference on wireless channel, damage to storage media etc.). How does single-bit error correction help with that?

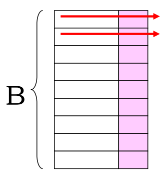


Experiment on 802.11g network (static and mobile)

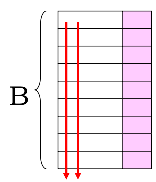
Note: P(pkt lost) ≈ s*P(bit error) where s = pkt size

Coping with Burst Errors by Interleaving

Well, can we think of a way to turn a B-bit error burst into B single-bit errors?



Row-by-row transmission order



Col-by-col transmission order

Problem: Bits from a particular codeword are transmitted sequentially, so a B-bit burst produces multi-bit errors.

Solution: interleave bits from B different codewords. Now a B-bit burst produces 1-bit errors in B different codewords.

Framing

- Looking at a received bit stream, how do we know where a block of interleaved codewords begins?
- Physical indication (transmitter turns on, beginning of disk sector, separate control channel)
- Place a unique bit pattern (frame sync sequence) in the bit stream to mark start of a block
  - Frame = sync pattern + interleaved code word block
  - Search for sync pattern in bit stream to find start of frame
  - Bit pattern can't appear elsewhere in frame (otherwise our search will get confused), so have to make sure no legal combination of codeword bits can accidentally generate the sync pattern (can be tricky…)
  - Sync pattern can't be protected by ECC, so errors may cause us to lose a frame every now and then, a problem that will need to be addressed at some higher level of the communication protocol.

Summary: example channel coding steps

1. Break message stream into k-bit blocks.
2. Add redundant info in the form of (n-k) parity bits to form n-bit codeword. Goal: choose parity bits so we can correct single-bit errors, detect double-bit errors.
3. Interleave bits from a group of B codewords to protect against B-bit burst errors.
4. Add unique pattern of bits to start of each interleaved codeword block so receiver can tell how to extract blocks from received bitstream.
5. Send new (longer) bitstream to transmitter.

011011101101

Step 1: k=4

0110
1110
1101

Step 2: (8,4,3) code

01101111
11100101
11010110

Step 3: B = 3

011111110001100111101110

Step 4: sync = 0111110

01111100111101110001100111100110

Sync pattern has five consecutive 1's. To prevent sync from appearing in message, "bit-stuff" 0's after any sequence of four 1's in the message. This step is easily reversed at receiver (just remove 0 after any sequence of four consecutive 1's in the message).

Summary: example error correction steps

1. Search through received bit stream for sync pattern, extract interleaved codeword block
2. De-interleave the bits to form B n-bit codewords
3. Check parity bits in each code word to see if an error has occurred. If there's a single-bit error, correct it.
4. Extract k message bits from each corrected codeword and concatenate to form message stream.

01111100111101110010000111100101110

Step 1: sync = 0111110

011111110010000111101110

Step 2: B = 3, n = 8

01100111
11110101
11000110

Step 3: (8,4,3) code

010 110 110
101 111 001
11 01 10

Step 4

0110 1110 1101