

Channel Coding
Linda Doyle
CTVR

what we did the last day

- had basic intro block codes
- distinguished between error detection and correction
- defined the Hamming distance and understood consequence for error detection and correction
- we have not yet learned how to generate codes

recall hamming distance

- The maximum number of detectable errors is

$$d_{\min} - 1$$

- That is the maximum number of correctable errors is given by,

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$$

where d_{\min} is the minimum Hamming distance between 2 codewords and $\lfloor \cdot \rfloor$ means the smallest integer

parity codes

- a great deal of block code theory is an extension of the idea of a parity check.
- we are going to look at a simple parity code to get the concepts and then move to a more generalised example

simple parity check

- A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{\min} = 2$.
need this to detect at least 1 error
- The extra bit (parity bit) is to make the total number of 1s in the codeword even [odd parity also possible]
- A simple parity-check code can detect an odd number of errors.

Datawords	Codewords	Datawords	Codewords
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11010
0110	01100	1110	11101
0111	01110	1111	11110

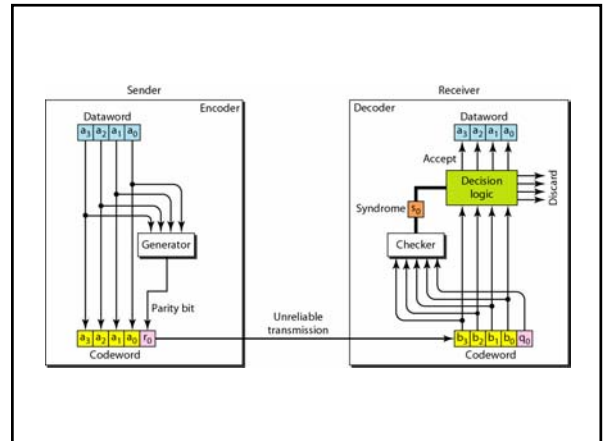
some simple parity check codes. $C(5,4)$, i.e. one redundant bit to make sure that the total number of ones are even

decoding

- To decode
 - Calculate sum of received bits in block (mod 2)
 - If sum is 0 (1) for even (odd) parity then the dataword is the first k bits of the received codeword
 - Otherwise error
- Code can detect single errors
- But cannot correct error since the error could be in any bit

syndrome

- this term is used very often in coding.
- the **syndrome** is the result of a parity check on the received codeword to determine whether it is a valid member of a codeword set.
- if it is a member the syndrome is 0 and if it is not the syndrome is 1 [for even parity]
- the syndrome is used to make the decision whether to discard the code or not



looking in more detail

Assume the sender sends the dataword 1011.

The codeword created from this dataword is **10111**, which is sent to the receiver.

We examine five cases for the even parity system we are using.

case 1

No error occurs; the received codeword is **10111**.

The syndrome is **0**.

The dataword **1011** is created.

case 2

- One single bit error changes a1. The received codeword is 10011.
- The syndrome is 1.
- No dataword is created.

case 3

- The parity bit, in our example changes. So now 10111 is 10110
- The syndrome is still 1
- No dataword is created

case 4

- Suppose now that we have two changes and the received codeword is 00110.
- The syndrome is 0.
- The dataword 0011 is created at the receiver.
- Note that here the dataword is wrongly created due to the syndrome value.

case 5

Suppose now we have three changes and the received codeword is 01011.

The syndrome is 1.

The dataword is not created.

This shows that the simple parity check, guaranteed to detect one single error, can also find any odd number of errors.

so we have an overall concept

- add controlled redundancy
- typically you use a look up table for the codewords
- send codeword
- compute syndrome
- discard or keep dataword on basis of the syndrome computation

what we just looked at

- Known as a **single error detecting code (SED)**. Only useful if probability of getting 2 errors is small since parity will become correct again
- Used in serial communications
- Low overhead but **not very powerful**
- Decoder can be implemented efficiently using a tree of XOR gates

challenge to do now

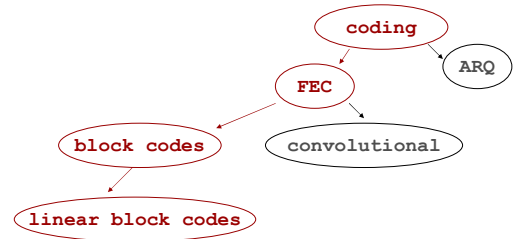
Can you design a coding scheme that would be able to correct a bit that was in error?

do this exercise in pairs and tell me what you came up with.

- so you have some basic ideas
- let's refine the concepts further
- remember we want to generate codes that are more powerful than the simple parity check
- we look now at **linear block codes**

linear block codes

- Almost all block codes used today belong to a subset called **linear block codes**.
- A linear block code is a code in which the exclusive OR (addition module 2) of two valid codewords creates another valid codeword



Let's see if the following codes are linear block codes

<i>Dataword</i>	<i>Codeword</i>
00	00000
01	01011
10	10101
11	11110

answer

- The scheme is a linear block code because the result of XORing any codeword with any other codeword is a valid codeword.
- For example, the XORing of the second and third codewords creates the fourth one.
- We can create all four codewords by XORing two other codewords.

why are these linear block codes of interest?

- in the simple parity example we actually created a lookup table for codes - example below

Datawords	Codewords	Datawords	Codewords
0000	00000	1000	10001
0001	00011	1001	10010
0010	00101	1010	10100
0011	00110	1011	10111
0100	01001	1100	11000
0101	01010	1101	11011
0110	01100	1110	11101
0111	01111	1111	11110

problem

- if we have large codes the lookup table can become enormous
- For a (127,92) code in which $k = 92$ there are 2^{92} code vectors

we now take a more dynamic approach

- Using linear block codes it is possible to create codewords by addition of other codewords - so we could try and store some kind of basic minimum set of codewords and not the whole set
- This has the advantage that there is now **no longer the need to hold every possible codeword in the table.**

the lookup table reduces in size

If there are k data bits, all that is required is to hold k **linearly independent codewords**, i.e., a set of k codewords none of which can be produced by linear combinations of 2 or more codewords in the set.

How do we do this?

The easiest way to find k linearly independent codewords is to choose those which have '1' in just one of the first k positions and '0' in the other $k-1$ of the first k positions.

- For example for a (7,4) code, only four codewords are required, e.g.,

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

- So, to obtain the codeword for dataword 1011, the first, third and fourth codewords in the list are added together, giving 1011010
- This process will now be described in more detail

- An (n, k) block code has code vectors $d = (d_1, d_2, \dots, d_k)$ and $c = (c_1, c_2, \dots, c_n)$
- The block coding process can be written as $c = dG$ where G is the Generator Matrix

$$G = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \cdot & \cdot & \dots & \cdot \\ a_{k1} & a_{k2} & \dots & a_{kn} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \cdot \\ a_k \end{bmatrix}$$

The generator matrix is a matrix of linearly independent vectors

- Thus,

$$c = \sum_{i=1}^k d_i a_i$$

Since codewords are given by summations of the a_i vectors, then to avoid 2 datawords having the same codeword the a_i vectors must be linearly independent

- Sum (mod 2) of any 2 codewords is also a codeword, i.e.,

- Since for datawords d_1 and d_2 we have;

$$d_3 = d_1 + d_2$$

- So,

$$c_3 = \sum_{i=1}^k d_3 a_i = \sum_{i=1}^k (d_{1i} + d_{2i}) a_i = \sum_{i=1}^k d_{1i} a_i + \sum_{i=1}^k d_{2i} a_i$$

$$c_3 = c_1 + c_2$$

note

0 is always a codeword, i.e., Since all zeros is a dataword then,

$$c = \sum_{i=1}^k 0 a_i = 0$$

Linear Block Codes - example 1

- For example a $(4, 2)$ code, suppose;

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad p_1 = [1011] \\ p_2 = [0101]$$

- For $d = [1 \ 1]$, then;

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

systematic block codes

For a **systematic** block code the dataword appears unaltered in the codeword - usually at the start - but can be elsewhere as well

The hardware needed to decode these is simple and hence systematic codes are desirable

note: everything we have been looking at is systematic but you can get codes that are not

Systematic Codes

- The generator matrix has the structure,

$$G = \begin{bmatrix} 1 & 0 & \dots & 0 & p_{11} & p_{12} & \dots & p_{1R} \\ 0 & 1 & \dots & 0 & p_{21} & p_{22} & \dots & p_{2R} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & p_{k1} & p_{k2} & \dots & p_{kR} \end{bmatrix} = [I|P] \quad R = n - k$$

length
k
n-k

- P is often referred to as parity bits [we do not need to have just one]

I and P

- I is the $k \times k$ identity matrix.
- This ensures the dataword appears at beginning of the codeword
- P is a $k \times R$ matrix.
- note k rows of generator matrix still must be linearly independent - i.e. it should not be possible to express any row in the matrix as a linear combination of the remaining rows

Linear Block Codes - example 2

- A (6,5) code with

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

identity matrix with rows of length k

- Is an even single parity code
- So we can think of our parity codes in terms of linear block codes

- suppose we want the code for 11100 we can add the first three together

- and we get 111001 as expected

$$c = dG$$

essentially once we have a generator matrix for a specific code type we can generate all the codes we need

decoding linear block codes

something to note about the hamming distance

Error Correcting Power of LBC

- The Hamming distance of a linear block code is simply the minimum Hamming weight (number of 1's or equivalently the distance from the all 0 codeword) of the non-zero codewords
- Therefore to find min Hamming distance just need to search among the 2^k codewords to find the min Hamming weight - **far simpler than doing a pair wise check for all possible codewords.**

minimum hamming distance for linear block codes

- recall we calculated the min distance of these using all the pairs (and in meantime have learned that these are linear)

Datawords	Codewords
00	000
01	011
10	101
11	110

Dataword	Codeword
00	0000
01	01011
10	10101
11	11110

Find the minimum Hamming distance of the coding scheme in Table 10.1.

Datawords	Codewords
00	000
01	011
10	101
11	110

we could have just looked at this

Solution

We first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(000, 110) = 2$ $d(011, 101) = 2$
 $d(011, 110) = 2$ $d(101, 110) = 2$

The d_{\min} in this case is 2.

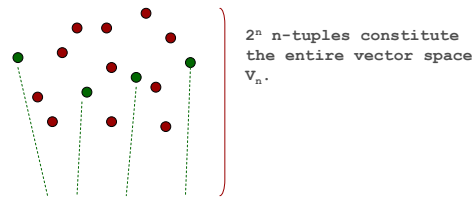
Decoding Linear Codes

- We take the approach we had before in the simple example, i.e. the error flag is computed from the received codeword
- But how does this apply to the more complex cases we are discussing now?
- We look towards what is called a parity check matrix

to understand the parity check matrix we need to talk about vector subspaces

vector subspaces

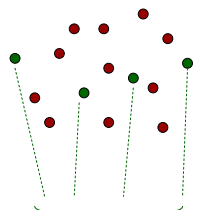
A linear block code is a linear subspace S_{sub} of all length n vectors (Space S)



2^k n-tuples constitute the **subspace** of codewords.

vector subspaces

- Consider the subset S_{null} of all **length n** vectors in space S that are **orthogonal** to all **length n** vectors in S_{sub}
- It can be shown that the dimensionality of S_{null} is **$n-k$** , where n is the dimensionality of S and k is the dimensionality of S_{sub}
- It can also be shown that S_{null} is a valid subspace of S and consequently S_{sub} is also the null space of S_{null}



2^k n-tuples constitute the **subspace** of codewords.

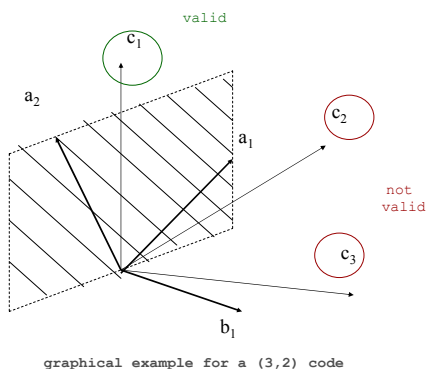
in other words there is another subspace in S of vectors that are orthogonal to the subspace shown here and this is denoted S_{null} and has dimensions $n-k$

- S_{null} can be represented by its **basis vectors**. In this case the generator basis vectors (or 'generator matrix' H) denote the generator matrix for S_{null} - of dimension $n-k = R$
- This matrix is called the **parity check matrix** of the code defined by G , where G is obviously the generator matrix for S_{sub} of dimension k
- Note that the number of vectors in the basis defines the dimension of the subspace

from last day

what are the main points you recall from the last lecture?

- S = the space of all vectors length n
- S_{sub} is the subspace of those vectors length n corresponding to the codewords that are used
- S_{null} is the subspace of vectors length n with are orthogonal to S_{sub}
- [in general the null space of a matrix, it is the set of points which go to zero when multiplied by the matrix.]
- G is the generator matrix for S_{sub}
- H is the generator matrix for S_{null} and is used to check for valid codewords - hence called the parity check matrix



the parity check matrix
-what do we do with it?

- It can be used as we have seen already to determine if a codeword is valid
- BUT it can also be used to correct

Error Syndrome

- For error correcting codes we need a method to compute the required correction
 - To do this we use the Error Syndrome, s of a received codeword, c_r

$$s = c_r H^T$$
 - If c_r is corrupted by the addition of an error vector, e , then

$$c_r = c + e$$
 and

$$s = (c + e) H^T = c H^T + e H^T$$

$$s = 0 + e H^T$$
- Syndrome depends only on the error

- That is, we can add the same error pattern to different codewords and get the same syndrome.
- There are $2^{(n-k)}$ syndromes but 2^n error patterns
- For example for a (3,2) code there is 1 syndrome and 8 error patterns
- There is no error correction possible in this case because we only have one syndrome to play with
- But a (7,4) code has 8 syndromes and 128 error patterns.
- With 8 syndromes we can provide a different value to indicate single errors in any of the 7 bit positions as well as the zero value to indicate no errors

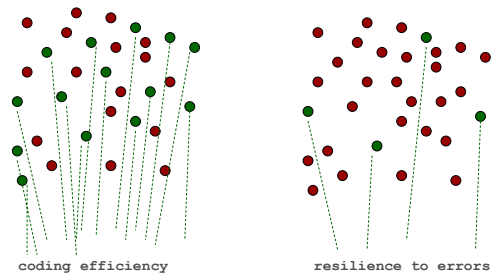
in otherwords using suitable codes
we can make a one to one
correspondence between syndromes and
correctable error patterns

we will now put all this in
context with a clear example

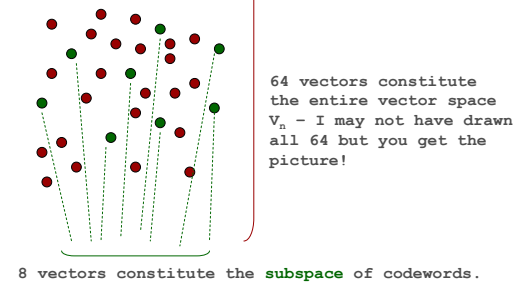
the graphical way of looking at
things helps

- we want to strive for coding efficiency by packing the V_n space with as many code vectors as possible - this is tantamount to saying we want the least amount of redundancy (excess bandwidth)
- we want the code vectors to be far enough apart from one another as possible so that if the vectors experience some corruption during transmission, they may still be correctly decoded with a high probability

the graphical way of looking at
things can help



- the best way to understand what we have learned about G and H is to work through an example
- let's look at a (6,3) code
- there are 2^k message vectors = $2^3=8$
- there are 2^n possible vectors in the space that constitutes all vectors of length n = 64
- we only need 8 of the 64 to represent our messages



How do we pick the 8 from the 64?

For linear block codes we pick 8 codes that are linear

message	codeword
000	000000
100	110100
010	011010
110	101110
001	101001
101	011101
011	110011
111	000111

these are linear codes

The eight code vectors form a subspace of V_6 and the sum of any two yields another codeword in the same subspace

how do we find the generator matrix G for this set of codes?

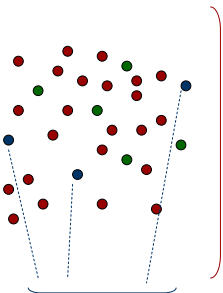
- the linearly independent set that spans the subspace is called a basis of the subspace
- any basis set of k independent n tuples can be used to generate the required linear block codewords

message	codeword
000	000000
100	110100
010	011010
110	101110
001	101001
101	011101
011	110011
111	000111

the basis set



these are used for the generator matrix



64 vectors constitute the entire vector space V_n - I may not have drawn all 64 but you get the picture!

3 vectors constitute the null subspace.

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

we can test this as we know $c=dG$ so let's test for $d = [110]$

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

$$c = 110100 + 011010 + 000000$$

we find that $c = 101110$ which is the code vector we expected

of course the value of the generator matrix is more obvious for big systems but a big example would take too long to calculate here!

looking at the generator matrix

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

parity bits identity matrix

we can reverse the order of these - as per earlier slides it does not matter

$$c = [d_1 \quad d_2 \quad d_3] \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

parity bits identity matrix

when we look at what happens we see
 $c = d_1+d_3, d_1+d_2, d_2+d_3, d_1, d_2, d_3$

the redundant bits are produced in a variety of ways - and intuitively we can get a feeling that this is more robust than a simple parity - providing greater ability to detect and correct

so what happens at the receiver?

suppose now we transmit our codeword and errors occur - the parity check matrix comes to play ...

creating the parity check matrix

$$H = \left[I_{n-k} \mid P^T \right]$$

This is how you generate the parity check matrix - remember you are finding all the vectors that are orthogonal to the vectors in the generator matrix

$$H^T = \begin{bmatrix} I_{n-k} \\ P \end{bmatrix}$$

write this down as you will need this later

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

parity bits identity matrix

$$H^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

identity matrix parity bits

some calculations

- so we learned already that if $r =$ the received codeword then S , the syndrome $= rH^T = eH^T$

write this down as you will need this later

- let's look at sending the codeword 101110 and receiving a correct codeword and a codeword with an error resulting in $r = 001110$.

receiving the correct codeword

- $rH^T = 0$
- as the codeword is correct

receiving the incorrect codeword

$$S = [0 \ 0 \ 1 \ 1 \ 1 \ 0] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= [1, 1+1, 1+1] = [1, 0, 0]$$

we can verify that we get the same answer from $S = eH^T$ where $e = [1 \ 0 \ 0 \ 0 \ 0 \ 0]$

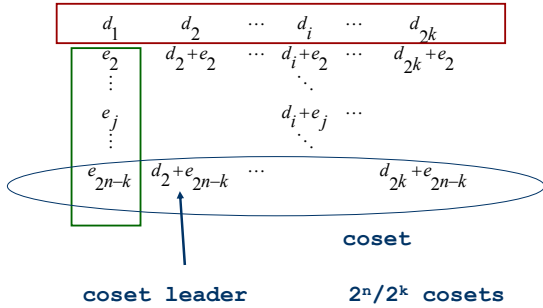
so we can detect that there is an error because the syndrome is not zero but how does this help us with correction?

the standard array

- we need to make use of what is called a standard array
- this is an array of all the n length vectors in the space V_n that are arranged in a special order.
- the code vectors or codewords are along the top - first row and the error vectors make up the first column.

	d_1	d_2	...	d_i	...	d_{2k}
e_2	$d_2 + e_2$...	$d_i + e_2$...	$d_{2k} + e_2$	
\vdots			\ddots			
e_j			$d_i + e_j$...		
\vdots			\ddots			
e_{2n-k}	$d_2 + e_{2n-k}$...			$d_{2k} + e_{2n-k}$	

I cut and paste this table from somewhere
and the codewords are labelled d_1, d_2 etc and not
as I had labelled them earlier



SO ...

- If a code vector c_i is transmitted over a noisy channel the receive vector will be decoded correctly if the error pattern caused by the channel is a coset leader.
- If the error pattern is not a coset leader then erroneous decoding will result.
- All members of a coset have the same syndrome and in fact the *syndrome is used to estimate the error pattern.*

error correction decoding

1. Calculate the syndrome of r using $S = rH^T$
2. Locate the coset leader (error pattern), e_j , whose syndrome equals rH^T
3. This error pattern is the corruption caused by the channel.
4. The corrected received vector is identified as $U = r + e_j$. We retrieve the valid code vector by subtracting out the identified error

Note: In modulo-2 arithmetic *subtraction is identical to that of addition*

Locating the error pattern:

For the (6,3) linear block code we have seen before the standard array can be arranged as:

000000	110100	010000	101100	101001	011101	110011	000111
000001	110001	011011	101111	101000	011100	110010	000110
000010	110010	011000	101100	101011	011111	110001	000101
000011	110011	011011	101011	101101	011001	110011	000001
001000	111100	010010	100110	100001	010101	111011	001011
001001	111101	010001	111110	111001	010101	100011	001011
001010	111000	011010	001110	001001	111101	010011	100111
001011	111001	011001	111111	111000	001100	100010	010110

correctable error patterns

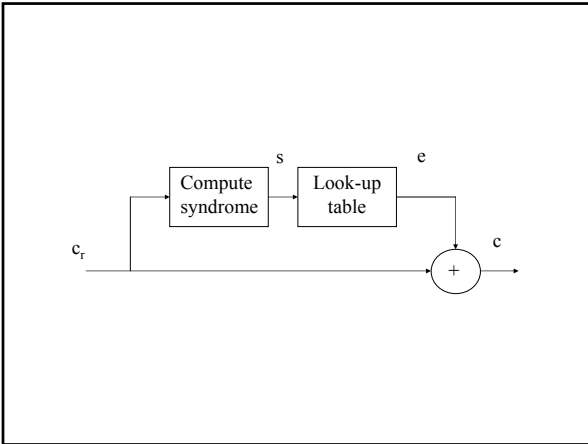
syndrome for each error pattern

$$S = e_j$$

error pattern	Syndrome
000000	000
000001	101
000010	011
000100	110
001000	001
010000	010
100000	100
010001	111

process

- We receive the vector r and calculate its syndrome S
- We then use the syndrome-look-up table to find the corresponding error pattern.
- This error pattern is an estimate of the error, we denote it as \hat{e}
- The decoder then adds \hat{e} to r to obtain an estimate of the transmitted code vector \hat{U}
- $\hat{U} = r + \hat{e} = (U + e) + \hat{e} = U + (e \oplus \hat{e})$
- If the estimated error pattern is the same as the actual error pattern that is if $\hat{e} = e$ then $\hat{U} = U$
- If $\hat{e} \neq e$ the decoder will estimate a code vector that was not transmitted and hence we have an undetectable decoding error.



EXAMPLE

- Assume code vector $U = [1\ 0\ 1\ 1\ 1\ 0]$ is transmitted and the vector $r = [0\ 0\ 1\ 1\ 1\ 0]$ is received.
- The syndrome of r is computed as:
 $S = [0\ 0\ 1\ 1\ 1\ 0]H^T = [1\ 0\ 0]$
- From the look-up table 100 has corresponding error pattern:
 $\hat{e} = [1\ 0\ 0\ 0\ 0\ 0]$
- The corrected vectors is the $\hat{U} = r + \hat{e} = 0\ 0\ 1\ 1\ 1\ 1$
 $= 1\ 0\ 1\ 1\ 1\ 0$ (corrected)
- In this example actual error pattern is the estimated error pattern,
- Hence $\hat{U} = U$

- so like any approach it has its limitations
- codes are designed only to be robust to particular points

error patterns	Syndromes
000000	000
000001	101
000010	011
000100	110
001000	001
010000	010
100000	100
010001	111

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

- G is the generator matrix we looked at for the (6,3) code. What codeword is sent to represent the data word [111]?
- What is the error syndrome when the dataword part of the codeword gets flipped to [110] and when it gets flipped to [100]?
- Using the error pattern and syndrome table above determine if you can do error correction.

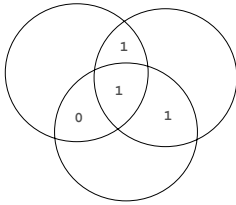
Hamming Codes

- Hamming codes are a particular type of block codes
- Number of parity bits $R = n - k$ and $n = 2^R - 1$
- All of what you learned so far applies and Hamming's really are just block codes of specific forms such as (3,1), (7,4), (15,11).

general concept - based on 7,4

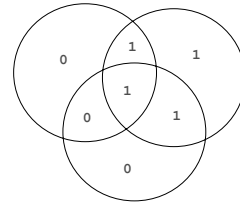
suppose you want to get a code for the dataword 0111

general concept - based on 7,4



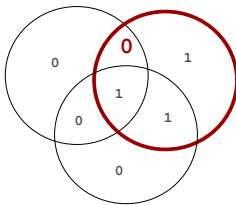
suppose you want to get a code for the dataword 0111

general concept - based on 7,4



suppose you want to get a code for the dataword 0111

general concept - based on 7,4



suppose we get a flipped bit in the received codeword
-it can cope with it. The hamming codes all have $d_{\min} = 3$

Hamming (7,4)

$$c_i = d_i, \quad i=1,2,3,4$$

$$c_5 = c_1+c_2+c_3$$

$$c_6 = c_2+c_3+c_4$$

$$c_7 = c_1+c_2+c_4$$

exercise for Friday

1. choose one idea/concept that was covered on the course
2. you have three minutes to describe the idea/concept and insight/understanding you have gained
3. you are to explain this idea/concept as if explaining to **first year engineers**
4. you should make one powerpoint slide to aid in the process
5. you cannot use equations at all
6. you can use simple text or images provided they are **NOT** from the course notes
7. marks will be given for this and another exercise next term