

**EEE8099 - Information Theory & Coding**  
**EEE8104 - Digital Communications**

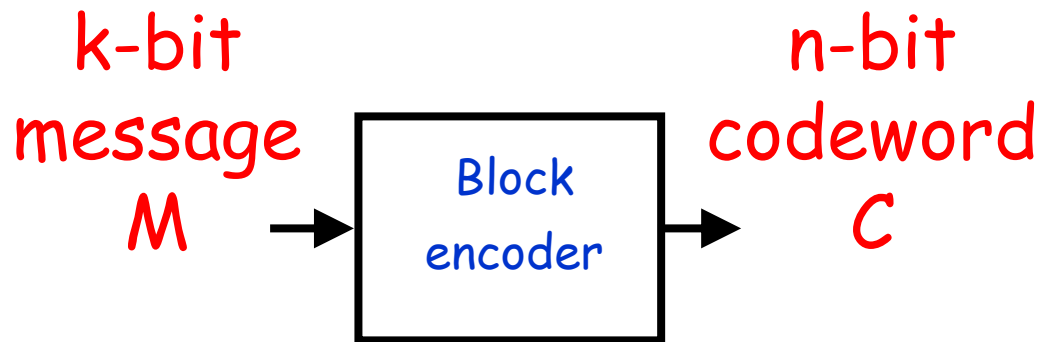
**S. Le Goff**

**School of Engineering @ Newcastle University**

Part 6  
Practical Error-Correcting Codes  
1948 - 1993

# Hamming codes (1950)

1950: Richard W Hamming introduces Hamming codes, the first ever class of linear block codes.



Richard W Hamming (1915 - 1998)

# Hamming codes

For any integer  $m \geq 3$ , there is a Hamming code, with  $n = 2^m - 1$  and  $k = 2^m - 1 - m$ , so that one error can be corrected in a received word of  $n$  bits ( $t = 1, d_{min} = 3$ ).

- $n = 7, k = 4$  ( $R_c \sim 0.57$ )
- $n = 15, k = 11$  ( $R_c \sim 0.73$ )
- $n = 31, k = 26$  ( $R_c \sim 0.84$ )
- $n = 63, k = 57$  ( $R_c \sim 0.90$ )
- $n = 127, k = 120$  ( $R_c \sim 0.94$ ), and so on.

Over BSC, Hamming codes are decoded using the syndrome decoding algorithm.

If  $n$  is small enough, they can also be decoded using the ML algorithm in hard and soft-decisions.

# Hamming codes

Encoding table for the (7, 4) Hamming code

M				C						
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1	0	1
0	0	1	0	0	0	1	0	1	1	1
0	0	1	1	0	0	1	1	0	1	0
0	1	0	0	0	1	0	0	0	1	1
0	1	0	1	0	1	0	1	1	1	0
0	1	1	0	0	1	1	0	1	0	0
0	1	1	1	0	1	1	1	0	0	1

M				C						
1	0	0	0	1	0	0	0	1	1	0
1	0	0	1	1	0	0	1	0	1	1
1	0	1	0	1	0	1	0	0	0	1
1	0	1	1	1	0	1	1	1	0	0
1	1	0	0	1	1	0	0	1	0	1
1	1	0	1	1	1	0	1	0	0	0
1	1	1	0	1	1	1	0	0	1	0
1	1	1	1	1	1	1	1	1	1	1

# Hamming codes

Hamming codes are linear codes: they are defined by a generator matrix  $G$  so that  $C = M.G$ .

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \Rightarrow \begin{aligned} c_0 &= m_0 \\ c_1 &= m_1 \\ c_2 &= m_2 \\ c_3 &= m_3 \\ c_4 &= m_0 + m_2 + m_3 \\ c_5 &= m_0 + m_1 + m_2 \\ c_6 &= m_1 + m_2 + m_3 \end{aligned}$$

The code is completely defined by a set of  $n$  linear equations

$c_0, c_1, c_2, c_3$ : info bits -  $c_4, c_5, c_6$ : parity bits.

# Parity-check matrix of a linear block code

Another possible representation for a linear block code: the parity-check matrix  $H$ .

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

A binary word  $W$  of  $n = 7$  bits is a codeword if and only if the syndrome  $s$  defined as  $s = W \cdot H^T$  is the all-zero vector. We write the condition as  $s = W \cdot H^T = 0$ .

# Parity-check matrix of a linear block code

It is easy to generate the parity-check matrix  $H$  if we know the generator matrix  $G$ .

The seven encoding linear equations are

$$(0) \ c_0 = m_0$$

$$(1) \ c_1 = m_1$$

$$(2) \ c_2 = m_2$$

$$(3) \ c_3 = m_3$$

$$(4) \ c_4 = m_0 + m_2 + m_3$$

$$(5) \ c_5 = m_0 + m_1 + m_2$$

$$(6) \ c_6 = m_1 + m_2 + m_3$$



# Parity-check matrix of a linear block code

Equations (4) to (6) can be rewritten as

$$m_0 + m_2 + m_3 + c_4 = 0 \rightarrow c_0 + c_2 + c_3 + c_4 = 0$$

$$m_0 + m_1 + m_2 + c_5 = 0 \rightarrow c_0 + c_1 + c_2 + c_5 = 0$$

$$m_1 + m_2 + m_3 + c_6 = 0 \rightarrow c_1 + c_2 + c_3 + c_6 = 0$$

These  $(n - k) = 3$  equations specify the conditions that must be satisfied for a 7-bit word to be a valid codeword. They can be written in a  $(n - k) \times n$  matrix form as

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

# Parity-check matrix of a linear block code

The transpose of  $H$  is a  $n \times (n - k)$  matrix given by

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

If we consider a 7-bit binary word  $W$  defined as  $W = (w_0 \ w_1 \ w_2 \ w_3 \ w_4 \ w_5 \ w_6)$ , it is easy to see that  $W$  is a valid codeword if and only if

# Parity-check matrix of a linear block code

$$W \cdot H^T = (w_0 \ w_1 \ w_2 \ w_3 \ w_4 \ w_5 \ w_6) \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = (000).$$

In general, if the generator matrix is a  $k \times n$  matrix in the form  $G = [I_k | P]$ , where  $I_k$  is a  $k \times k$  identity matrix and  $P$  is a  $k \times (n - k)$  matrix,...

# Parity-check matrix of a linear block code

...then the corresponding parity-check matrix is a  $(n - k) \times n$  matrix in the form  $H = [P^T | I_{n-k}]$ , where  $I_{n-k}$  is a  $(n - k) \times (n - k)$  identity matrix and  $P^T$  is a  $(n - k) \times k$  matrix.

All codewords  $C$  are such that  $C \cdot H^T = 0$ . In other words, all codewords  $C$  satisfy the parity-check equations

$$c_0 + c_2 + c_3 + c_4 = 0$$

$$c_0 + c_1 + c_2 + c_5 = 0$$

$$c_1 + c_2 + c_3 + c_6 = 0$$

# BCH codes (1959)

1959: Raj C Bose (1901 - 1987) and D K Chaudhuri propose a new class of multiple-error correcting linear block codes, discovered independently by A. Hocquenghem (1908 - 1990) → BCH codes.

BCH codes are cyclic codes. A code is said to be cyclic if any shift of a codeword is also a codeword.

Cyclic codes have considerable algebraic structure, which simplifies the encoding and decoding procedures.

# BCH codes

For any positive integers  $m \geq 3$  and  $t < 2^{m-1}$ , there is a binary BCH code with  $n = 2^m - 1$ ,  $k = n - mt$ ,  $d_{\min} = 2t + 1$ .

Ex:  $t = 1 \rightarrow$  Hamming codes

Ex:  $t = 2$ ,  $d_{\min} = 5$

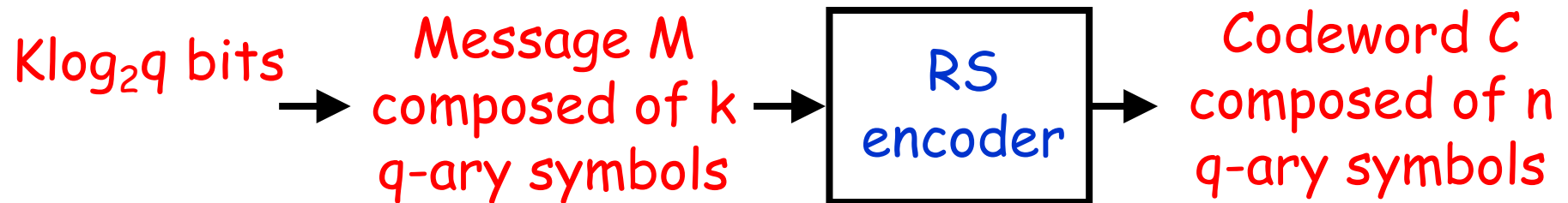
- $n = 15, k = 7$
- $n = 31, k = 21$
- $n = 63, k = 51$
- $n = 127, k = 113$
- $n = 255, k = 239$  etc.

Decoded using either the Berlekamp algorithm or the Euclidean algorithm.

# Reed-Solomon codes (1960)

1960: Irving S Reed (1923 - 2012) and Gustave Solomon (1930 - 1996) develop a block coding scheme particularly powerful for correcting bursts of errors  
→ Reed-Solomon codes.

RS codes are non-binary cyclic codes (subset of non-binary BCH codes). Each codeword  $C$  is composed of  $n$   $q$ -ary symbols, where  $q$  is a power of 2.



# Reed-Solomon codes (1960)

A RS code is defined by the following parameters:

$n = q-1$ ,  $k = 1, 3, \dots, n-2$ , and  $d_{\min} = n - k + 1 \Rightarrow t = (n-k)/2$ .

Ex:  $q = 256$  (1 symbol  $\equiv$  8 bits)

- $n = 255, k = 223 \rightarrow d_{\min} = 33 \rightarrow t = 16$
- $n = 255, k = 239 \rightarrow d_{\min} = 17 \rightarrow t = 8$

RS codes are particularly powerful for situations where errors tend to happen in "bursts" rather than randomly (e.g., CD-Rom).



# Reed-Solomon codes (1960)

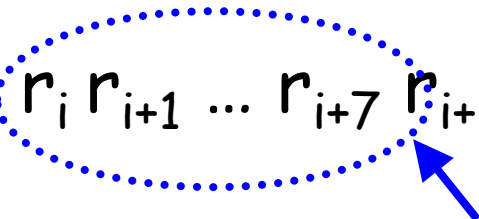
Ex:  $(n = 255, k = 239)$  RS code with  $d_{\min} = 17$  ( $t = 8$ )

Such code can correct up to 8 256-ary symbols in a received word of 255 256-ary symbols.

Length of a codeword  $C = (c_0 c_1 c_2 \dots c_{254}) = 2040$  bits.

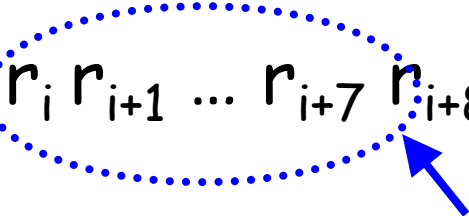
Assume we receive the word

$R = (r_0 r_1 r_2 \dots r_{i-1} r_i r_{i+1} \dots r_{i+7} r_{i+8} \dots r_{254})$



Error burst  $\equiv$  8 consecutive symbols are received erroneously  
(1 bit out of 8 is wrong or all 8 bits are wrong)

# Reed-Solomon codes (1960)

$$R = (r_0 \ r_1 \ r_2 \ \dots \ r_{i-1} \ r_i \ r_{i+1} \ \dots \ r_{i+7} \ r_{i+8} \ \dots \ r_{254})$$


Such error burst can be corrected since  $t = 8$

If all bits in every symbol are wrong  $\rightarrow$  Up to 64 bits in a received word of 2040 bits can be corrected.

If only one bit per symbol is wrong  $\rightarrow$  Up to 8 bits in a received word of 2040 bits can be corrected.

RS codes are very attractive for correcting error bursts.

# LDPC codes (1962)

1962: Robert G Gallager introduces low-density parity-check (LDPC) codes. He did it during his PhD studies supervised by Peter Elias at the MIT.



The potential of LDPC codes remained undiscovered at the time due to the computational demands of simulations in an era when vacuum tubes were only just being replaced by the first transistors.

They remained largely neglected for over 35 years, and were rediscovered in 1995 by David MacKay at the University of Cambridge.

# LDPC codes

LDPC codes are linear block codes with a very sparse parity-check matrix  $H$ .

In other words, the parity-check matrix of a LDPC code contains only a very small number of 1s.

The sparseness of  $H$  guarantees a decoding complexity that increases only linearly with the code length  $n$ .

This sparseness is also the reason why the minimum distance  $d_{\min}$  of LDPC codes increases linearly with  $n$ .

# LDPC codes

The difference between classical block codes (e.g., Hamming, BCH, and Reed-Solomon) and LDPC codes is how they are decoded.

Classical block codes are generally decoded with ML-like decoding algorithms and so are usually short (meaning that  $k$  and  $n$  are small) and designed algebraically to make this task less complex.

LDPC codes are decoded iteratively using a graphical representation of their parity-check matrix and so are designed with the properties of  $H$  as a focus.

# LDPC codes

LDPC codes are designed by constructing a sparse parity-check matrix first and then determining the generator matrix  $G$  for the code afterwards.

An LDPC code parity-check matrix is called  $(w_c, w_r)$ -regular if each coded bit is contained in a fixed number,  $w_c$ , of parity checks and each parity-check equation contains a fixed number,  $w_r$ , of coded bits.

In other words, an LDPC code parity-check matrix is called  $(w_c, w_r)$ -regular if each column contains  $w_c$  1s and each row contains  $w_r$  1s.

# LDPC codes

Gallager provided the following method to design a  $(w_c, w_r)$ -regular parity-check matrix.

The  $(n - k)$  rows are divided into  $w_c$  sets with  $\frac{n-k}{w_c}$  rows in each set. The first set of rows contains  $w_r$  consecutive 1s ordered from left to right across the columns. Every other set of rows is a randomly-chosen column permutation of the first set.

As a result, each column has a 1 once in every one of the  $w_c$  sets.

# LDPC codes

Example of a length-12 ( $n = 12$ ), ( $w_c = 3, w_r = 4$ )-regular matrix constructed using Gallager's method

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



# LDPC codes

The coding rate of the resulting LDPC code is determined by counting the number of rows and columns in this parity-check matrix: 12 columns  $\rightarrow n = 12$  and 9 rows  $\rightarrow n - k = 9 \rightarrow k = n - 9 = 3$ .

The coding rate is thus  $R_c = \frac{k}{n} = \frac{3}{12} = \frac{1}{4}$ .

We notice that  $H$  is constructed in a pseudo-random fashion which reminds us about the advice given by Shannon: use a random code if you want to operate near the channel capacity limit.

# LDPC codes

Other linear block codes as well as convolutional codes are not designed in a random fashion at all.

On the contrary, they are constructed with plenty of structure to maximise the parameter  $d_{min}$  and make the encoding and decoding as simple as possible.

Apart from Gallager's method, there are many other methods that have been proposed over the years to construct pseudo-random parity-check matrices.

LDPC codes are represented in graphical form by a Tanner graph.

# Encoding of a LDPC code

To implement an LDPC encoder, we need to know its generator matrix  $G$ .

If  $H$  is in the form  $H = [A|I_{n-k}]$ , then the generator matrix is given by  $G = [I_k|A^T]$ .

The issue is to obtain a parity-check matrix in the form  $H = [A|I_{n-k}]$ .

This can be done by performing Gauss-Jordan elimination on an original parity-check matrix  $H$ .

# Tanner graph of a LDPC code

Graphical representation of LDPC codes: Tanner graph  
(R M Tanner, 1981)

Consider a length-6 ( $n = 6$ ),  $(w_c = 2, w_r = 3)$ -regular LDPC code with the following parity-check matrix.

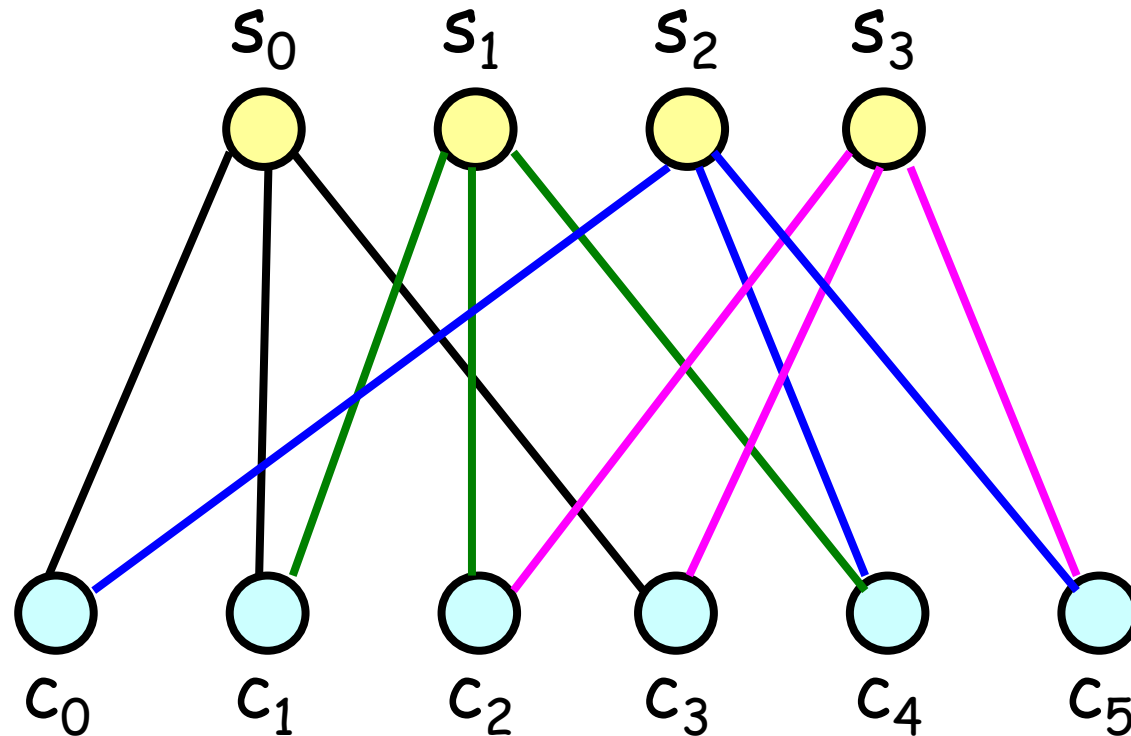
$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

The coding rate of this code is  $R_c = \frac{2}{6} = \frac{1}{3}$ .

# Tanner graph of a LDPC code

The Tanner graph of this code is shown below.

Check nodes representing the  $(n - k)$  parity checks (also called constraint nodes)



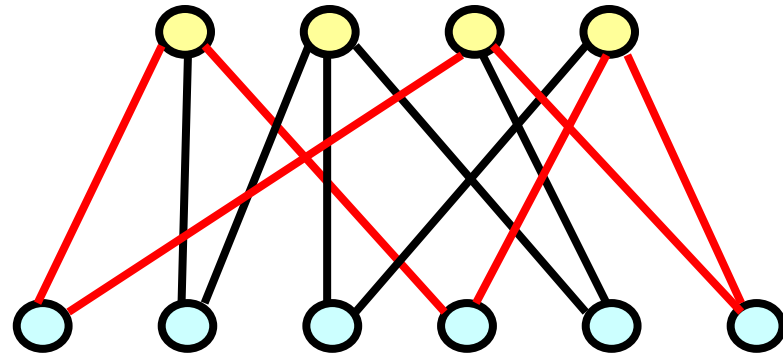
Bit nodes representing the  $n$  coded bits (also called variable nodes)

# Tanner graph of a LDPC code

A cycle in a tanner graph is a sequence of connected vertices which start and end at the same vertex in the graph, and which contain other vertices no more than once.

The length of a cycle is the number of edges it contains, and the girth of a graph is the size of its smallest cycle.

A cycle of length 6 is shown in red in this example.

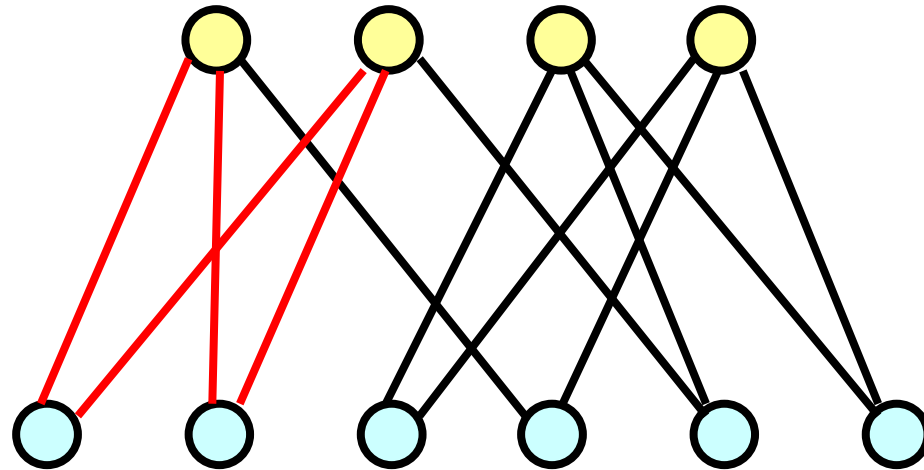


# Tanner graph of a LDPC code

We will later see that cycles of length 4 are particularly detrimental to the performance of the decoding algorithm.

A matrix  $H$  should therefore be designed so that there are no length-4 cycles in its Tanner graph.

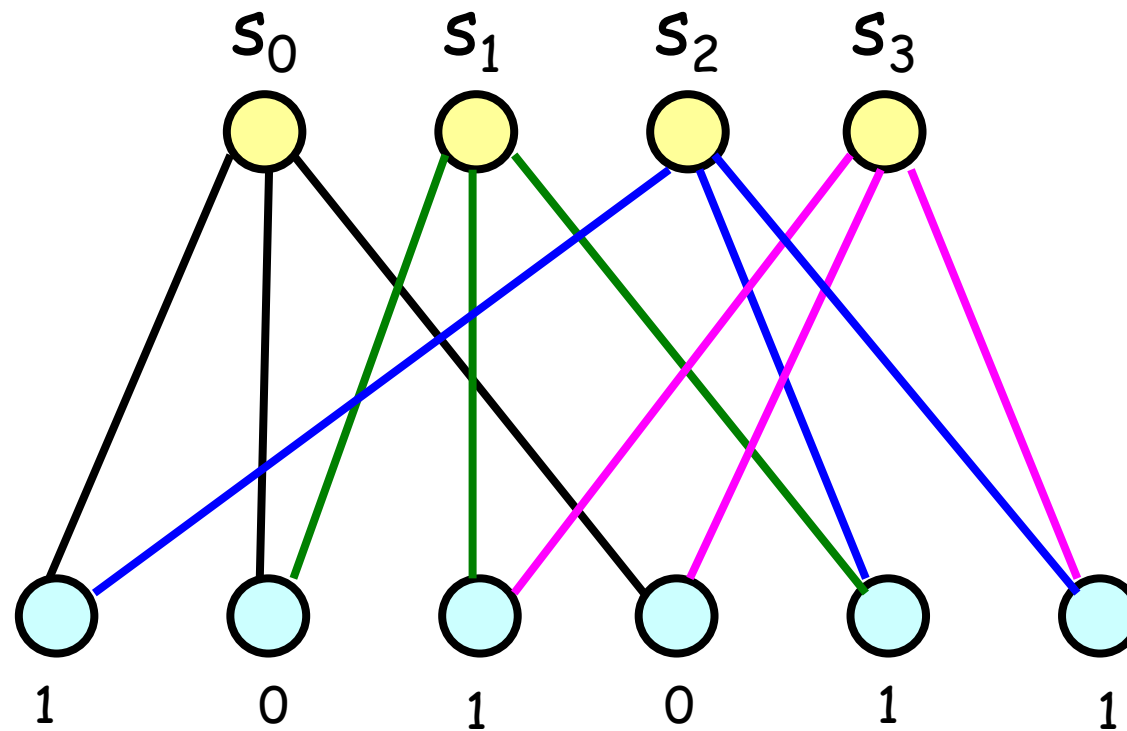
A cycle of length 4 is shown in red in this Tanner graph.



# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

Consider an LDPC code with the Tanner graph shown below. Assume  $R = (101011)$ . What is the decoded codeword?

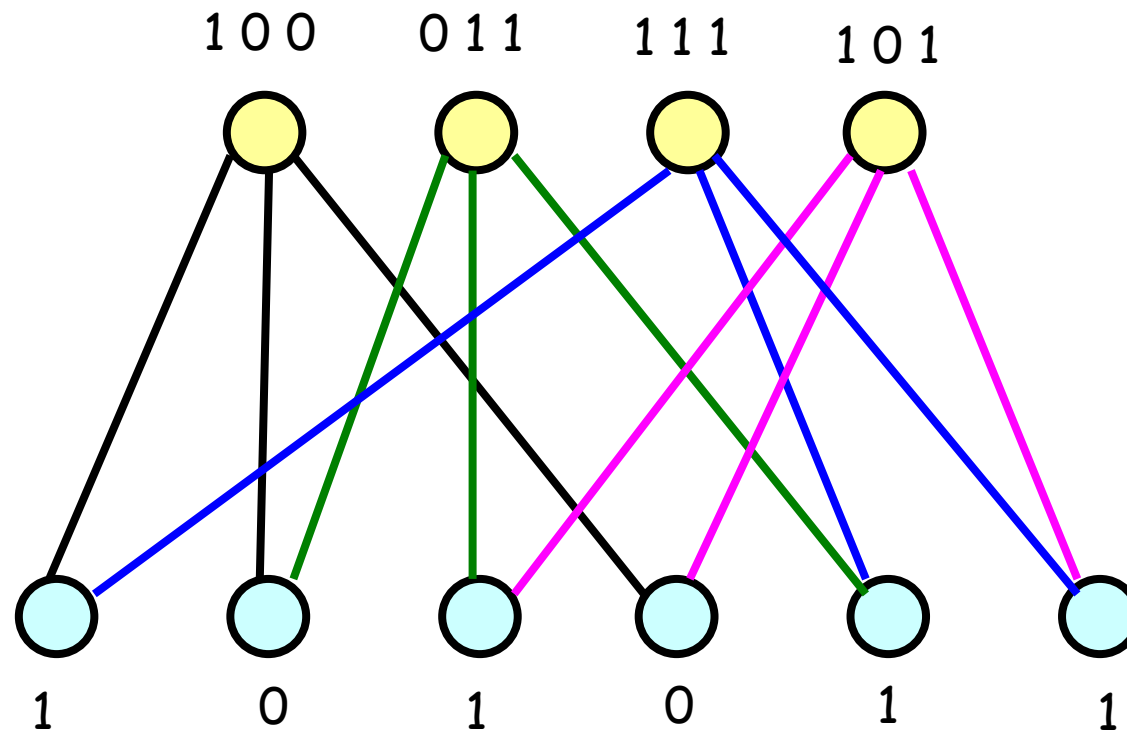




# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

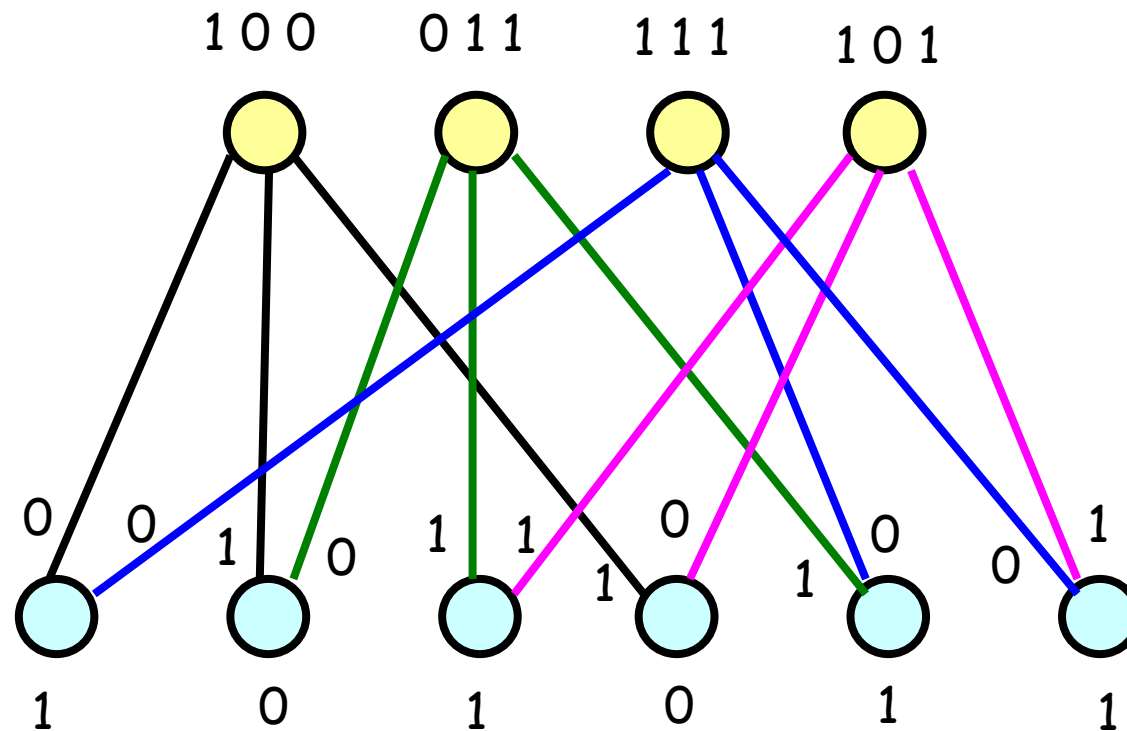
The bits received from the BSC are sent to the check nodes. We see that the constraints at the check nodes  $s_0$  and  $s_2$  are not respected.



# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

Each check node computes new estimates of the bits associated with it and send them to the relevant bit nodes.



# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

Each check node computes new estimates of the bits associated with it and send them to the relevant bit nodes.

The computation of each estimate is done by using the fact that a constraint  $c_i + c_j + c_l = 0$  implies that we can write

$$c_i = c_j + c_l;$$

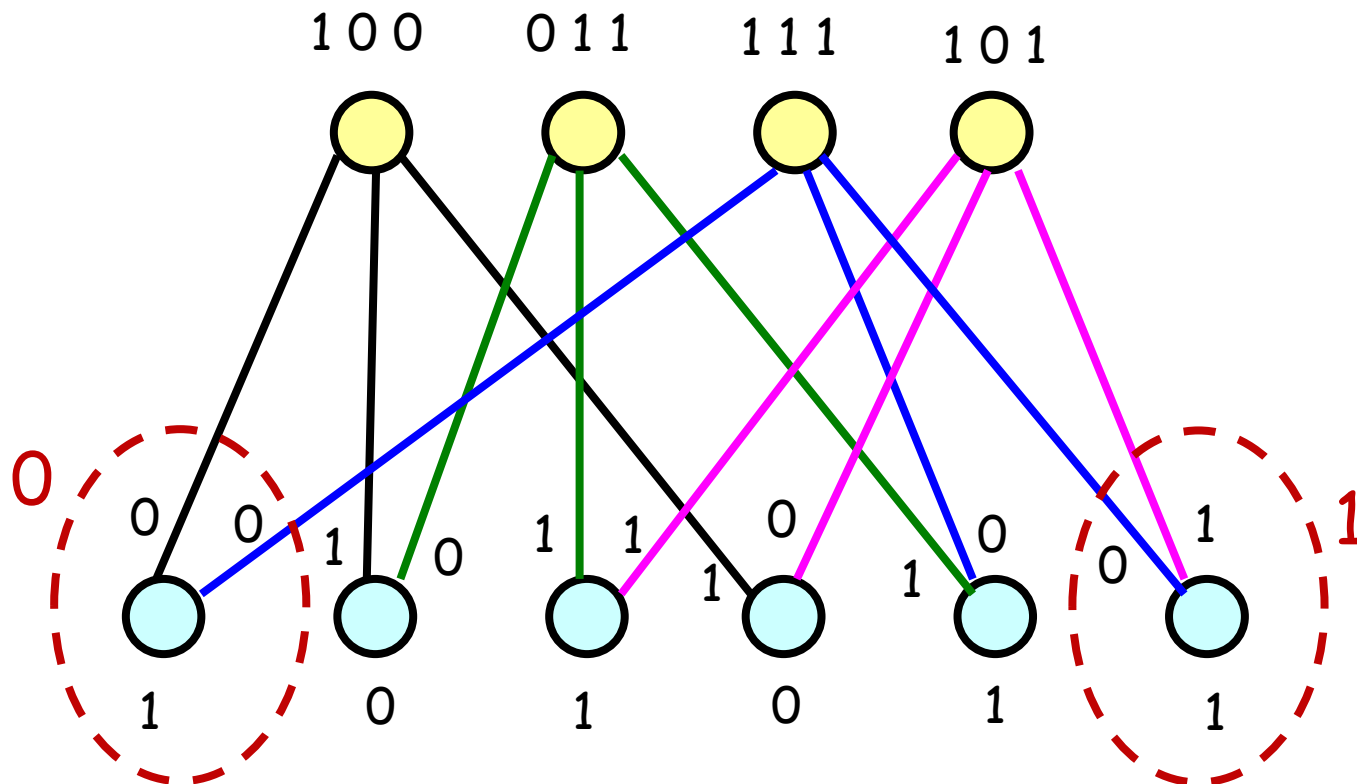
$$c_j = c_i + c_l;$$

$$c_l = c_i + c_j.$$

# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

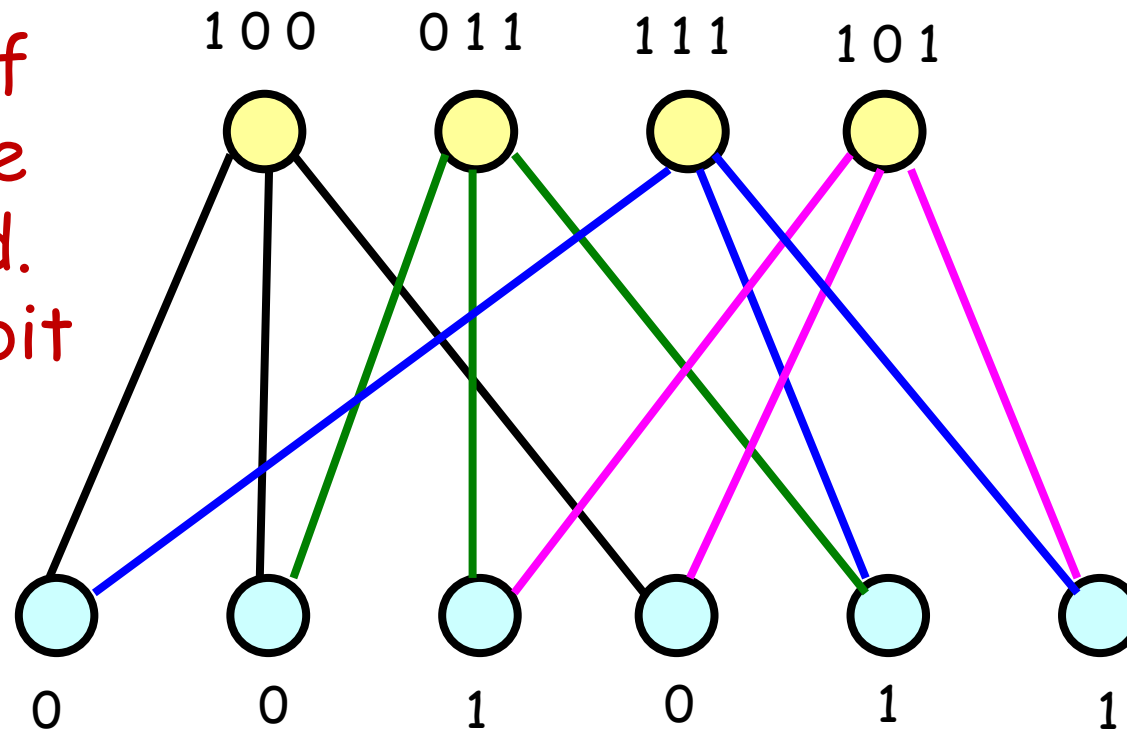
Each bit node value is updated by taking a majority decision based on the three available estimates.



# Decoding of a LDPC code over BSC Bit-Flipping Algorithm

Each bit node value is updated by taking a majority decision based on the three available estimates.

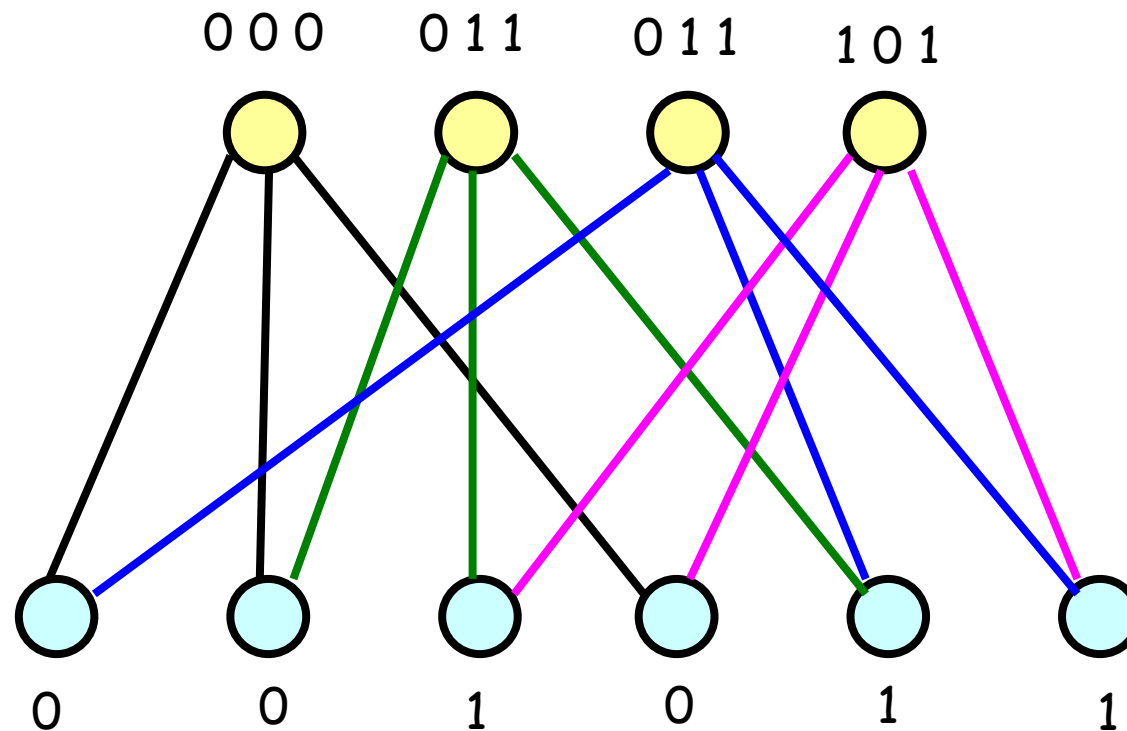
The value of the bit node  $c_0$  is flipped. The other bit nodes are unchanged.



# Decoding of a LDPC code over BSC

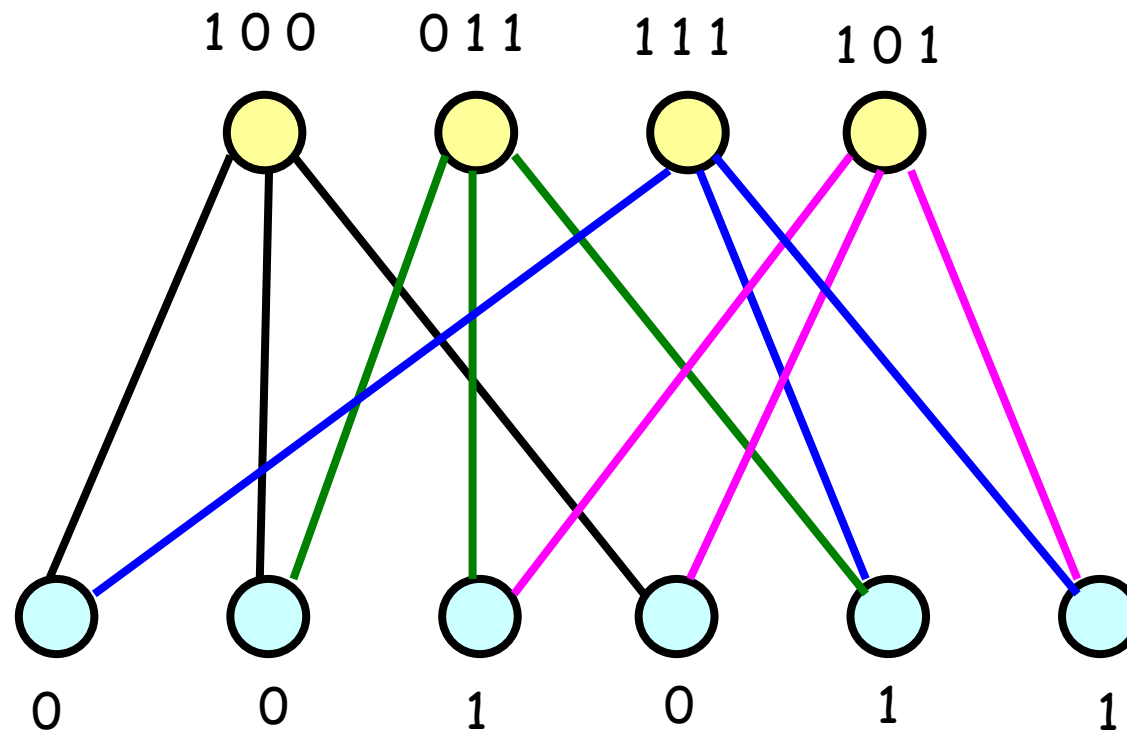
## Bit-Flipping Algorithm

The updated bits are sent once again to the check nodes for another decoding iteration. We must check whether further bit updates are still possible.



# Decoding of a LDPC code over BSC Bit-Flipping Algorithm

We see that all check node constraints are now respected  $\rightarrow$  The decoding process can stop and (001011) is the decoded codeword.



# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

The bit-flipping algorithm is a hard-decision (iterative) decoding process whose complexity is proportional to the codeword length  $n$  (very good).

This algorithm is thus suitable for decoding linear block codes with large values of  $k$  and  $n$ .

This is in contrast with the ML decoding process whose complexity is proportional to  $2^k$  and thus not suitable for large values of  $k$  and  $n$ .



# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

The bit-flipping algorithm works well if

- There are very few bits associated with each check node (to minimise the probability that two or more wrong bits are sent to the same check node during the decoding process).
- A bit node is connected to very few check nodes (to minimise the number of check nodes affected by a wrong bit).

# Decoding of a LDPC code over BSC

## Bit-Flipping Algorithm

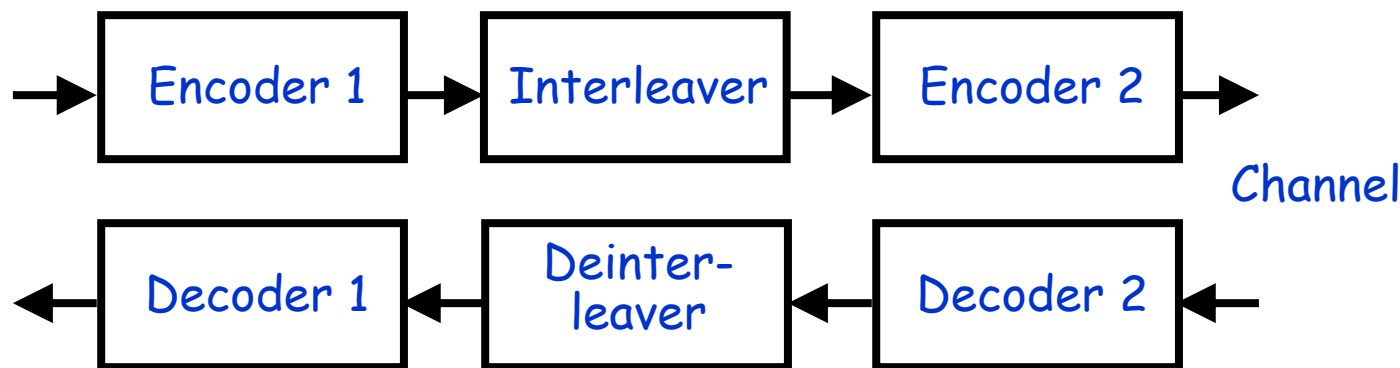
In other words, the bit-flipping algorithm works well only if the parity-check matrix  $H$  is very sparse.

We must understand that any linear block code could potentially be decoded using the bit-flipping algorithm. After all, any linear block code can be described using a Tanner graph.

But only the codes with a low-density parity-check matrix can be decoded efficiently using the bit-flipping algorithm.

# Concatenation of codes (1966)

1966: David Forney proposes to concatenate codes for improving error performance without having to use complex codes.



NASA standard in the mid-90s:

Code 1 (outer code): Reed-Solomon code;

Code 2 (inner code):  $R_c = \frac{1}{2}$ ,  $K = 7$ , (133, 171) CC  
decoded using soft-decision Viterbi decoding.