AN INTRODUCTION TO ERROR CORRECTING CODES
Part 1

Jack Keil Wolf

ECE 154C

Spring 2008

Noisy Communications

- **Noise in a communications channel can cause errors in the transmission of binary digits.**

- **Transmit: 1 1 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0 …**
- **Receive: 1 1 0 1 1 0 1 0 0 0 1 0 0 0 0 1 0 …**

- **For some types of information, errors can be detected and corrected but not in others.**

Example: Transmit: Come to my house at 17:25 …
Receive: Come tc my houzx at 14:25 …

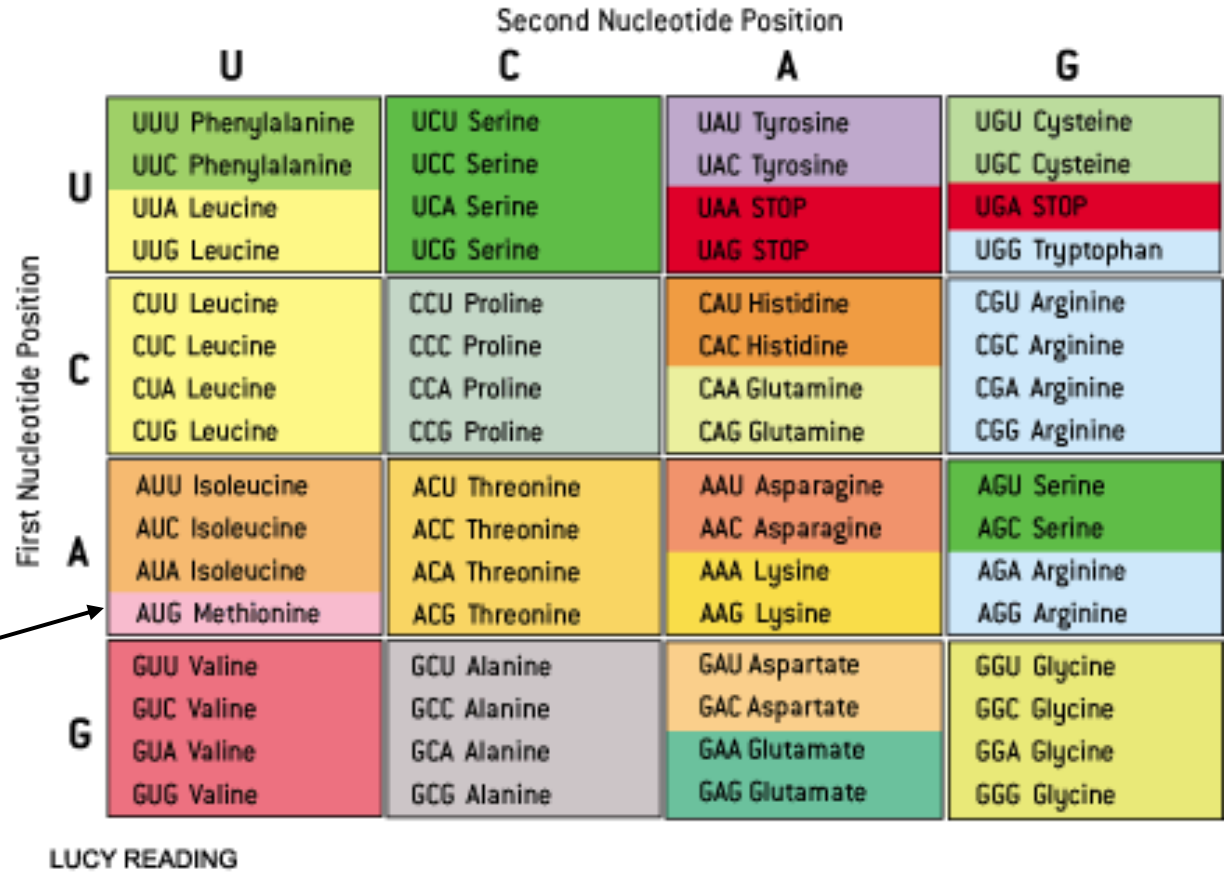
Making Digits Redundant

- **In binary error correcting codes, only certain binary sequences (called code words) are transmitted.**
- **This is similar to having a dictionary of allowable words.**
- **After transmission over a noisy channel, we can check to see if the received binary sequence is in the dictionary of code words and if not, choose the codeword most similar to what was received.**

NATURE'S ERROR CONTROL CODE

- **Nature's code is a mapping of RNA sequences to proteins.**
- **"RNA" consists of four "symbols": A, U, G, and C. "Proteins" consists of 20 "symbols": the amino acids.**
- **The genetic code is a code in which three nucleotides in RNA specify one amino acid in protein.**

NATURE'S ERROR CONTROL DECODING TABLE



AUG starts codon.

Sometimes one or more of the RNA symbols Is changed.

Hopefully, the resultant triplet still decodes to the same protein.

RNA-Amino Acid Coding

OUTLINE

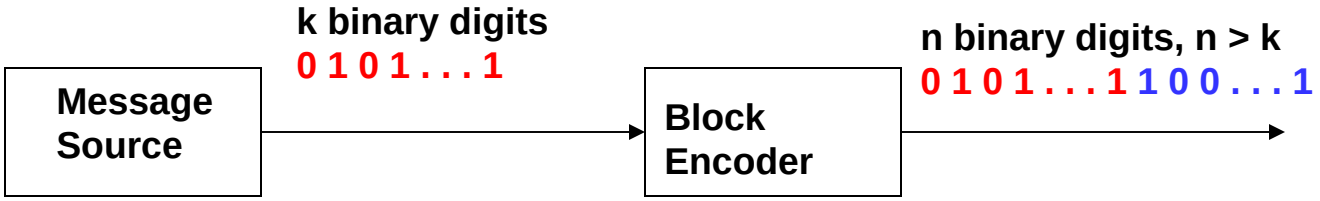
- Types of Error Correction Codes
- Block Codes:
  - Example: (7,4) Hamming Codes
  - General Theory of Binary Group Codes
  - Low Density Parity Check (LDPC) Codes
  - Reed Solomon (RS) Codes
- Convolutional Codes & Viterbi Decoding
  - Example: Rate ½ 4 State Code
  - General Description of Convolutional Codes
  - Punctured Codes
  - Decoding and the Viterbi Algorithm
  - Turbo codes

BINARY ERROR CORRECTING CODES: (ECC)

- **2^k equally likely messages can be represented by k binary digits.**
- **If these k digits are not coded, an error in one or more of the k binary digits will result in the wrong message being received.**
- **Error correcting codes is a technique whereby more than the minimum number of binary digits are used to represent the messages.**
- **The aim of the extra digits, called redundant or parity digits, is to detect and hopefully correct any errors that occurred in transmission.**

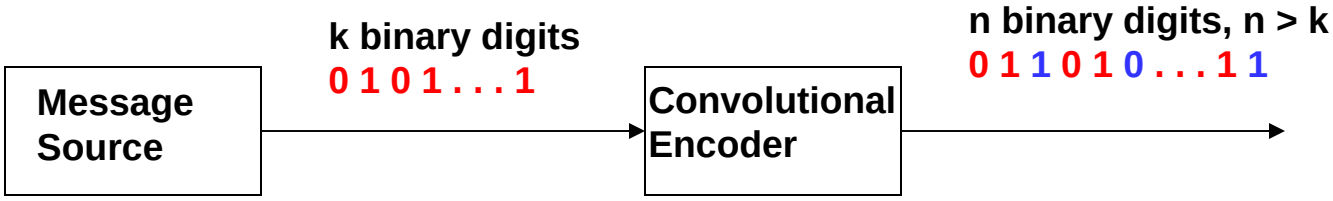
TWO TYPES OF BINARY CODES

- **Block Codes**



Rate = k / n

- **Convolutional Codes**



Rate = k / n

TYPES OF ECC

- **Binary Codes**
  - Encoder and decoder works on a bit basis.
- **Nonbinary Codes**
  - Encoder and decoder works on a byte or symbol basis.
  - Bytes usually are 8 bits but can be any number of bits.
  - Galois field arithmetic is used.
  - Example is a Reed Solomon Code
  - More generally, we can have codes where the number of symbols is a prime or a power of a prime.

TYPES OF DECODERS – BINARY CASE

- **Hard input decoders**
  - **Input to decoders are 0's and 1's.**
- **Soft input decoders**
  - **Input to decoders are probabilities of 0's and 1's.**
- **Hard output decoders**
  - **Output of decoders are 0's and 1's.**
- **Soft output decoders**
  - **Output of decoders are probabilities of 0's and 1's.**

Error Correcting and Detecting Codes

- **Binary block codes are easy to understand.**

- **Block code example:**

Information	Codeword
00	000101
01	010010
10	101101
11	111010

Which codeword was transmitted?

(a) Receive: 111011

(b) Receive: 100101

HAMMING BINARY BLOCK CODE WITH k=4 AND n=7

- **In general, a block code with k information digits and block length n is called an (n,k) code.**
- **Thus, this example is called an (7,4) code.**
- **This is a very special example where we use pictures to explain the code. Other codes are NOT explainable in this way.**
- **All that we need to know is modulo 2 addition, $\oplus$:**

  $0 \oplus 0 = 0, \quad 1 \oplus 0 = 1, \quad 0 \oplus 1 = 1, \quad 1 \oplus 1 = 0.$

HAMMING BINARY BLOCK CODE WITH k=4 AND n=7

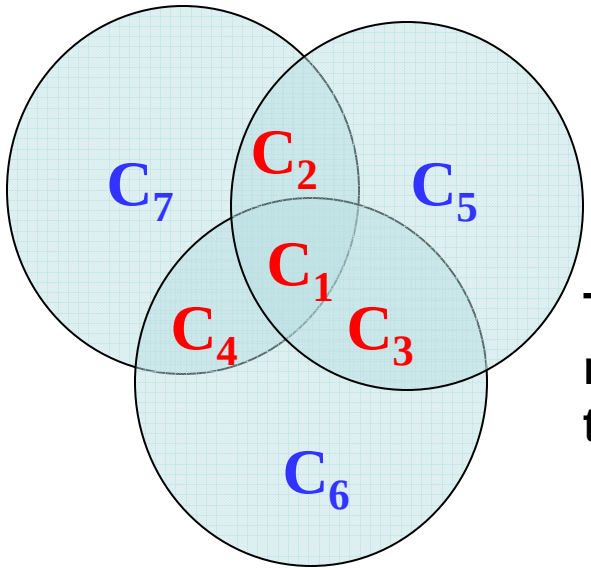
- **Message digits:** $C_1\ C_2\ C_3\ C_4$
- **Code word** $C_1\ C_2\ C_3\ C_4\ C_5\ C_6\ C_7$

Parity Check Equations:

$$C_1 \oplus C_2 \oplus C_3 \oplus C_5 = 0$$
$$C_1 \oplus C_3 \oplus C_4 \oplus C_6 = 0$$
$$C_1 \oplus C_2 \oplus C_4 \oplus C_7 = 0$$

Parity Check Matrix:

$$\begin{matrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{matrix}$$

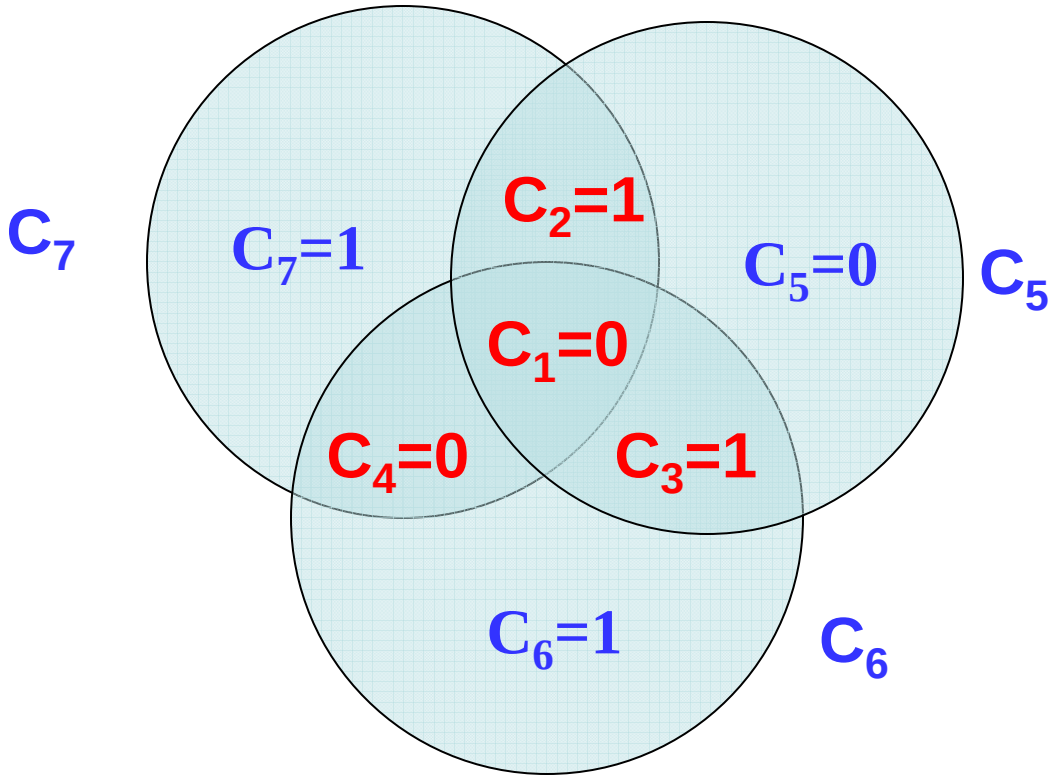


The circles represent the equations.

There is an even number of 1's in each circle.

HAMMING (7,4) CODE: ENCODING

- **Message:** $(C_1\ C_2\ C_3\ C_4) = (0\ \ 1\ \ 1\ \ 0)$

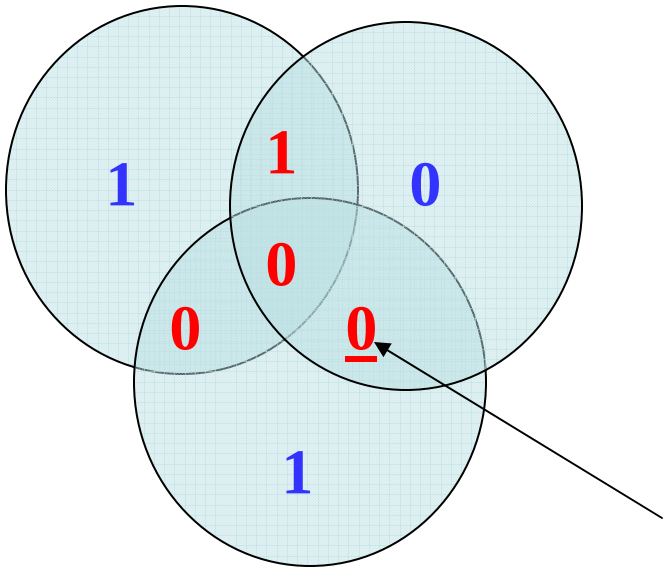


- **Resultant code word:** $0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 1$

HAMMING (7,4) CODE: DECODING

- **Transmitted code word:** **0 1 1 0 0 1 1**

- **Example 1:** **Received block with one error in a message bit.** **0 1 0 0 0 1 1**



By counting 1's in each circle we find:

There is an error in right circle.

There is an error in bottom circle

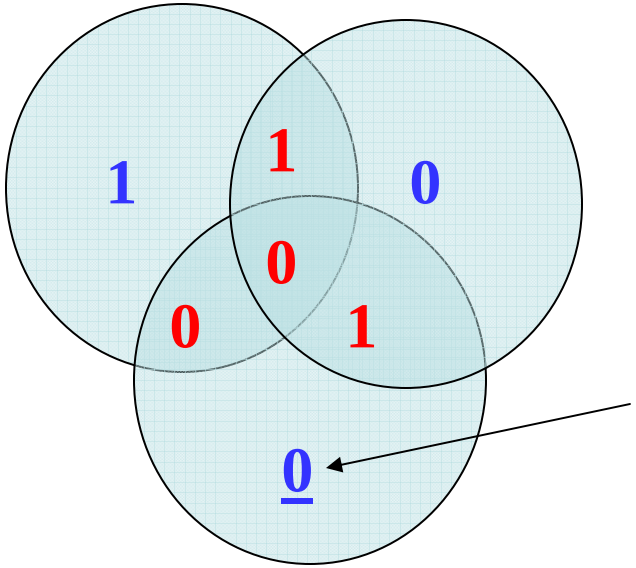
There is no error in left circle.

Therefore the error is in the third digit!

HAMMING (7,4) CODE: DECODING

- **Transmitted code word:** 0 1 1 0 0 1 1

- **Example 2:** Received block with one error in parity bit: 0 1 1 0 0 0 1



There is no error in right circle.

There is an error in bottom circle

There is no error in left circle.

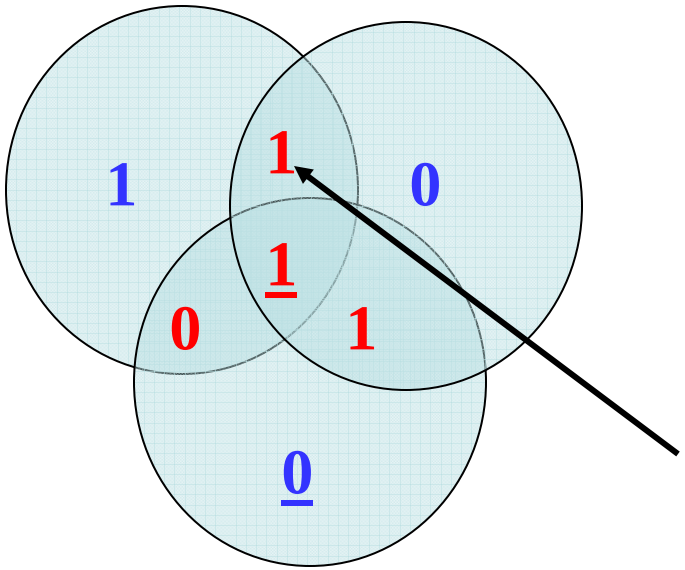
The 6th digit is in error!

HAMMING (7,4) CODE: DECODING

- **Transmitted code word:** 0 1 1 0 0 1 1

- **Example 3: Received block with two errors:**

  1 1 1 0 0 0 1



There is an error in right circle.

There is no error in bottom circle

There is an error in left circle.

The 2nd digit is in error.

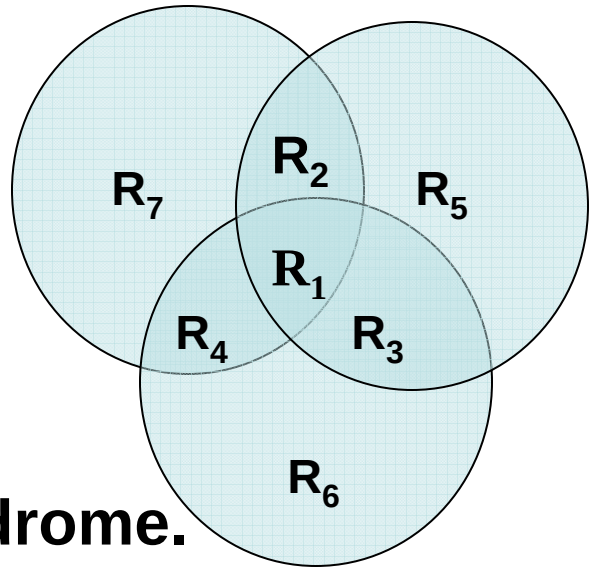
WRONG!!!

HAMMING (7,4) CODE: SYNDROME DECODING

- Let $R_1\ R_2\ R_3\ R_4\ R_5\ R_6\ R_7$ be the received block of binary digits, possibly with errors.

- Counting 1's in the circles is the same as computing the result of the following equations:

$$R_1 \oplus R_2 \oplus R_3 \oplus R_5 = S_1$$
$$R_1 \oplus R_3 \oplus R_4 \oplus R_6 = S_2$$
$$R_1 \oplus R_2 \oplus R_4 \oplus R_7 = S_3$$

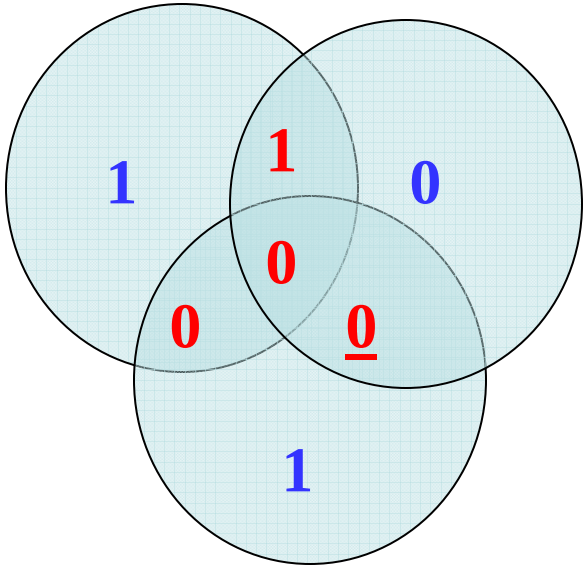


- S_1, S_2 and S_3 is called the syndrome.

HAMMING (7,4) CODE: SYNDROME DECODING

- **Resultant code word:** 0 1 1 0 0 1 1

- **Example 1:** Received block with one error in a message bit. 0 1 0 0 0 1 1



There is an error in right circle. $S_1 = 1$

There is an error in bottom circle. $S_2 = 1$

There is no error in left circle. $S_3 = 0$

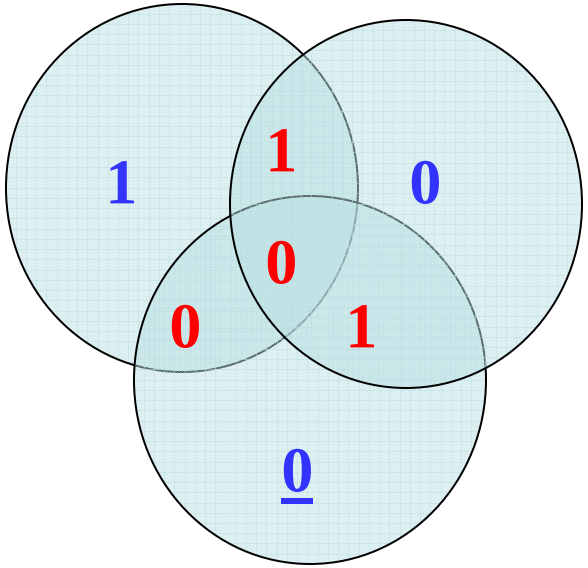
Parity Check Matrix:

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

HAMMING (7,4) CODE: SYNDROME DECODING

- **Transmitted code word:** 0 1 1 0 0 1 1

- **Example 2:** Received block with one error in parity bit: 0 1 1 0 0 0 1



There is no error in right circle. $S_1=0$

There is an error in bottom circle. $S_2=1$

There is no error in left circle. $S_3=0$

Parity Check Matrix:

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

HAMMING (7,4) CODE: SYNDROME DECODING

- **Thus to correct a single error based upon the received sequence R_1, R_2, R_3, R_4, R_5, R_6, R_7:**
  - **one can first compute the syndrome S_1, S_2, S_3 ,**
  - **and then compare it with the columns of the parity check matrix.**
  - **The matching column is where the error occurred.**
- **This technique will work for any single error correcting code.**

HAMMING (7,4) CODE

- **Another way of decoding is to compare the received sequence to all of the code words and choose the one that is "closest" to it, that is differs from it in the fewest number of positions.**

- **The list of 16 code words for this code is shown on the next slide.**

- **No matter how we decode, if more than one error occurs in the block of 7 digits the decoder will decode to the wrong code word.**

LIST OF CODE WORDS: HAMMING (7,4) CODE

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

PROPERTIES OF BINARY PARITY CHECK CODES

- **An (n,k) binary parity check code (also called an (n,k) group code) is a set of code words of length n, which consist of all of the binary n-vectors which are the solutions of r = (n-k) linearly independent equations called parity check equations.**

- **Each parity check equation specifies a subset of the components of the n-vector which sum to 0, modulo 2.**

- **If one has r = (n-k) linearly independent equations, there will be some set of k of the components of the n-vectors which can be arbitrarily specified such that one can solve for the other r = (n-k) components.**

PROPERTIES OF BINARY PARITY CHECK CODES

- **The k components that are specified are called information digits (or message digits) and the other r = (n-k) components are called parity digits (or redundant digits).**

- **Since there are a set of k binary symbols that can be chosen arbitrarily, these symbols can be filled in 2^k different ways.**

- **Thus the complete list of code words contains 2^k code words.**

- **Note that the all-zero vector always satisfies these parity check equations since any subset of the components of the all-zero vector sums to 0 modulo 2.**

PROPERTIES OF BINARY PARITY CHECK CODES

- **The coefficients of the r = (n-k) linearly independent parity check equations can be written as a matrix called the parity check matrix and is denoted H.**

- **The parity check matrix has r rows and n columns.**

- **The i-jth entry (ith row and jth column) in this parity check matrix, $h_{i,j}$, is equal to 1 if and only if the jth component of a code word is contained in the ith parity check equation. Otherwise it is equal to 0.**

FOR HAMMING (7,4) CODE

- **For the Hamming (7,4) code there were 3 equations**

$$C1 \oplus C2 \oplus C3 \oplus C5 = 0$$
$$C1 \oplus C3 \oplus C4 \oplus C6 = 0$$
$$C1 \oplus C2 \oplus C4 \oplus C7 = 0.$$

Thus the parity check matrix for this code is:

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

FOR HAMMING (7,4) CODE

- **For the Hamming (7,4) code there were 3 linearly independent equations**

$$C1 \oplus C2 \oplus C3 \oplus C5 = 0$$
$$C1 \oplus C3 \oplus C4 \oplus C6 = 0$$
$$C1 \oplus C2 \oplus C4 \oplus C7 = 0$$

so r=3 and k=4. Thus there are $2^4 = 16$ code words in this code.

- **Note that the all-zero vector is a code word.**

PROPERTIES OF BINARY PARITY CHECK CODES

- **Since the parity check equations are linear (modulo 2), if $\underline{C}_1$ is a solution of the equations and if $\underline{C}_2$ is a solution to the equations, then $\underline{C}_1 \oplus \underline{C}_2$ is also a solution to the equations.**

- **Thus the modulo 2 sum of any two code words is a code word.**

- **Consider the set of k distinct code words each of which had a single 1 in one of the information positions. Any of the 2^k code words can be constructed by taking a linear combination of these k code words.**

- **These k code words are said to be generators of the code.**

PROPERTIES OF BINARY PARITY CHECK CODES

- **A k row by n column matrix made up of these code words is called the generator matrix, G, of the code.**

- **We will assume that the components of the code words are ordered so that the first k components are the message digits. Then the rows of G can be ordered so that there is a k by k unit matrix on the left.**

LIST OF CODE WORDS: HAMMING (7,4) CODE

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

This is the generator matrix of the code.

FOR HAMMING (7,4) CODE

- **In the list of 16 code words for the (7,4) Hamming code, the 16 code words can be formed by taking all of the linear combinations of the code words having a single 1 in the information positions. These were:**

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

This 4-row by 7-column matrix is the generator matrix, G, of the code. Note the 4 by 4 unit matrix on the left.

PROPERTIES OF BINARY PARITY CHECK CODES

- **For any (n,k) binary code, assume the parity check matrix H is of the form:**

$$H = [A \quad 1_{r,r,}]$$

where A is an arbitrary (n-k) by k binary matrix and where $1_{r,r}$ is an r by r unit matrix.

- **Then G is of the form:**

$$G = [1_{k,k} \quad A^T]$$

where $1_{k,k}$ is a k by k unit matrix and A^T is A transpose. The proof follows.

PROPERTIES OF BINARY PARITY CHECK CODES

- **Since every code word $\underline{C}$ must satisfy the parity check equations, this says that $\underline{C}$ must satisfy the matrix vector equation:**

$$H\,\underline{C} = \underline{0}.$$

Here we are assuming that $\underline{C}$ and $\underline{0}$ are column vectors of dimension n and r=(n-k) respectively.

- **But since the rows of G are all code words, the H and G must satisfy the matrix equation:**

$$H\,G^t = 0.$$

Here G^t is the transpose of the matrix G.

PROPERTIES OF BINARY PARITY CHECK CODES

- **Proof:**

$$H G^t = [A \;\; 1_{r,r,}] \, [1_{k,k} \;\; A^T]^T$$

$$= [A \;\; 1_{r,r,}] \begin{bmatrix} 1_{k,k} \\ A \end{bmatrix}$$

$$= A \oplus A = 0$$

FOR HAMMING (7,4) CODE

- **The parity check matrix for this code is:**

$$\left[\begin{array}{ccccc:ccc} 1 & 1 & 1 & 0 & & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & & 0 & 0 & 1 \end{array}\right]$$

and the generator matrix is:

$$\left[\begin{array}{ccccc:ccc} 1 & 0 & 0 & 0 & & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & & 0 & 1 & 1 \end{array}\right]$$

PROPERTIES OF BINARY PARITY CHECK CODES

- If $\underline{X}$ and $\underline{Y}$ are any two binary vectors of the same length, define the **Hamming distance** between $\underline{X}$ and $\underline{Y}$, denoted $d_H(\underline{X},\underline{Y})$, as the number of positions in which $\underline{X}$ and $\underline{Y}$ differ.

- For any binary vector $\underline{Z}$, define the **Hamming weight** of $\underline{Z}$, denoted $w_H(\underline{Z})$, as the number of 1's in the vector $\underline{Z}$.

- Then it is easy to see that $d_H(\underline{X},\underline{Y}) = w_H(\underline{X} \oplus \underline{Y})$.

PROPERTIES OF BINARY PARITY CHECK CODES

- **The minimum Hamming distance of a code C, denoted $d_{min}(C)$, is defined as the minimum Hamming distance between any two distinct code words in C.**

- **For any two code words, $\underline{C}_i$ and $\underline{C}_j$,**

$$\underline{C}_i \oplus \underline{C}_j = \underline{C}_k.$$

- **But then,**

$$d_H(\underline{C}_i , \underline{C}_j) = w_H(\underline{C}_i \oplus \underline{C}_j) = w_H(\underline{C}_k)$$

- **Thus, $d_{min}(C)$ is equal to the minimum Hamming weight of any non-zero code word.**

HAMMING (7,4) CODE

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

For this (7,4) Hamming code, $d_{min} = 3$.

PROPERTIES OF BINARY PARITY CHECK CODES

- **For any code that has minimum distance d_{min}:**
  - **The code can detect any pattern of $(d_{min} - 1)$ or fewer errors.**
  - **The code can fill in any pattern of $(d_{min} - 1)$ or fewer erasures.**
  - **The code can correct any pattern of $int[(d_{min} - 1)/2]$ or fewer errors.**
  - **The code can simultaneously fill in e or fewer erasures and correct t or fewer errors if:**

$$2t + e \leq (d_{min} - 1).$$

HAMMING (7,4) CODE

- **Since the (7,4) Hamming code has minimum distance $d_{min} = 3$:**
  - **The code can detect any pattern of 2 or fewer errors. It can detect many more error patterns than that. This will be discussed later.**
  - **The code can correct any single error.**
  - **The code can fill in any pattern of 2 or fewer erasures. It can sometimes fill in 3 erasures.**

PROPERTIES OF BINARY PARITY CHECK CODES

- **Since every code word $\underline{C}$ must satisfy the parity check equations, then $\underline{C}$ must satisfy the equation:**

$$H\,\underline{C} = \underline{0}.$$

- **Assume that $\underline{C}$ is a code word that has d 1's and (n- d) 0's. Then, d columns of H must sum to $\underline{0}$.**

- **The smallest value of d for which this is true is $d = d_{min}$. Thus d_{min} columns of H sum to 0 and no fewer than d_{min} columns of H sum to 0.**

- **Said in another way, a code has minimum distance d_{min}, if and only if d_{min} columns of H sum to $\underline{0}$ and no fewer than d_{min} columns of H sum to $\underline{0}$.**

HAMMING (7,4) CODE

- **Consider the parity check matrix for the Hamming (7,4) code:**

1	**1**	**1**	**0**	**1**	**0**	**0**
1	**0**	**1**	**1**	**0**	**1**	**0**
1	**1**	**0**	**1**	**0**	**0**	**1**

- **No single column is 0 and no two columns sum to 0. (Two columns sum to 0 iff the columns are the same.)**

- **But there are many instance where 3 columns sum to 0: e.g., the 4^{th}, 6^{th} and 7^{th} column of the parity check matrix.**

- **Thus $d_{min} = 3$ for the code.**

PROPERTIES OF BINARY PARITY CHECK CODES

- **Sometimes we modify a code by adding one more parity digit, called an overall parity digit.**

- **The equation corresponding to this extra parity digit is such that the modulo 2 summation of all of the digits in the code word (including this overall parity digit) is equal to 0.**

- **The result is that the parity check matrix is modified by adding an extra row of all 1's and a column on the right of all 0's and a 1 at the bottom.**

PROPERTIES OF BINARY PARITY CHECK CODES

- **This overall parity digit insures that every code word has even Hamming weight.**

- **Thus if an overall parity digit is appended to a code that had an odd minimum Hamming distance, d_{min}, then the new code has a minimum distance (d_{min} +1).**

- **However, the new code has one more parity digit and the same number of information digits as the original code. (The new code has block length one more than the original code.)**

MODIFYING A HAMMING (7,4) CODE

- **Original (7,4) code had a parity check matrix given as:**

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

- **The new code is an (8,4) code with parity check matrix:**

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

- **The new code has $d_{min} = 4$.**

MODIFYING A HAMMING (7,4) CODE

- **But this parity check matrix does not have a unit matrix on the right. We can make this happen by replacing the last equation with the sum of all of the equations resulting in the parity check matrix:**

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

- **Note that d_{min} = 4 since 4 columns sum to 0 (e.g., the 1st, 5th, 6th and 7th) but no fewer than 4 columns sum to 0.**

HAMMING (8,4) CODE

- **The code words in the new code are:**

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

DECODING OF BINARY PARITY CHECK CODES

- **Assume that one is using a code with parity check matrix H and that the transmitter transmits a code word C.**

- **Assume the receiver receives the binary vector R where R = C ⊕ E. Thus E = C ⊕ R.**

- **E is called the error vector and has a 1 in those positions where C and R differ (i.e., where there are errors) and 0's elsewhere.**

SYNDROME DECODING OF BINARY PARITY CHECK CODES

- **The decoder first forms the syndrome $\underline{S}$ using the parity check matrix $\underline{H}$ and $\underline{R}$ by calculating:**

$$\underline{S} = H\,\underline{R}.$$

- **Note that since $\underline{R} = \underline{C} \oplus \underline{E}$ and since $H\,\underline{C} = \underline{0}$, then**

$$\underline{S} = H\,\underline{R} = H(\underline{C} \oplus \underline{E}) = H\underline{C} \oplus H\underline{E} = \underline{0} \oplus H\underline{E}.$$

- **Thus $\underline{S} = H\underline{E}$. This says that the syndrome, $\underline{S}$, is equal to the modulo 2 summation of those columns of H where the errors occurred.**

SYNDROME DECODING OF BINARY PARITY CHECK CODES

- **But there are many solutions for $\underline{E}$ to the equation**

$$\underline{S} = H\underline{E}.$$

- **In fact for each possible syndrome $\underline{S}$ there are 2^k different vectors $\underline{E}$ that satisfy that equation since if $\underline{E}$ is a solution so is $\underline{E} \oplus \underline{C}$ for any code word $\underline{C}$.**

- **For a random error channel with bit error probability $p < 0.5$, the most likely solution for $\underline{E}$ is the one that corresponds to the fewest errors. This means choosing the vector $\underline{E}$ with the fewest non-zero components.**

BINARY PARITY CHECK CODES: ENCODING

- **There are many circuits that are used in encoding binary parity check codes.**

- **For any code, if k is not too large, one can use a table of size 2^k by r, where we input the k information digits as an address and look up the r parity digits.**

ENCODING THE HAMMING (7,4) CODE USING A TABLE

- **Parity check matrix:**

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

- **Encoding table:**

Information Digits	Parity Digits
0 0 0 0	0 0 0
0 0 0 1	0 1 1
0 0 1 0	1 1 0
0 0 1 1	1 0 1
. . . .	. . .
. . . .	. . .
1 1 1 1	1 1 1

BINARY PARITY CHECK CODES: M.L. DECODING

- **If the code words are transmitted with equal apriori probability over a B.S.C. with error probability p, p < 0.5, a decoder which results in the smallest probability of word error is as follows:**

  Compare the received vector R with every code word and choose the code word that differs from it in the fewest positions.

- **This is like the optimal detector found in ECE 154B for the Gaussian channel but we here we use Hamming distance instead of Euclidean distance.**

- **Since there are 2^k code words, this is impractical if k is large.**

BINARY PARITY CHECK CODES: SYNDROME DECODING

- **Assume we first compute the syndrome.**
- **For many codes, one finds the minimum Hamming weight vector $\underline{E}$ which is the solution to the equation $\underline{S} = H\underline{E}$ by algebraic means.**
- **However, if the dimension of $\underline{S}$, r, is not too big one can construct a decoding table with 2^r entries that relate the syndrome to the minimum Hamming weight error pattern, $\underline{E}$.**
- **Such a decoder would be maximum likelihood.**

DECODING THE HAMMING (7,4) CODE USING A TABLE

- **Parity check matrix:**

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

- **Decoding table:**

Syndrome	Error Pattern
0 0 0	0 0 0 0 0 0 0
0 0 1	0 0 0 0 0 0 1
0 1 0	0 0 0 0 0 1 0
0 1 1	0 0 0 1 0 0 0
1 0 0	0 0 0 0 1 0 0
1 0 1	0 1 0 0 0 0 0
1 1 0	0 0 1 0 0 0 0
1 1 1	1 0 0 0 0 0 0

BINARY PARITY CHECK CODES: SYNDROME DECODING BY TABLES

- **If both k and r are not too large, two tables can be used to do the entire decoding.**

- **The syndrome can be calculated from R by using the encoding table with 2^k entries as follows:**

  1. The first k components of R are used as the address in the encoding table and the resulting parity bits are added (bit by bit modulo 2) to the last r bits of R.

  2. The result is the syndrome S.

- **Then the syndrome is used as the address in the decoding table with 2^r entries and the error pattern is read from the table.**

- **The error pattern is then added to R to find the decoded code word.**

BINARY PARITY CHECK CODES: SYNDROME DECODING

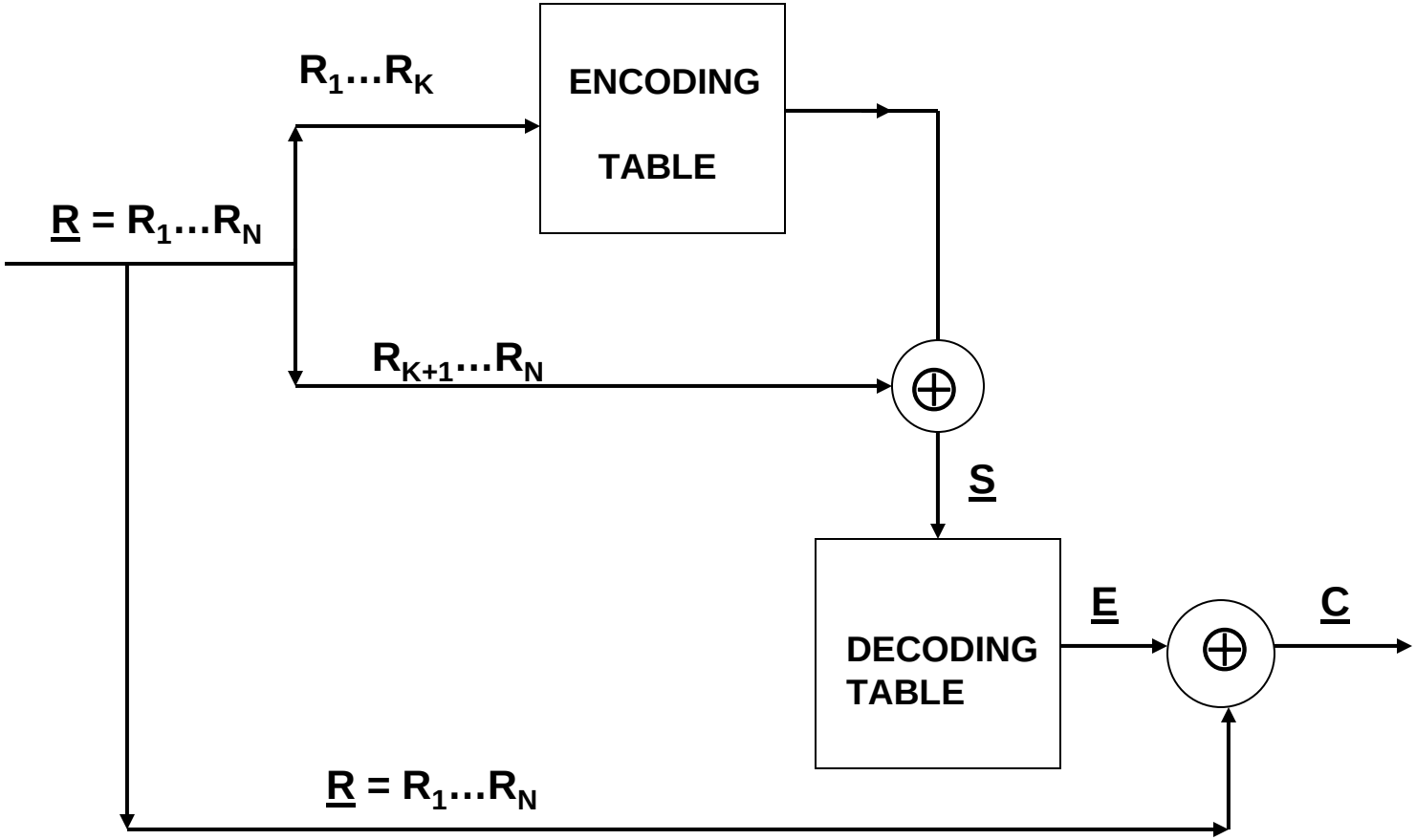
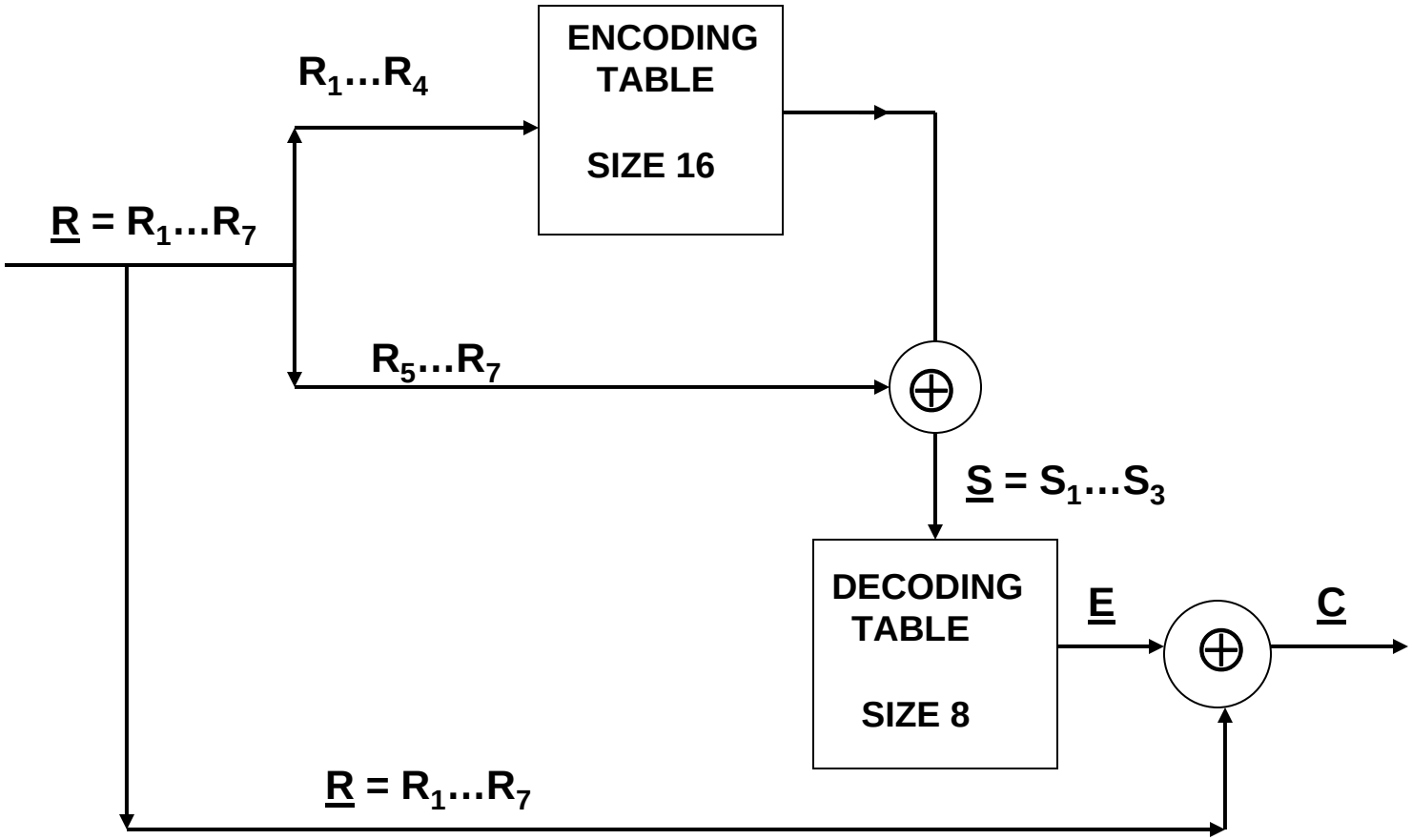


TABLE LOOK UP DECODER: HAMMING (7,4) CODE



CONSTRUCTING OTHER BINARY HAMMING CODES

- **One can construct single error correcting binary codes with d_{min}=3 having other block lengths.**

- **If one wants a code of block length n=(2^r-1) for any integer r, the n columns of the parity matrix are chosen as the (2^r-1) distinct non-zero vectors of length r. Note that since there are r rows in the parity matrix, r is the number of parity digits in the code.**

n	7	15	31	63	127	255
k	4	11	26	57	120	247
r	3	4	5	6	7	8

THE GOLAY (23,12) CODE

- **This code has d_{min}=7. A parity check matrix for this code is:**

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

THE GOLAY (23,12) CODE

- **Since the (23,12) Golay code has $d_{min}=7$, it can correct all patterns of 1, 2, or 3 errors in each code block of 23 digits.**

- **The decoder uses two table: one of size 2^{12} and the other of size 2^{11}.**

- **One can make a (24,12) code with $d_{min}=8$ by appending an overall parity digit to the (23,12) Golay code. To decode this (24,12) code one could use a two tables of size 2^{12}.**

TABLE LOOK UP DECODER: GOLAY (23,12) CODE

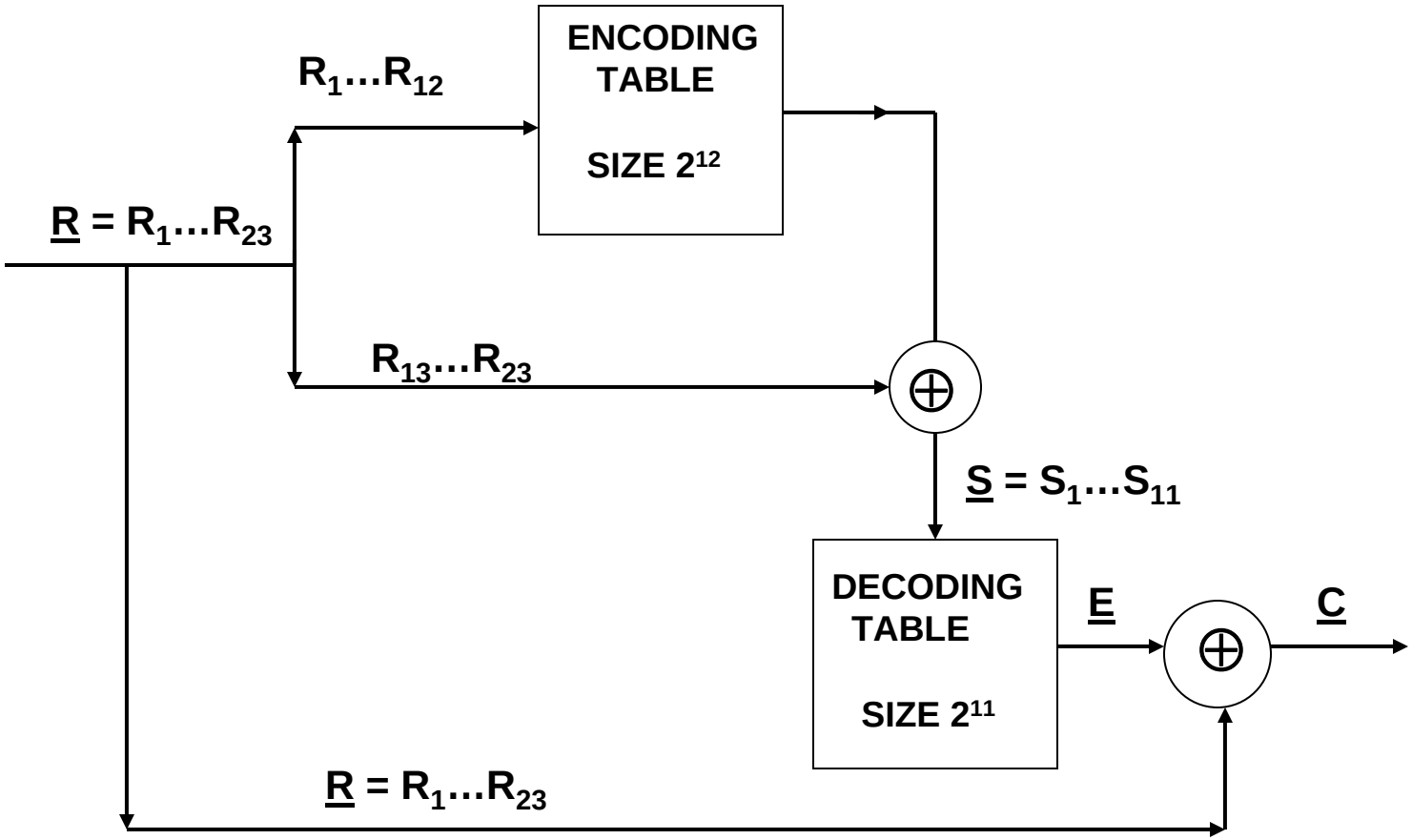
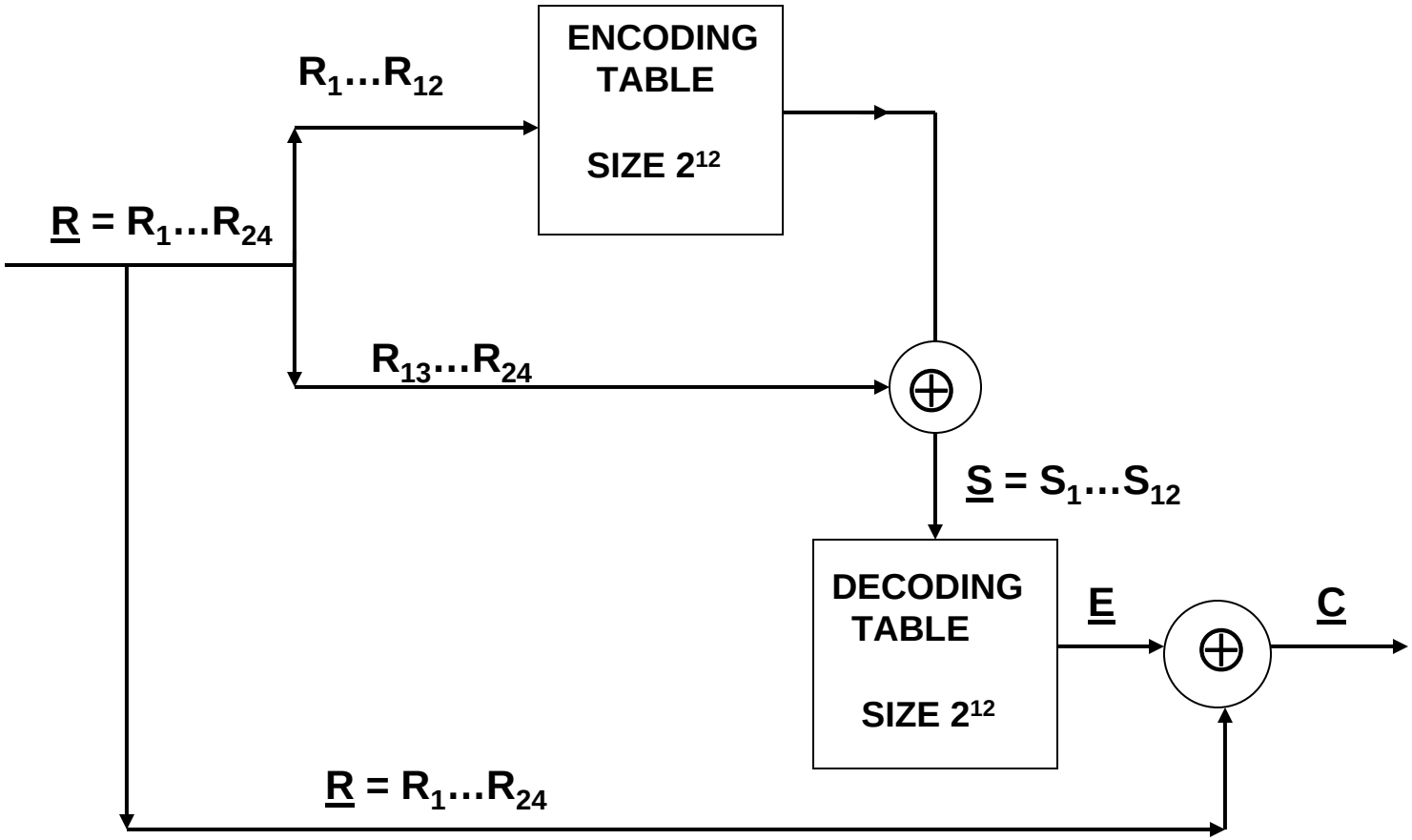


TABLE LOOK UP DECODER: GOLAY (24,12) CODE



SHORTENING BINARY PARITY CHECK CODES

- **For any positive integer "a" (a< k), if one starts with an (n,k) binary parity check code with minimum distance d_{min}, one can construct an (n-a, k-a) parity check code with minimum distance at least d_{min}.**

- **One can do this by setting the setting the first "a" information digits to 0 and not transmitting them.**

- **The parity check matrix of this (n-a,k-a) code is formed from the parity check matrix of the (n,k) code by eliminating the first "a" columns.**

SHORTENING BINARY PARITY CHECK CODES

- For example, if one shortens the (24,12) with $d_{min}=8$ by 2 digits (i.e., a=2) one would have a (22,10) code with minimum distance at least 8.

- If one shortens the code enough the minimum distance could actually increase. This is true, since the minimum distance between the remaining code words might be greater than the minimum distance between the original list of code words.

- There is no general rule, however, which predicts by how much the minimum distance would increase.

PUNCTURING BINARY PARITY CHECK CODES

- **While shortening a code reduces the number of information digits in a code, puncturing a code reduces the number of parity digits.**

- **One punctures a code by eliminating one or more of the parity equations and thus eliminating the corresponding parity digits.**

- **In general, puncturing a code reduces the minimum distance of the code but increases the code rate, R.**

ERROR DETECTION ONLY

- **If one only wants to detect errors, one can compute the syndrome to see if it is all-zero.**
  - **If the syndrome is all-zero one assumes that no errors occurred since the received vector is a code word.**
  - **If the syndrome is not all zero, one knows that errors have occurred.**

- **The only time that the decoder will be incorrect is if the error pattern itself is a code word. Then, the syndrome will be all-zero but errors will have occurred.**

- **For a binary symmetric channel, if one knows the number of code words of each Hamming weight, one can write an expression for the probability of undetected error.**

AN INTRODUCTION TO ERROR CORRECTING CODES
Part 2

Jack Keil Wolf

ECE 154 C

Spring 2010

BINARY CONVOLUTIONAL CODES

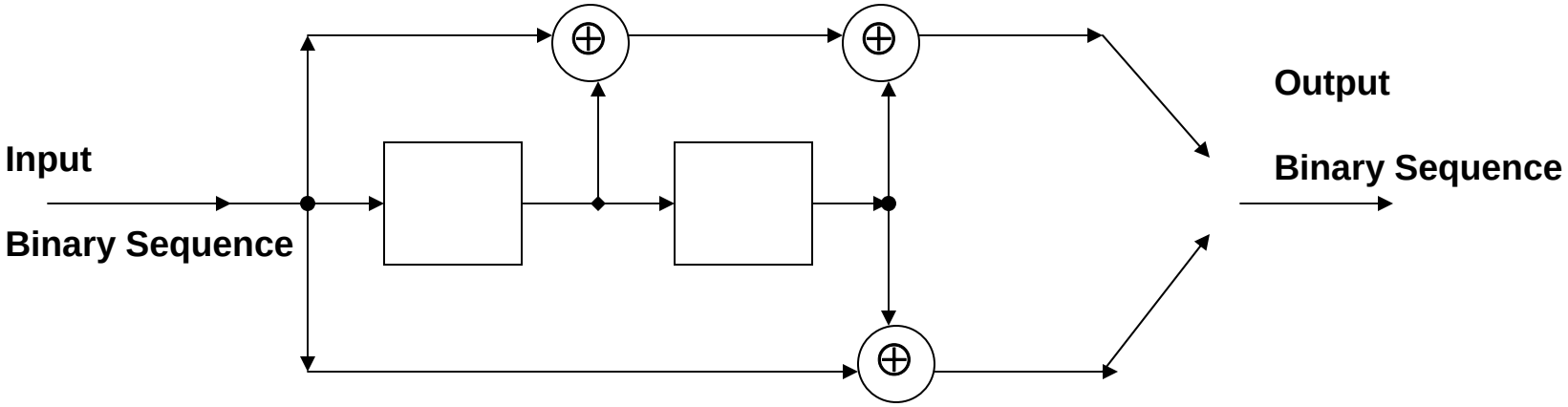
- **A binary convolutional code is a set of infinite length binary sequences which satisfy a certain set of conditions. In particular the sum of two code words is a code word.**

- **It is easiest to describe the set of sequences in terms of a convolutional encoder that produces these sequences. However for a given code, the encoder is not unique.**

- **We start with a simple example.**

A RATE ½, 4-STATE, BINARY CONVOLUTIONAL ENCODER

- **Consider the 4-state, binary, convolutional encoder shown below:**

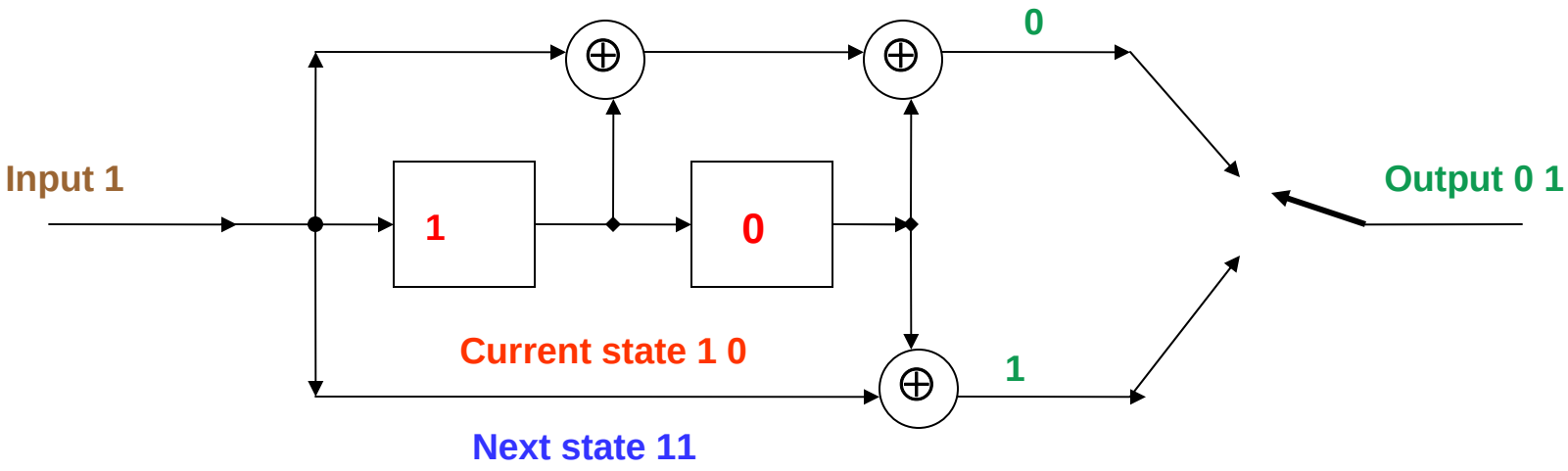


- **There are <u>two</u> output binary digits for each (<u>one</u>) input binary digit. This gives a rate ½ code.**

- **Each adder adds its two inputs modulo 2.**

A RATE ½ ENCODER

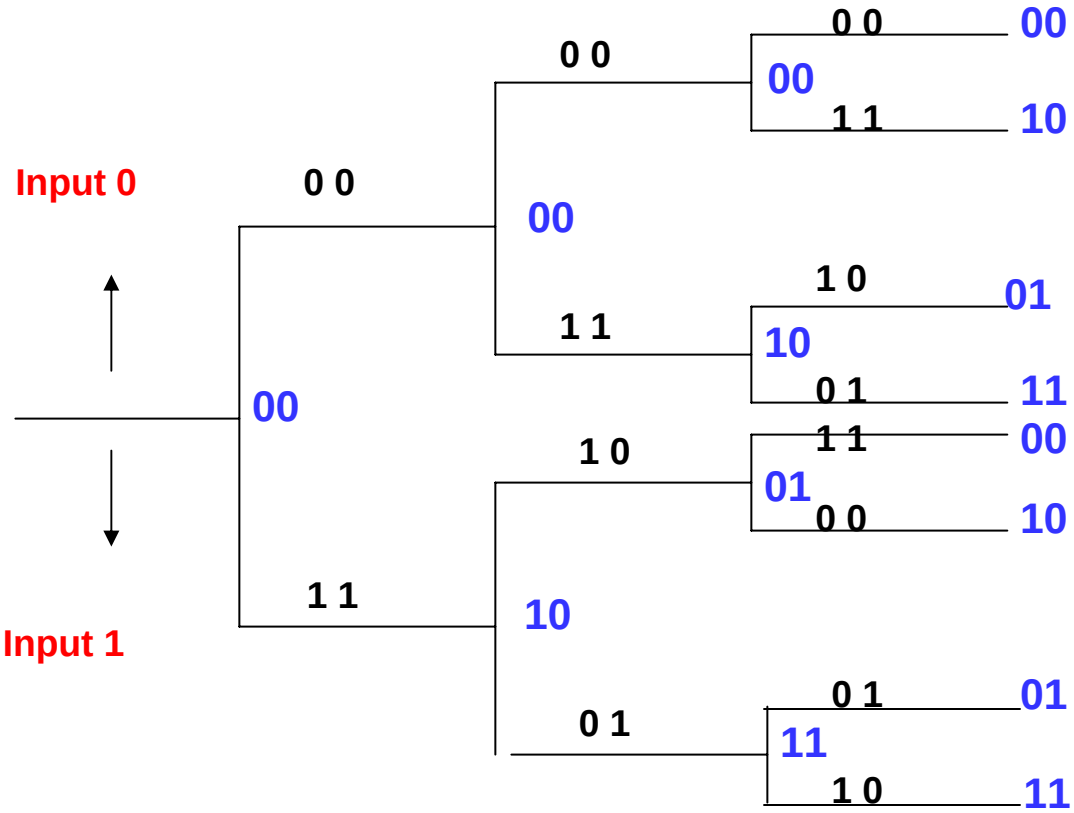
- **Consider the following encoder.**

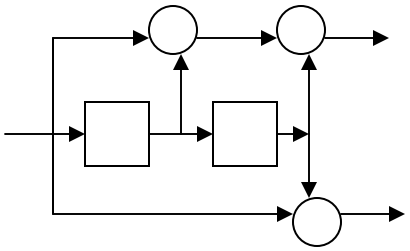


- **Assume current state is 1 0 and input is a 1.**
- **Then, the next state is 1 1.**
- **The outputs are 0 1.**

A RATE ½ ENCODER

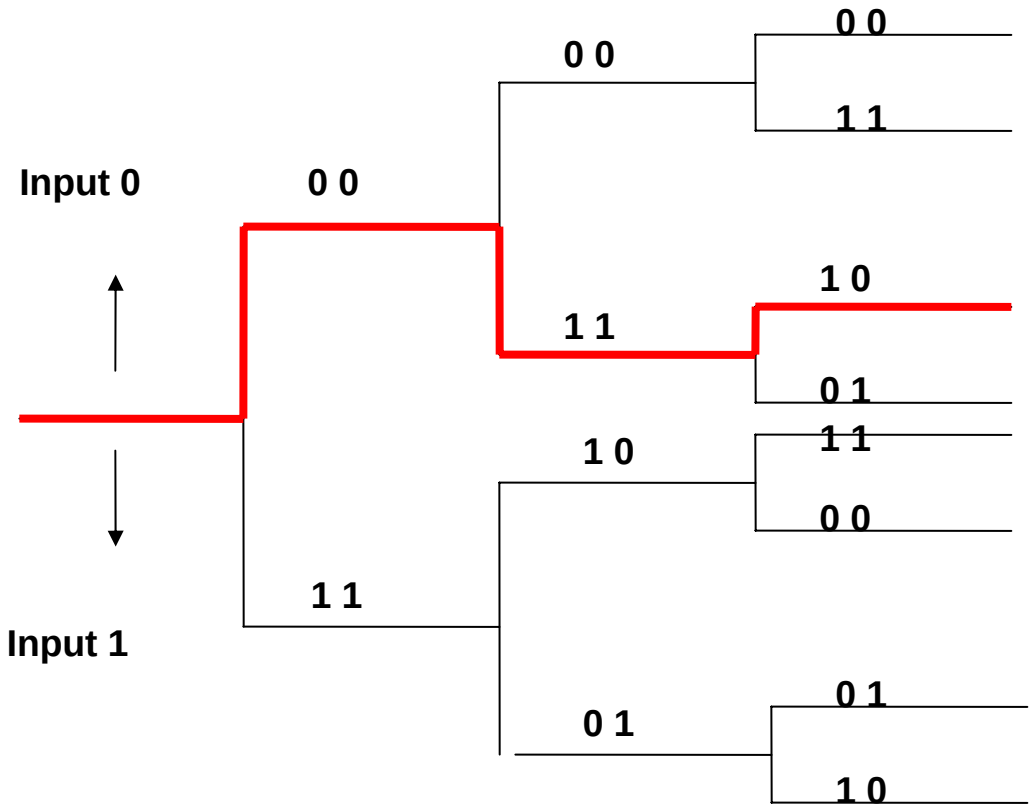
- **The output code words are the labels on the branches of all of the paths through the binary tree. The nodes in the tree are the states of the encoder. Assume we start in the 00 state.**





A RATE ½ ENCODER

- **Consider the input sequence 0 1 0 . . .**

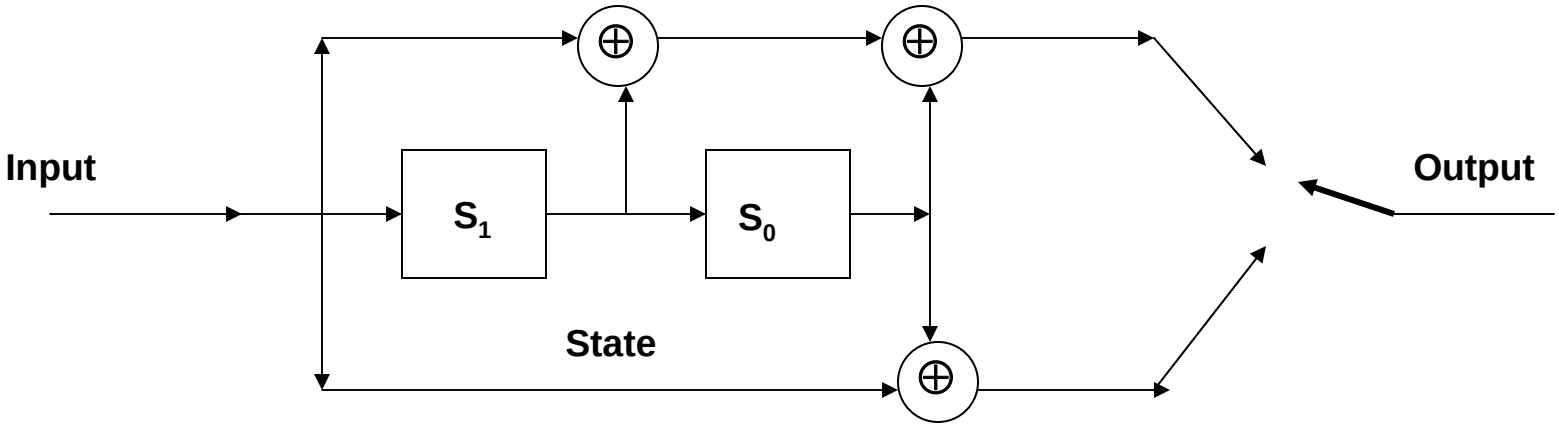


- **Then the output sequence would be**

0 0 1 1 1 0 . . .

NOTION OF STATES

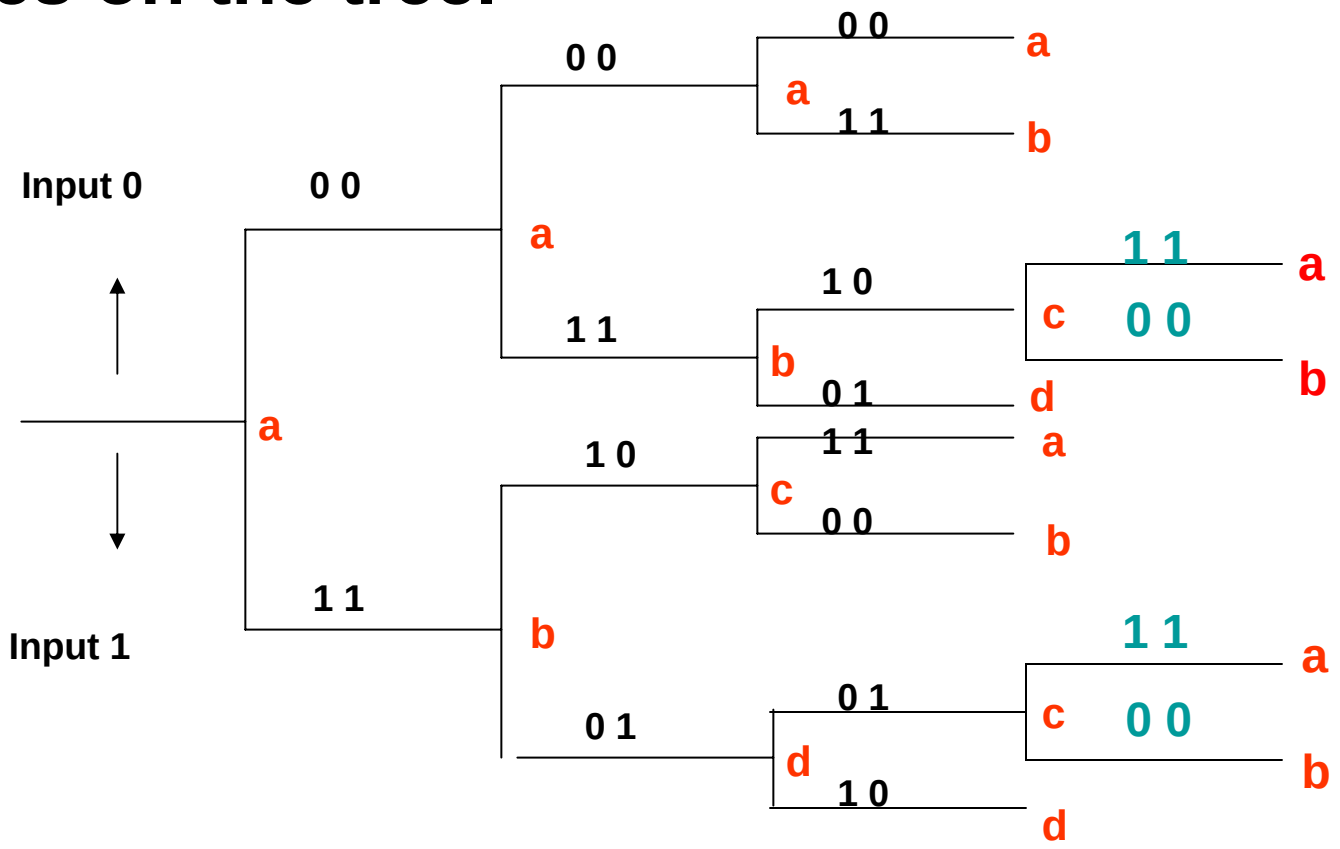
- **At any time the encoder can be in one of 4 states:**



	S_1	S_0
a	0	0
b	1	0
c	0	1
d	1	1

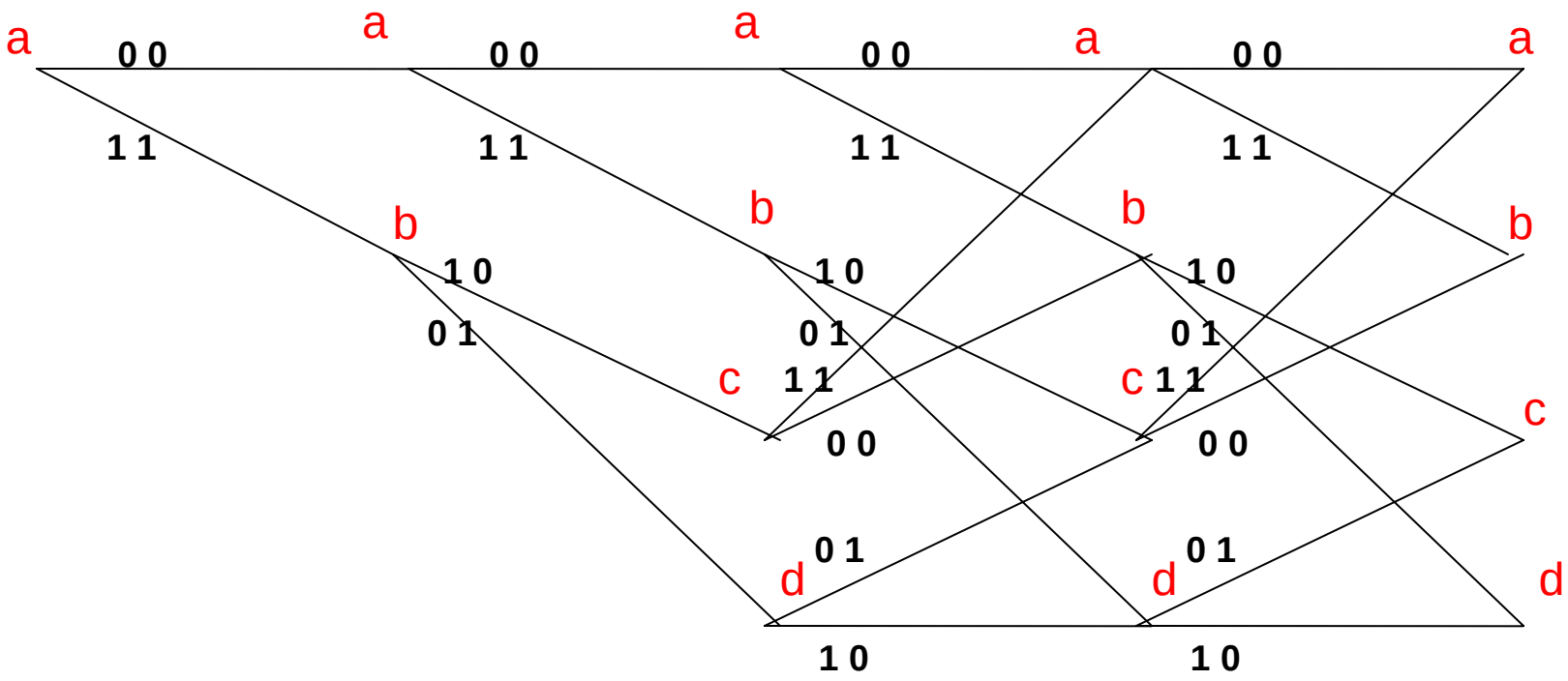
NOTION OF STATES

- **These states can be put as labels of the nodes on the tree.**



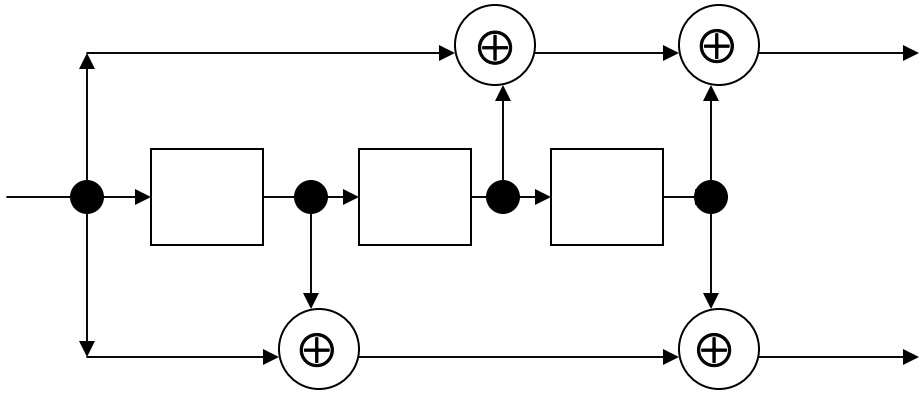
NOTION OF A TRELLIS

- **There are only 4 states. Thus, the states at the same depth in the tree can be merged, and the tree can be redrawn as a trellis:**



OTHER EXAMPLES

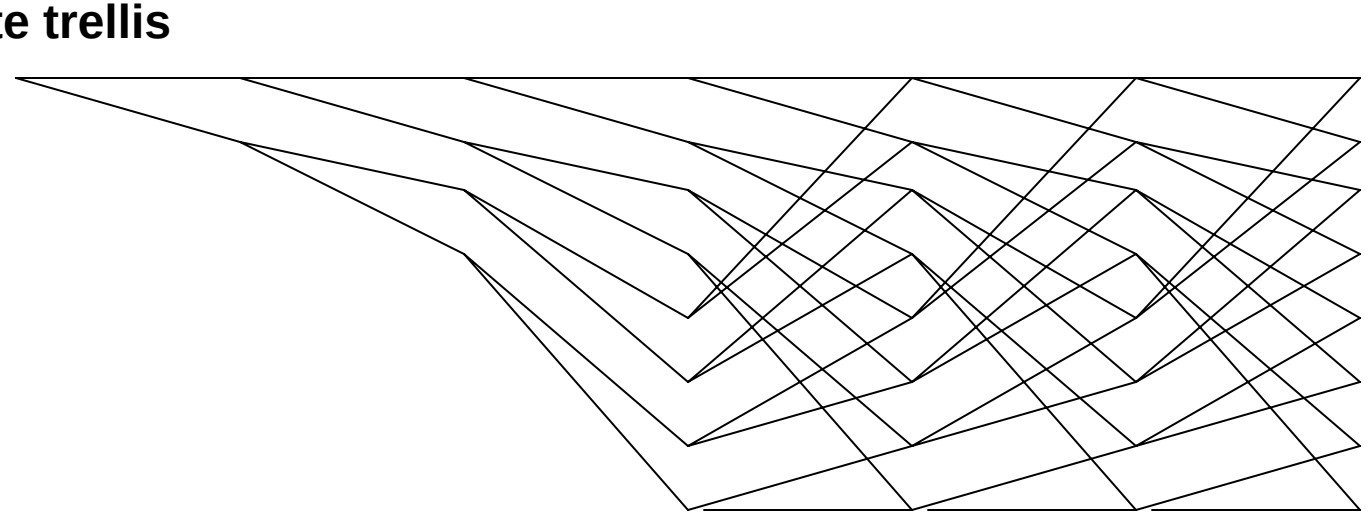
- **An 8-state rate ½ code**



Can represent the tap connections as:
Top – 1 0 1 1
Bottom – 1 1 0 1.

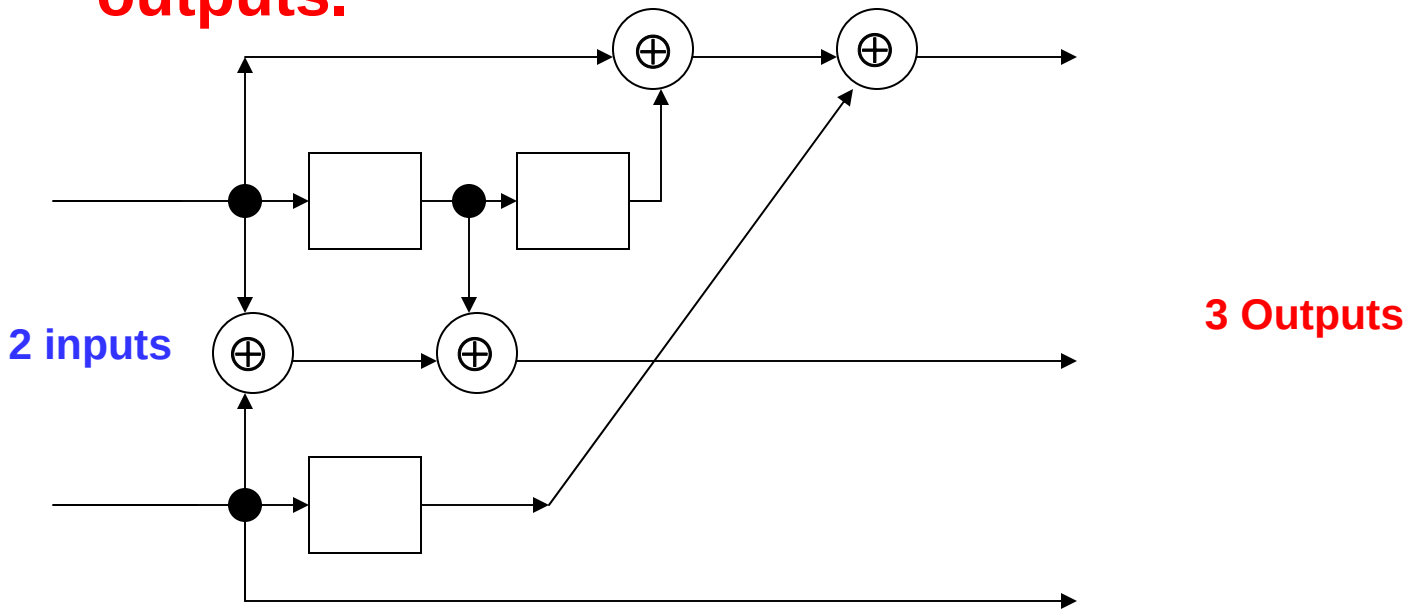
This is written in octal as:
(1,3) or (1,5).

8-state trellis



OTHER EXAMPLES

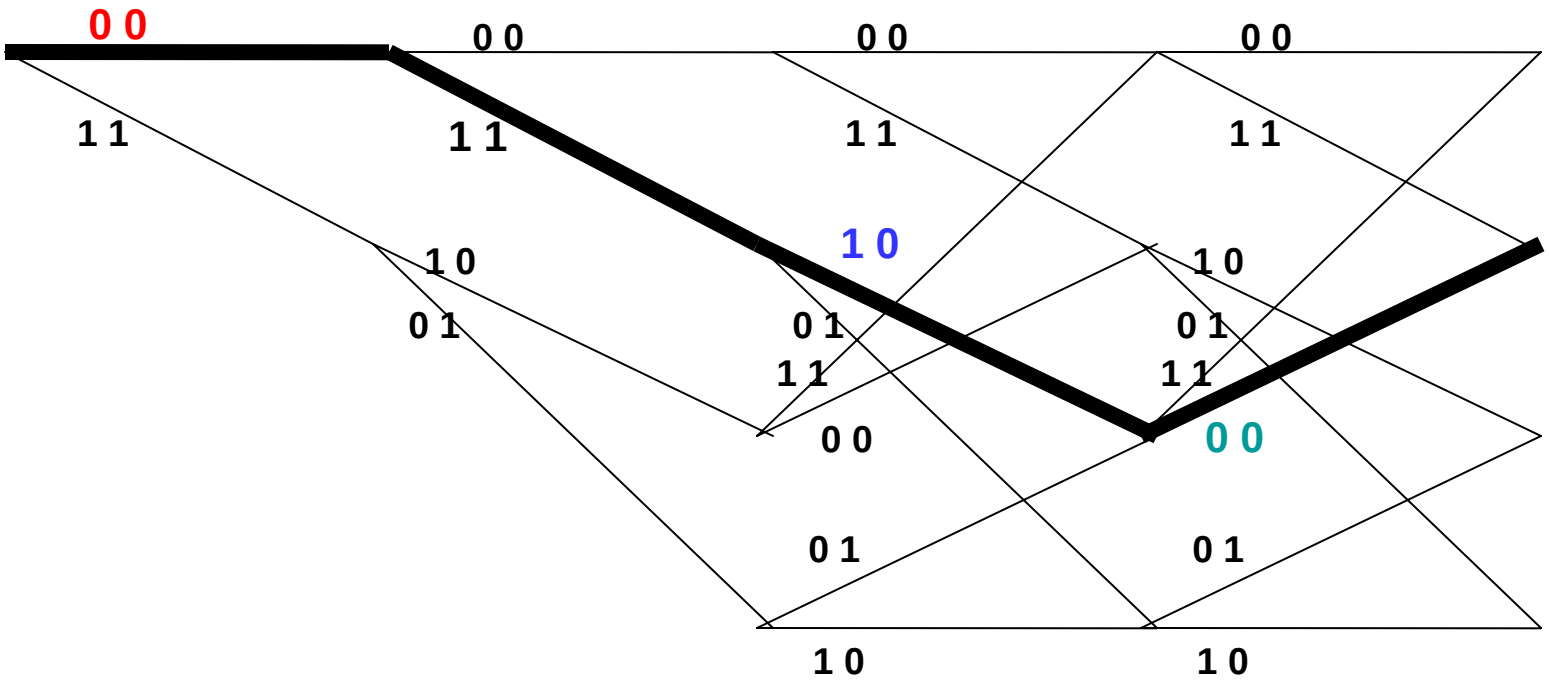
- **An 8-state, rate 2/3 encoder with 2 inputs and 3 outputs.**



- **The trellis has 8 states but 4 branches coming out of each state. There are three output binary digits on each branch. TRY DRAWING IT!!!**

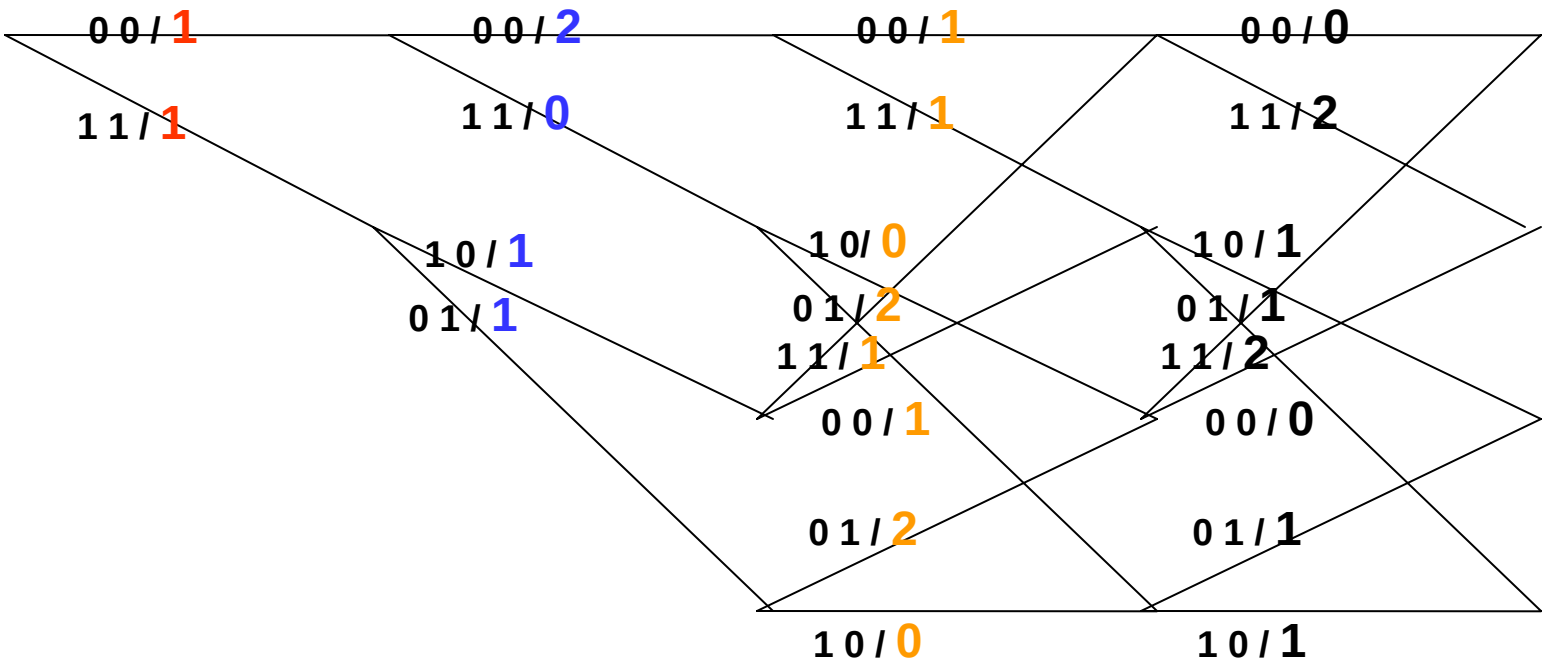
ENCODING ON THE RATE ½, 4-STATE TRELLIS

- **The path corresponding to the input 0 1 0 1 . . . is shown as the dark line and corresponds to the output 0 0 1 1 1 0 0 0 . . .**



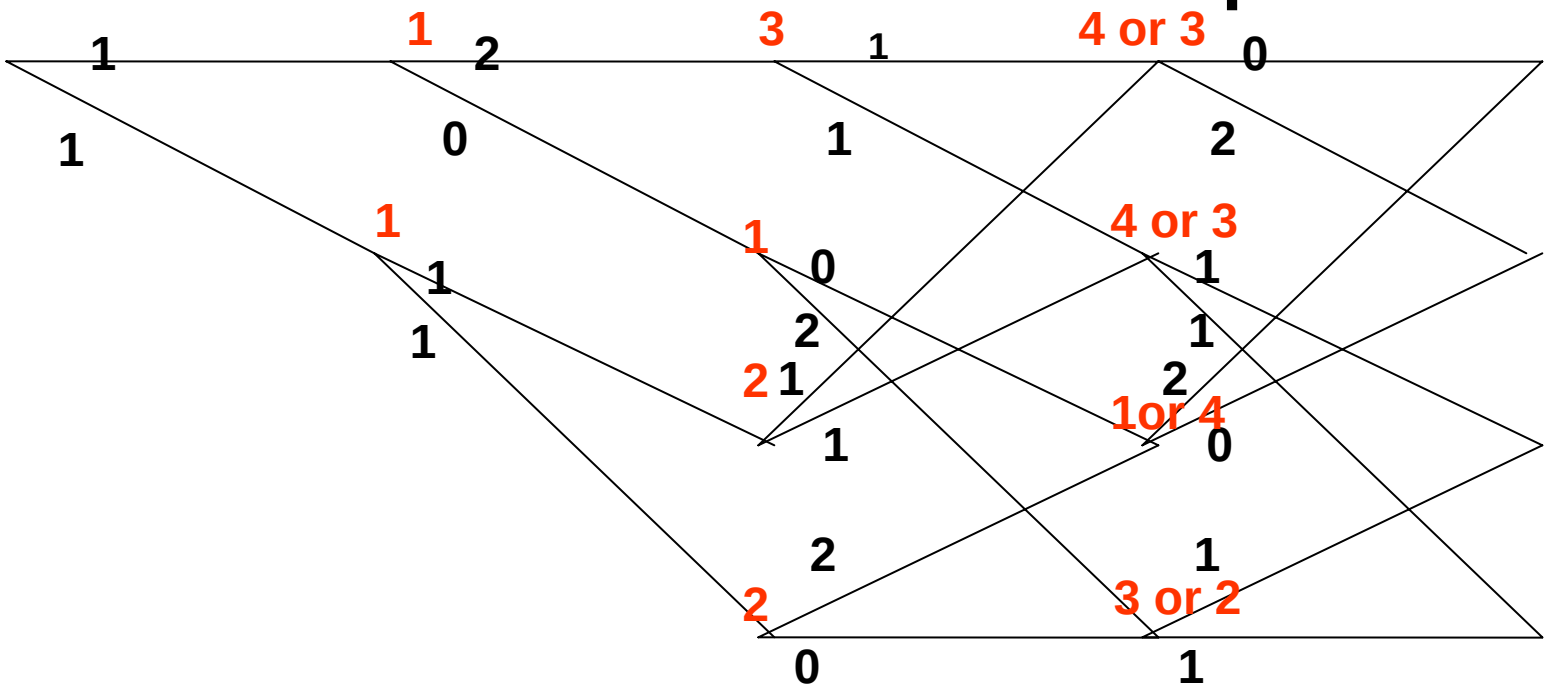
HARD DECISION DECODING ON THE RATE ½, 4-STATE TRELLIS

- **Assume that we receive 0 1 1 1 1 0 0 0 . . . We put the number of differences or "errors" on each branch.**



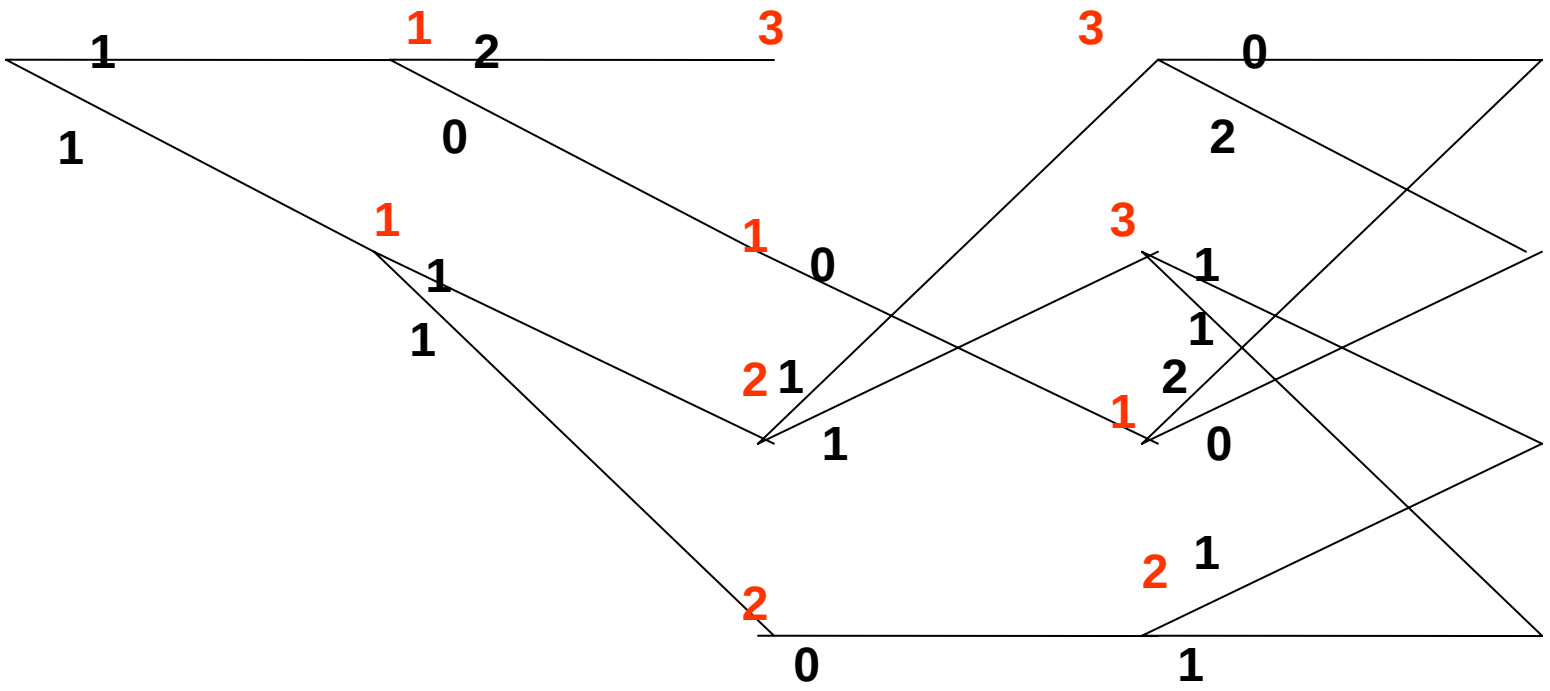
VITERBI DECODING

- **We count the total number of errors on each possible path and choose the path with the fewest errors. We do this one step at a time.**



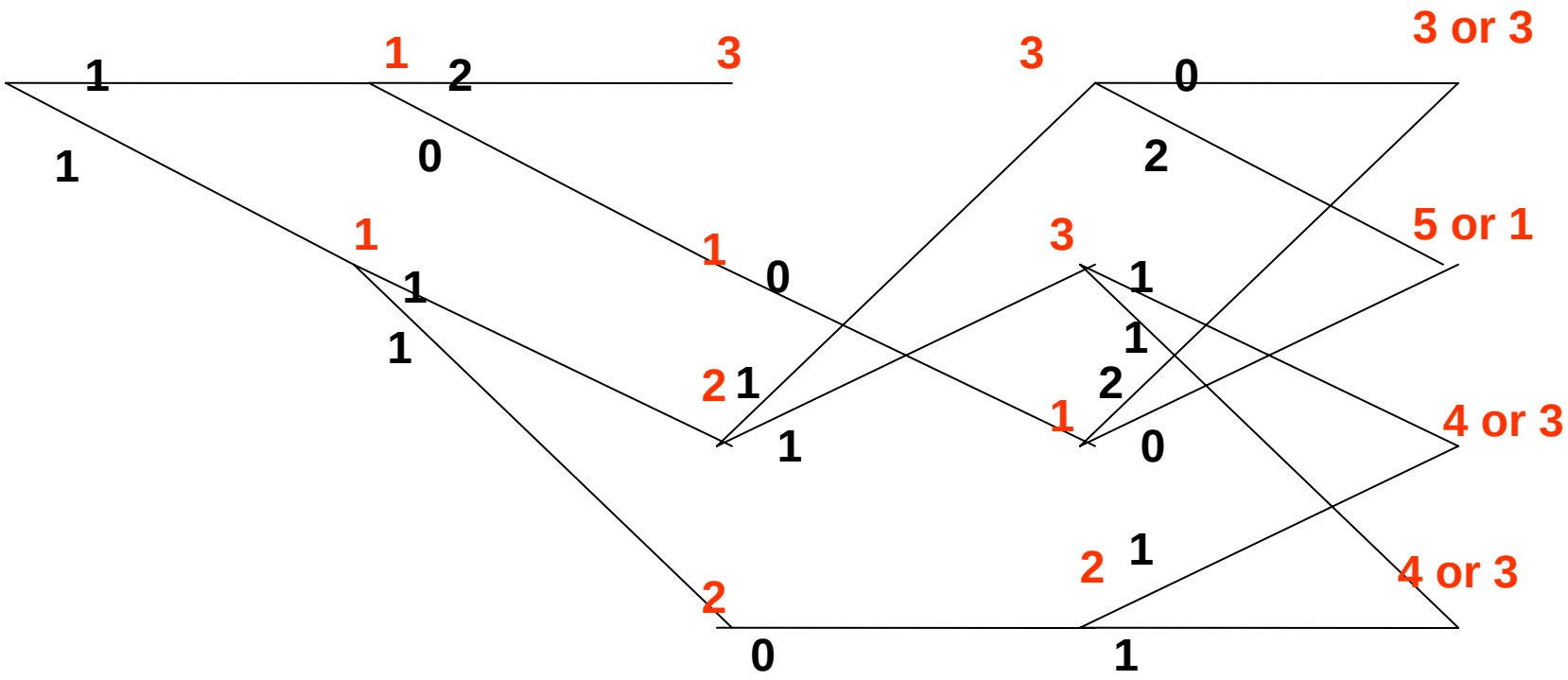
VITERBI DECODING

- **Viterbi decoding tells us to choose the smallest number at each state and eliminate the other path:**



VITERBI DECODING

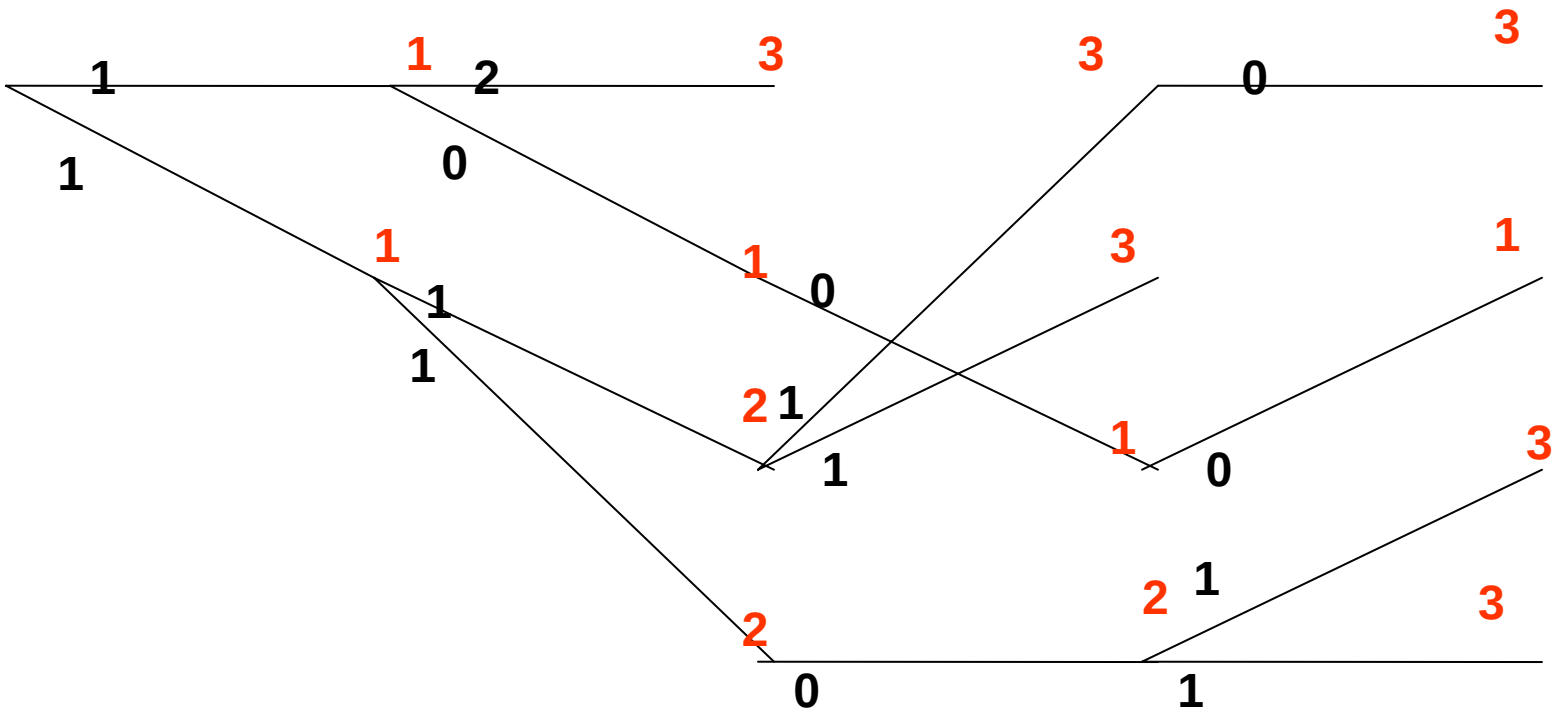
- **Continuing in this fashion we have:**



In case of a tie, take either path.

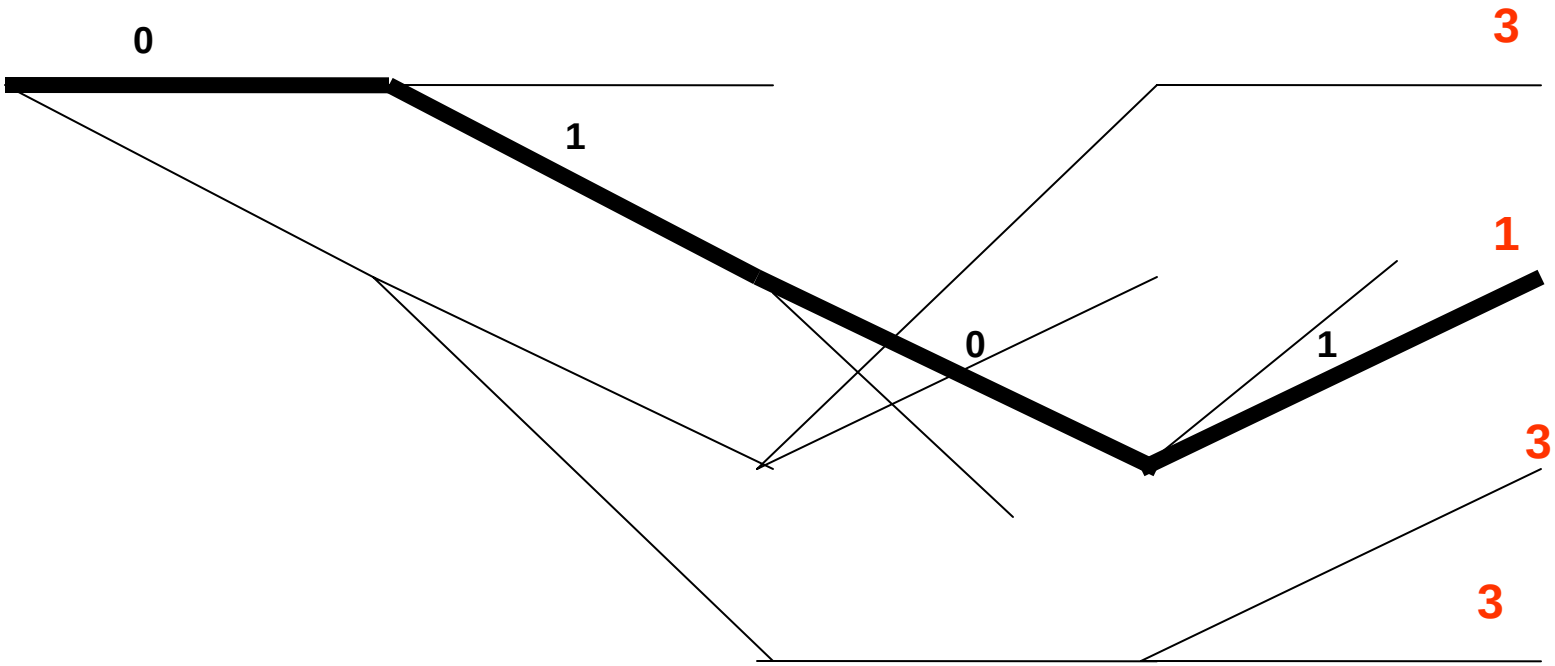
VITERBI DECODING

- **This yields:**



VITERBI DECODING

- **If a decision is to be made at this point we choose the path with the smallest sum. The information sequence (i.e., input to the encoder) 0 1 0 1 corresponds to that path.**



VITERBI DECODING

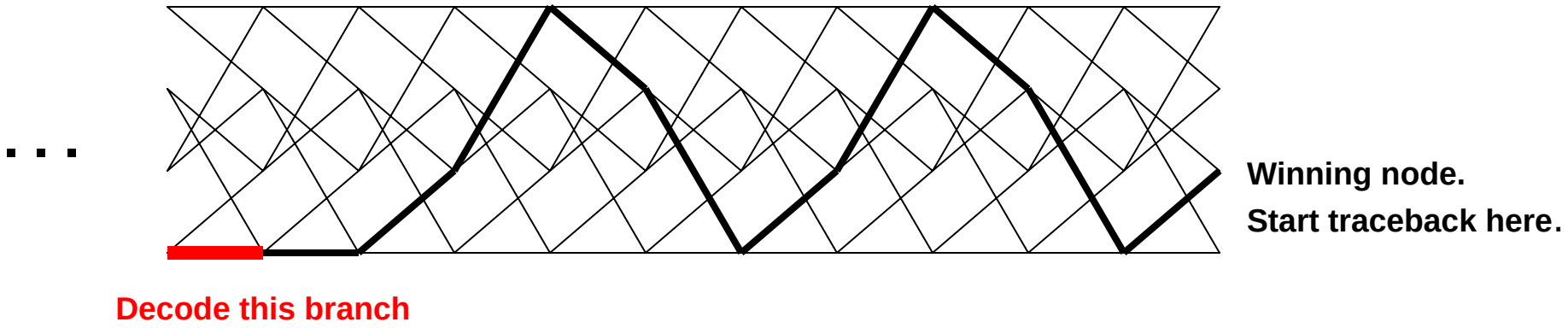
- **In a packet communication system, one could append 0's at the end of the message (i.e., the input to the encoder) to force the encoder back to the all 0 state.**

- **Then we would have only one path remaining and that would be the winning code word.**

VITERBI DECODING

- **The basic decoding module is referred to as an ACS module: Add, Compare, Select.**

- **At the decoder, one could make early decisions on the information bits by tracing back from the state with the lowest accumulated sum and then decoding some D steps back from the point of trace-back in the trellis.**



VITERBI DECODING

- **A great deal is known about how to choose D without losing much in performance.**

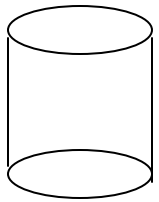
- **In a rate ½ code, a good value of D is about 4 to 5 times the number of delay elements in the decoder.**

VITERBI DECODING

- **The trace-back is complicated and may be the speed "bottleneck" in the decoder.**

- **One may not want to trace back at each step in the trellis but rather wait for L steps and then decode L branches at a time.**

- **Other tricks are used in designing the decoder. One is to use a circular buffer.**



DECODING ON THE TRELLIS: SOFT INPUT DECODING

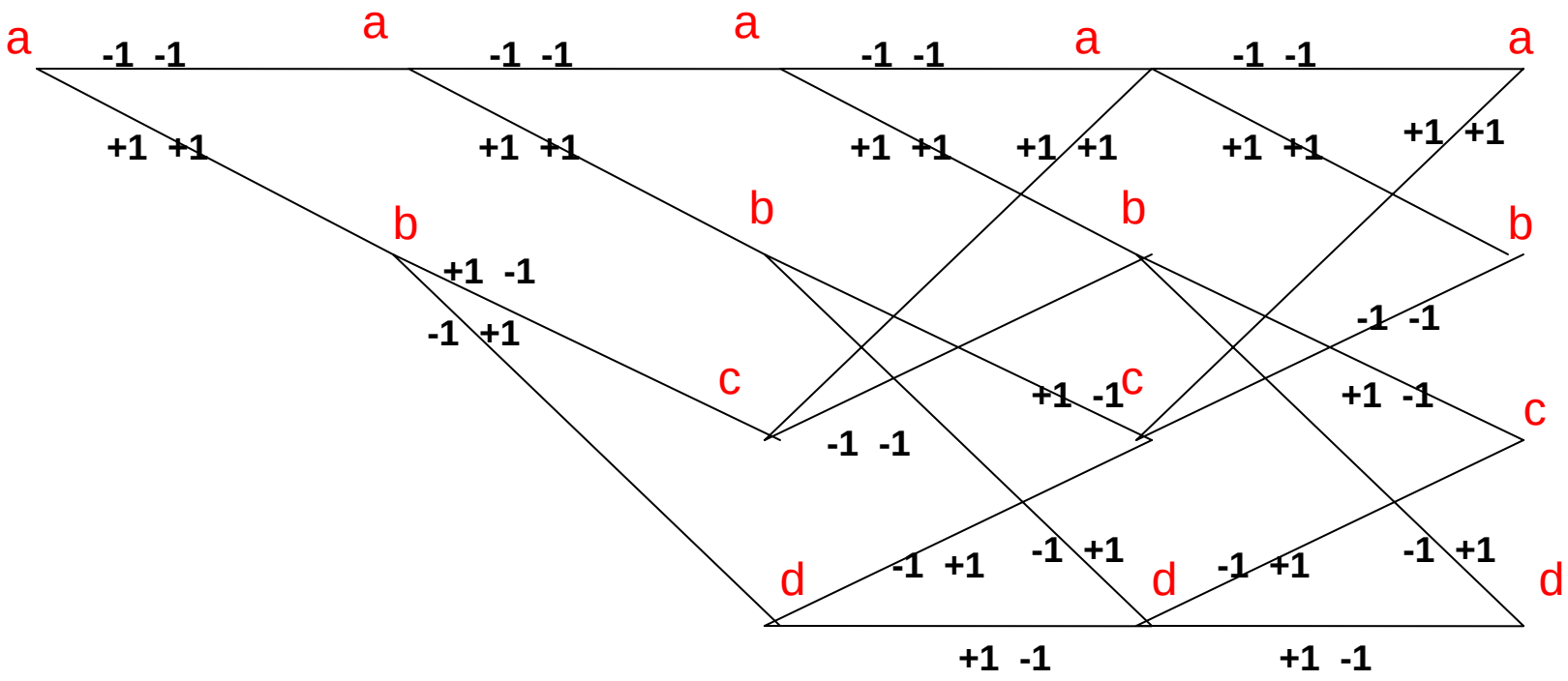
- **For a soft input decoder, the input to the decoders are probabilities of the code symbols.**

- **In actuality we use the logarithm of the probabilities so that for independent noise, the logarithm of the probability of a sequence is the sum of the logarithms of the individual symbols.**

- **FOR a AWGN channel, this corresponds to taking the squared difference between the noiseless transmitted outputs and the received values.**

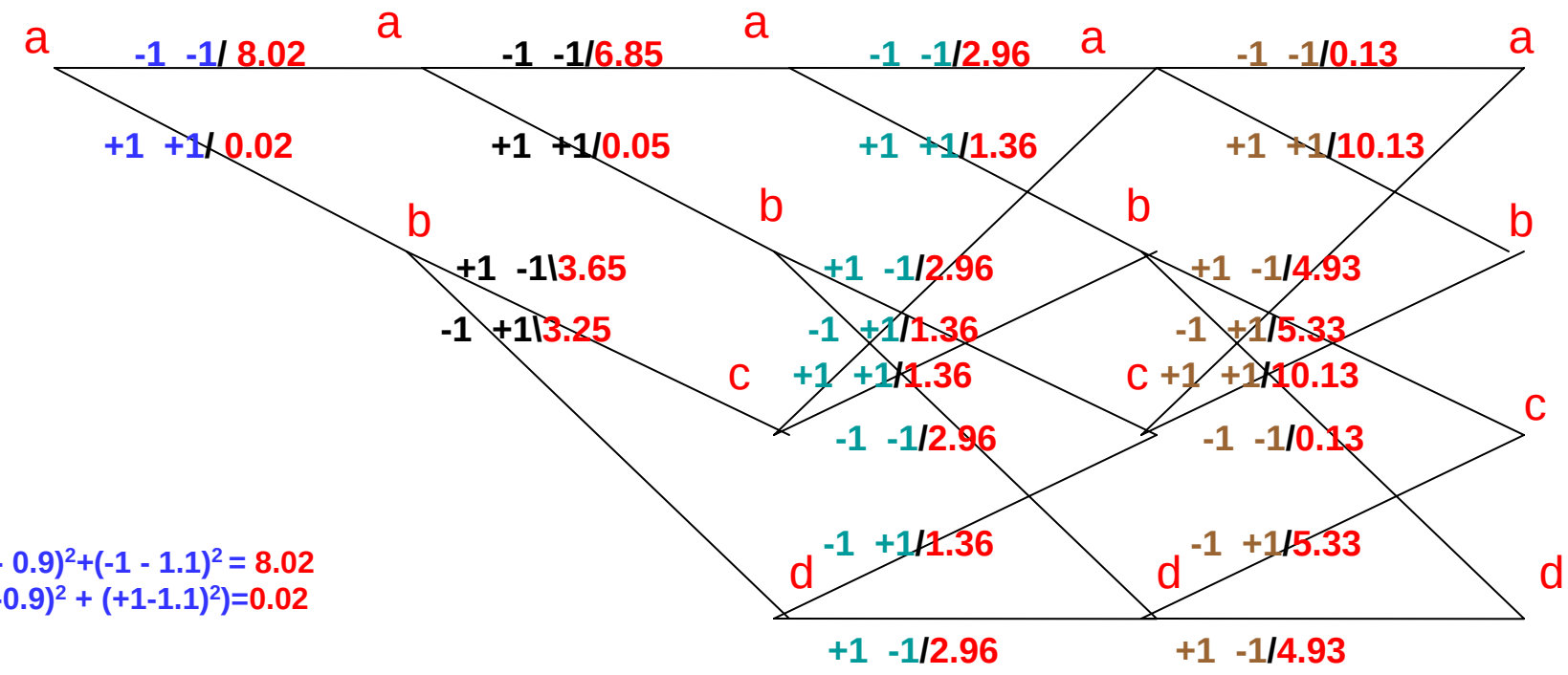
SOFT INPUT DECODING

- **Assume we transmit a code symbol 0 as a -1 and a code symbol 1 as a +1. Then we would label the branches in the trellis with the transmitted values:**



SOFT INPUT DECODING

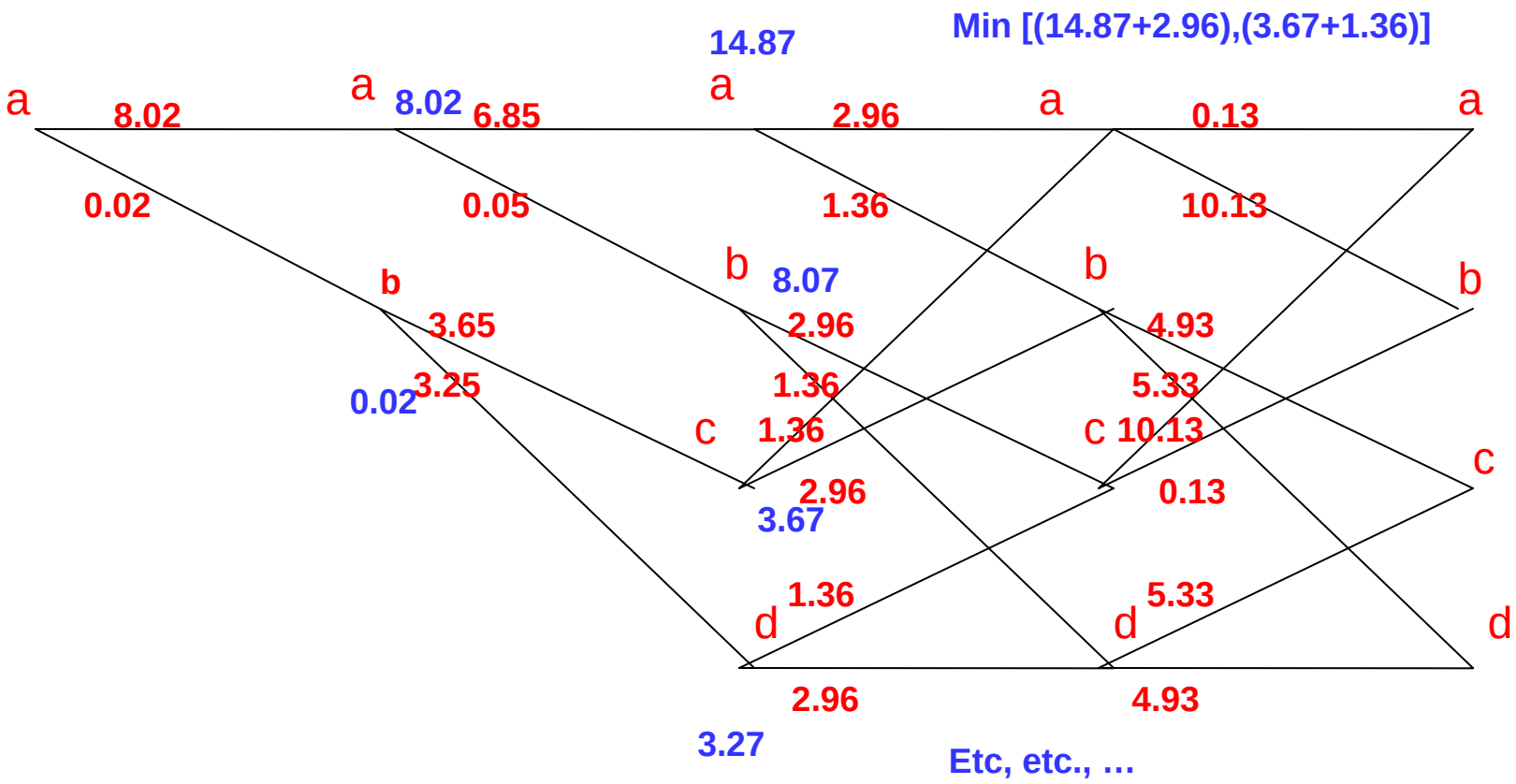
- **Now assume we receive the vector 0.9, 1.1, 0.8, 0.9, 0, 0.4, -1.2, -1.3, … We would put the following squared errors on the trellis:**



$(-1 - 0.9)^2+(-1 - 1.1)^2 = 8.02$
$(+1-0.9)^2 + (+1-1.1)^2)=0.02$

VITERBI ALGORITM WITH SOFT INPUT DECODING

- **Now use the Viterbi Algorithm.**



DECODING ON THE TRELLIS: SOFT OUTPUT DECODING

- **The Viterbi algorithm finds the path on the trellis that was most likely to have been transmitted. It makes hard decisions.**

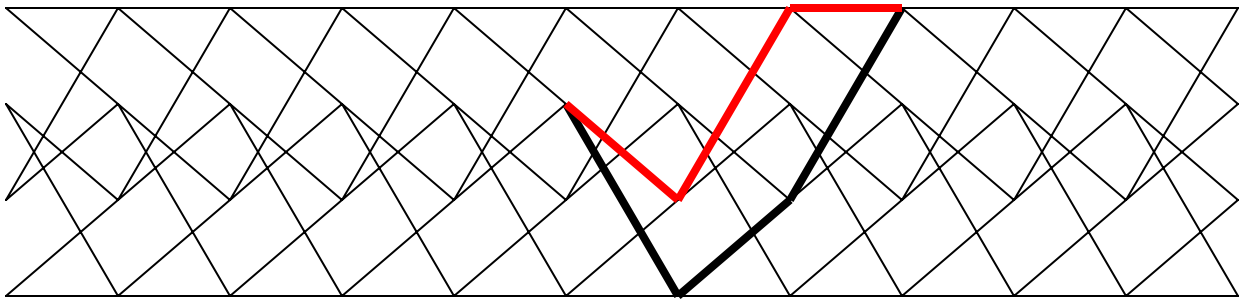
- **There is another (more complicated) algorithm called the BCJR algorithm that produces soft outputs: that is, it produces the probability that each symbol is a 0 or a 1.**

- **The output of the BCJR algorithm then can be used as the soft input to another decoder. This is used in iterative decoding for Turbo Codes.**

FREE HAMMING DISTANCE OF CONVOLUTIONAL CODES

- **The error correction capability of a convolutional code is governed by the so-called "free Hamming distance", d_{free}, of the code.**

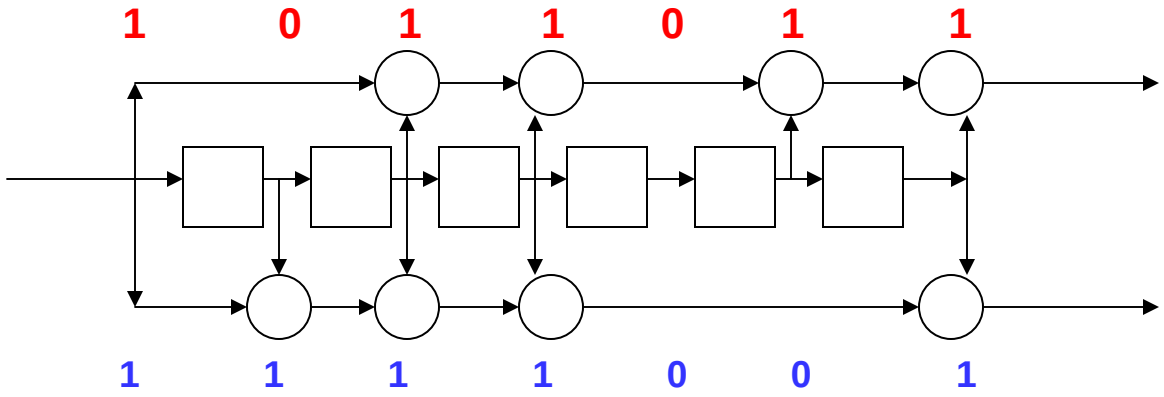
- **d_{free} is the smallest Hamming distance between two paths in the trellis that start in a common state and end in a common state. Two candidate paths are:**



FREE HAMMING DISTANCE OF CONVOLUTIONAL CODES

- **Tables giving the free distance of a large number of convolutional codes exist in many textbooks. They usually list the codes in octal form.**

- **One very popular rate ½ code with 64 states is usually listed as: 133, 171. It has d_{free}= 10. In binary, after dropping the leading 0's, this gives the tap connections: 1 0 1 1 0 1 1 and 1 1 1 1 0 0 1.**



PUNCTURING CONVOLUTIONAL CODES

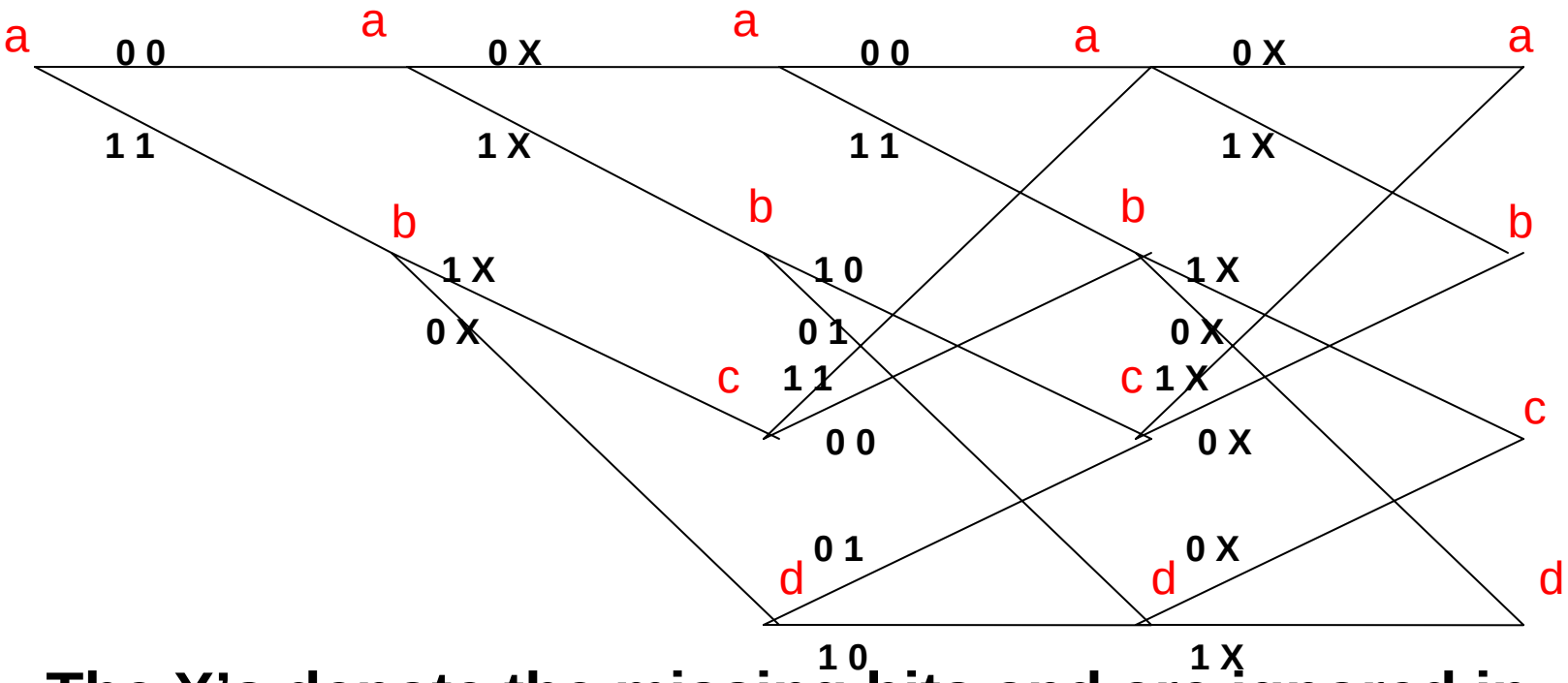
- **Starting with a rate ½ encoder one can construct higher rate codes by a technique called puncturing.**

- **For example, consider the following pattern:**
  - **Input the first binary digit and transmit both output bits**
  - **Input the next binary digit and transmit only one of the two output bits.**
  - **Input the next binary digit and transmit both output bits.**
  - **Input the next binary digit and transmit only one of the two output bits.**
  - **Continue in this way where the odd input binary digits produce two output bits but where the even inputs produce only one output bit.**

- **The result is a rate 2/3 code where we have 3 output binary digits for every two input binary digits.**

PUNCTURING CONVOLUTIONAL CODES

- **For example, in the 4-state trellis, if we omit the bottom output every other time, the result is:**



- **The X's denote the missing bits and are ignored in decoding.**

PUNCTURING CONVOLUTIONAL CODES

- **Note that the same trellis is used in decoding the original code and the punctured code.**

- **Puncturing lowers the d_{free} of the code. The best puncturing patterns were found by simulation.**

- **For example, for the rate ½, 64-state code we have:**

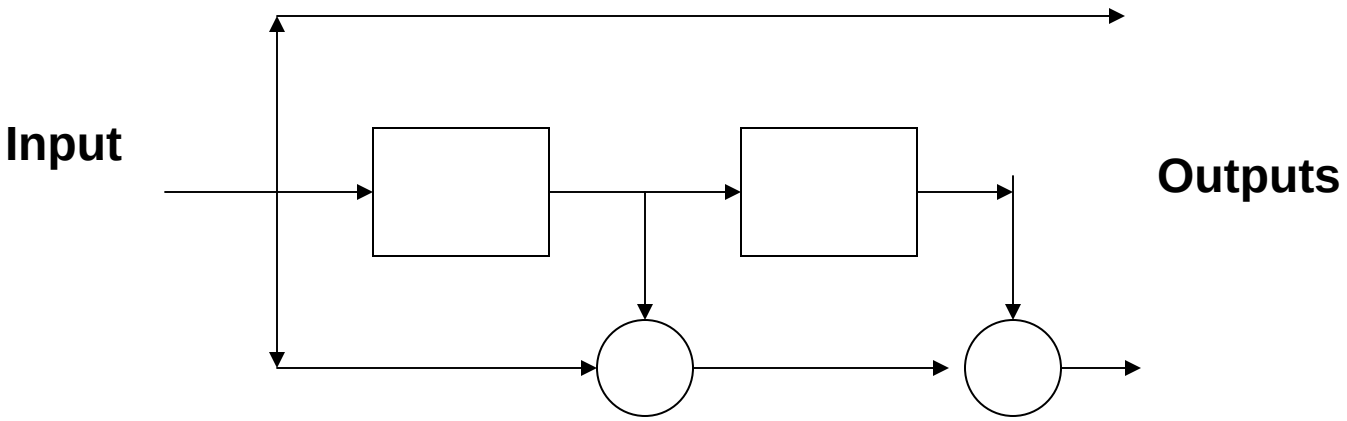
Rate	Puncturing Pattern	d_{free}
1/2	11, 11, 11, 11, 11, 11, 11 …	10
2/3	11, X1, 11, X1, 11, X1, 11 …	6
3/4	11, X1, 1X, 11, X1, 1X, 11 …	5
5/6	11, X1, 1X, X1, 1X, 11, X1 …	4
7/8	11, X1, X1, X1, 1X, X1, 1X …	3

TURBO CODES AND ITERATIVE DECODING

SYSTEMATIC CONVOLUTIONAL ENCODER

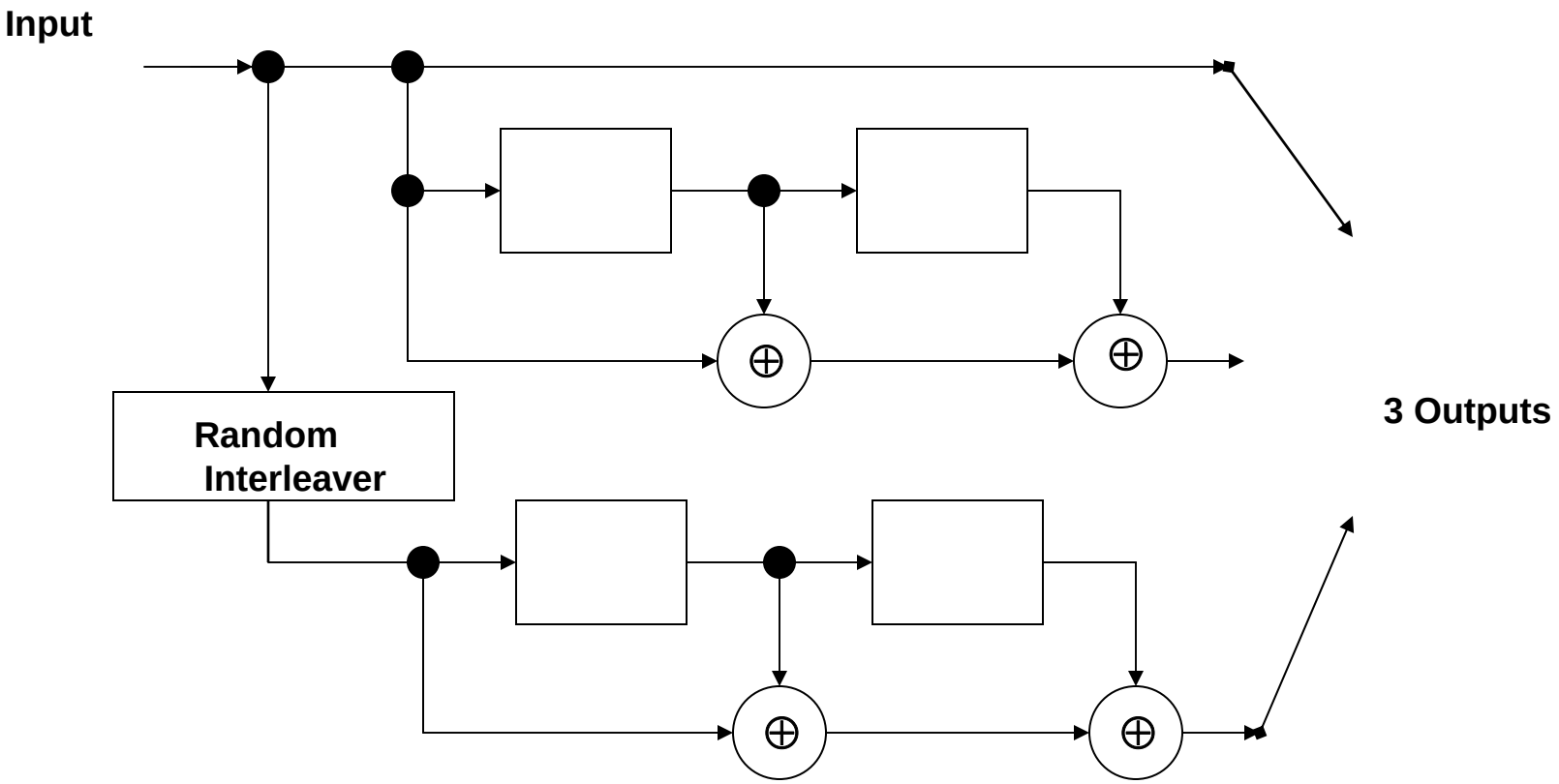
- **A systematic convolutional encoder has one of its outputs equal to its input.**

- **Example of a 4-state, rate ½, encoder:**

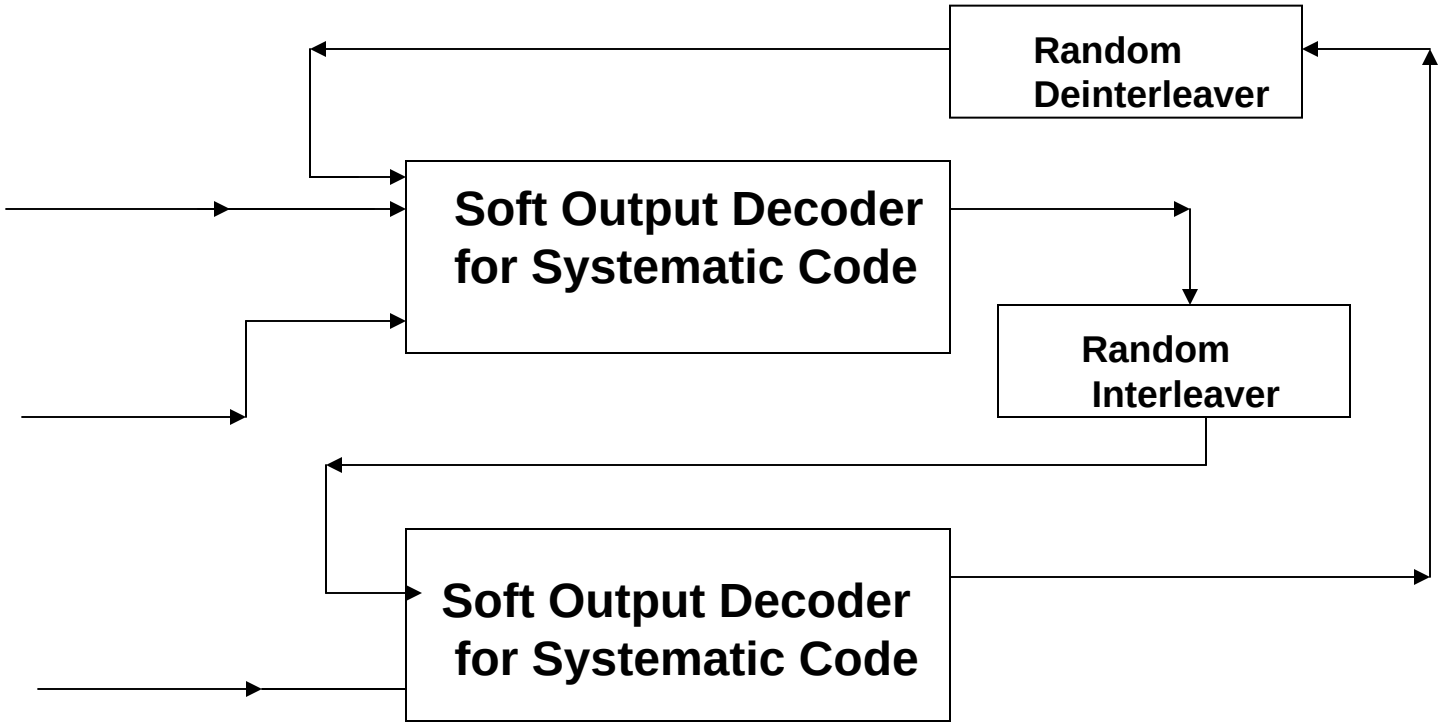


PARALLEL TURBO ENCODER

- **Example of a rate 1/3 Turbo encoder:**



PARALLEL TURBO DECODER



Many details are omitted

TURBO CODES

- **Codes with very few states give excellent performance if the interleaver size is large enough.**
- **Serial Turbo codes also exist and give comparable performance.**
- **Higher rate codes can be obtained by puncturing.**

AN INTRODUCTION TO ERROR CORRECTING CODES
Part 3

Jack Keil Wolf

ECE 154 C

Spring 2010

Introduction to LDPC Codes

- **These codes were invented by Gallager in his Ph.D. dissertation at M.I.T. in 1960.**
- **They were ignored for many years since they were thought to be impractical.**
- **But with present day technology they are very practical.**
- **Their performance is similar to turbo codes but they may have some implementation advantages.**

Outline: Some Questions

- What is a parity check code?
- What is an LDPC code?
- What is a message passing decoder for LDPC codes?
- What is the performance of these codes?

What is a Parity Check Code?

- **A binary parity check code is a block code: i.e., a collection of binary vectors of fixed length n.**

- **The symbols in the code satisfy r parity check equations of the form:**

$$x_a \oplus x_b \oplus x_c \oplus \ldots \oplus x_z = 0$$

where $\oplus$ means modulo 2 addition and

$$x_a, x_b, x_c, \ldots, x_z$$

are the code symbols in the equation.

- **Each codeword of length n can contain (n-r)=k information digits and r check digits.**

What is a Parity Check Matrix?

- **A parity check matrix is an r-row by n-column binary matrix. Remember k=n-r.**
- **The rows represent the equations and the columns represent the digits in the code word.**
- **There is a 1 in the i-th row and j-th column if and only if the i-th code digit is contained in the j-th equation.**

Example: Hamming Code with n=7, k=4, and r=3

- For a code word of the form $c_1, c_2, c_3, c_4, c_5, c_6, c_7$, the equations are:

$$c_1 \oplus c_2 \oplus c_3 \oplus c_5 = 0$$
$$c_1 \oplus c_2 \oplus c_4 \oplus c_6 = 0$$
$$c_1 \oplus c_3 \oplus c_4 \oplus c_7 = 0.$$

- The parity check matrix for this code is then:

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

- Note that c_1 is contained in all three equations while c_2 is contained in only the first two equations.

What is an LDPC Code?

- The percentage of 1's in the parity check matrix for a LDPC code is low.
- A regular LDPC code has the property that:
  - every code digit is contained in the same number of equations,
  - each equation contains the same number of code symbols.
- An irregular LDPC code relaxes these conditions.

The Equations for A Simple LDPC Code with n=12

$$c_3 \oplus c_6 \oplus c_7 \oplus c_8 = 0$$
$$c_1 \oplus c_2 \oplus c_5 \oplus c_{12} = 0$$
$$c_4 \oplus c_9 \oplus c_{10} \oplus c_{11} = 0$$
$$c_2 \oplus c_6 \oplus c_7 \oplus c_{10} = 0$$
$$c_1 \oplus c_3 \oplus c_8 \oplus c_{11} = 0$$
$$c_4 \oplus c_5 \oplus c_9 \oplus c_{12} = 0$$
$$c_1 \oplus c_4 \oplus c_5 \oplus c_7 = 0$$
$$c_6 \oplus c_8 \oplus c_{11} \oplus c_{12} = 0$$
$$c_2 \oplus c_3 \oplus c_9 \oplus c_{10} = 0.$$

- **There are actually only 7 independent equations so there are 7 parity digits**.

The Parity Check Matrix for the Simple LDPC Code

$c_1\ c_2\ c_3\ c_4\ c_5\ c_6\ c_7\ c_8\ c_9\ c_{10}\ c_{11}\ c_{12}$

0	0	1	0	0	1	1	1	0	0	0	0	$c_3 \oplus c_6 \oplus c_7 \oplus c_8 = 0$
1	1	0	0	1	0	0	0	0	0	0	1	$c_1 \oplus c_2 \oplus c_5 \oplus c_{12} = 0$
0	0	0	1	0	0	0	0	1	1	1	0	$c_4 \oplus c_9 \oplus c_{10} \oplus c_{11} = 0$
0	1	0	0	0	1	1	0	0	1	0	0	$c_2 \oplus c_6 \oplus c_7 \oplus c_{10} = 0$
1	0	1	0	0	0	0	1	0	0	1	0	$c_1 \oplus c_3 \oplus c_8 \oplus c_{11} = 0$
0	0	0	1	1	0	0	0	1	0	0	1	$c_4 \oplus c_5 \oplus c_9 \oplus c_{12} = 0$
1	0	0	1	1	0	1	0	0	0	0	0	$c_1 \oplus c_4 \oplus c_5 \oplus c_7 = 0$
0	0	0	0	0	1	0	1	0	0	1	1	$c_6 \oplus c_8 \oplus c_{11} \oplus c_{12} = 0$
0	1	1	0	0	0	0	0	1	1	0	0	$c_2 \oplus c_3 \oplus c_9 \oplus c_{10} = 0$

The Parity Check Matrix for the Simple LDPC Code

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

- **Note that each code symbol is contained in 3 equations and each equation involves 4 code symbols.**

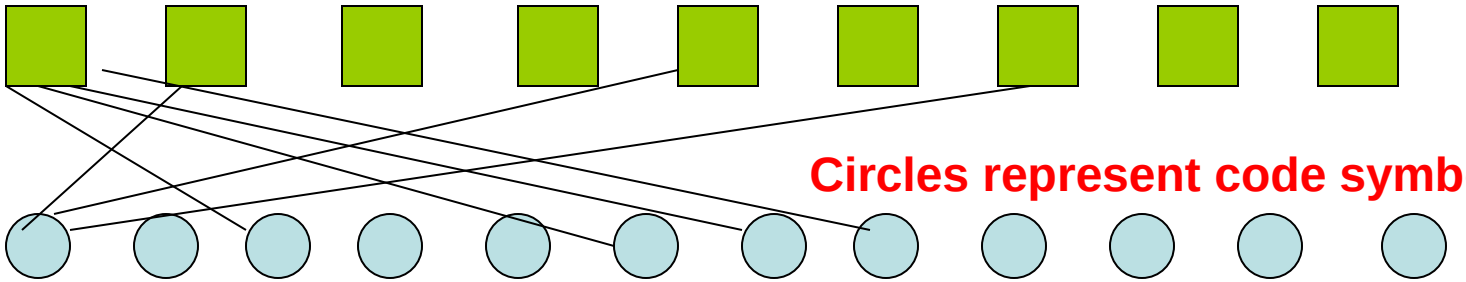
A Graphical Description of LDPC Codes

- **Decoding of LDPC codes is best understood by a graphical description.**
- **The graph has two types of nodes: bit nodes and parity nodes.**
- **Each bit node represents a code symbol and each parity node represents a parity equation.**
- **There is a line drawn between a bit node and a parity node if and only if that bit is involved in that parity equation.**

The Graph for the Simple LDPC Code

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

Squares represent parity equations.

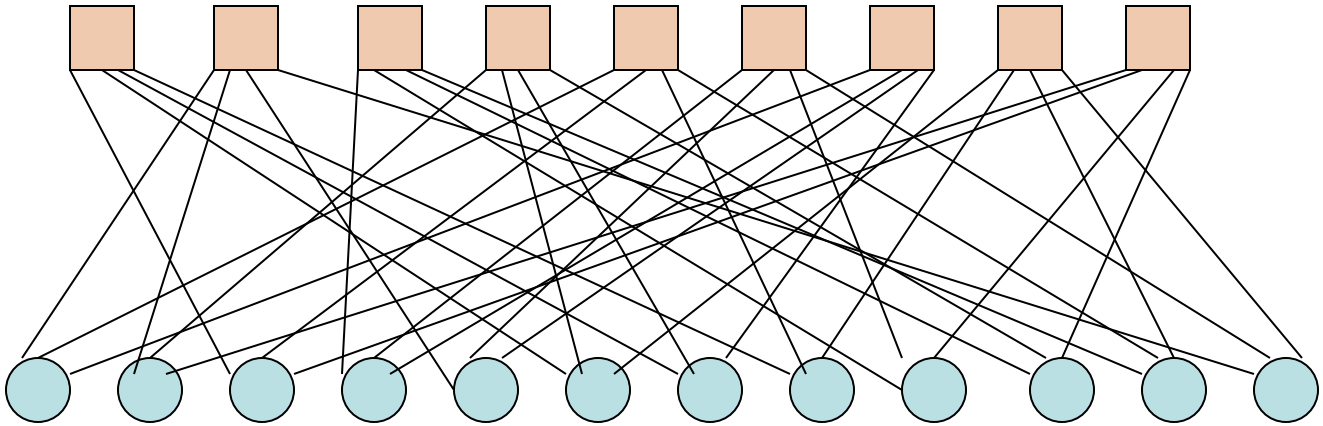


Circles represent code symbols.

Only the lines corresponding to the 1st row and 1st column are shown.

Entire Graph for the Simple LDPC Code

- **Note that each bit node has 3 lines connecting it to parity nodes and each parity node has 4 lines connecting it to bit nodes.**



Decoding of LDPC Codes by Message Passing on the Graph

- **Decoding is accomplished by passing messages along the lines of the graph.**

- **The messages on the lines that connect to the i-th bit node, c_i, are estimates of $Pr[c_i = 1]$ (or some equivalent information).**

- **At the nodes the various estimates are combined in a particular way.**

Decoding of LDPC Codes by Message Passing on the Graph

- **Each bit node is furnished an initial estimate of the probability it is a 1 from the soft output of the channel.**

- **The bit node broadcasts this initial estimate to the parity nodes on the lines connected to that bit node.**

- **But each parity node must make new estimates for the bits involved in that parity equation and send these new estimates (on the lines) back to the bit nodes.**

Estimation of Probabilities by Parity Nodes

- **Each parity node knows that there are an even number of 1's in the bits connected to that node.**
- **But the parity node has received estimates of the probability that each bit node connected to it is a 1.**
- **The parity node sends a new estimate to the i-th bit node based upon all the other probabilities furnished to it.**

Estimation of Probabilities by Parity Nodes

- For example, consider the parity node corresponding to the equation $c_3 \oplus c_6 \oplus c_7 \oplus c_8 = 0$.

- This parity node has the estimates p_3, p_6, p_7, and p_8 corresponding to the bit nodes c_3, c_6, c_7, and c_8, where p_i is an estimate for $Pr[c_i=1]$.

- The new estimate for the bit node c_3 is:

$$p'_3=p_6(1-p_7)(1-p_8)+ p_7(1-p_6)(1-p_8)+ p_8(1-p_6)(1-p_7)+ p_6p_7p_8$$

and for the other nodes:

$$p'_6=p_3(1-p_7)(1-p_8)+ p_7(1-p_3)(1-p_8)+ p_8(1-p_3)(1-p_7)+ p_3p_7p_8$$
$$p'_7=p_6(1-p_3)(1-p_8)+ p_3(1-p_6)(1-p_8)+ p_8(1-p_3)(1-p_6)+ p_3p_6p_8$$
$$p'_8=p_6(1-p_7)(1-p_3)+ p_7(1-p_6)(1-p_3)+ p_3(1-p_6)(1-p_7)+ p_3p_6p_7$$

Estimation of Probabilities by Bit Nodes

- But the bit nodes are provided different estimates of Pr[c=1] by the channel and by each of the parity nodes connected to it.
- It no longer broadcasts a single estimate but sends different estimates to each parity equation.
- The new estimate sent to each parity node is obtained by combining all other current estimates.
- That is, in determining the new estimate sent to a parity node, it ignores the estimate received from that parity node.

Estimation of Probabilities by Bit Nodes

- **The new estimate sent to each parity node is equal to the normalized product of the other estimates.**
- **The proper normalization is a detail which will be discussed later.**
- **If instead of passing estimates of Pr[c=1] we pass estimates of log {Pr[c=1]/Pr[c=0]} where Pr[c=0] = 1 - Pr[c=1], we merely need to add the appropriate terms.**
- **The channel estimate is always used in all estimates passed to the parity node.**

Estimation of Probabilities by Bit Nodes

- **The following table illustrates how estimates are combined by a bit node involved in 3 parity equations A, B, and C.**

Estimate received from channel:	p_{ch}
Estimate received from parity node A:	p_A
Estimate received from parity node B:	p_B
Estimate received from parity node C:	p_C
New estimate sent to parity node A:	$K\, p_{ch}\, p_B\, p_C$
New estimate sent to parity node B:	$K\, p_{ch} p_A\, p_C$
New estimate sent to parity node C:	$K\, p_{ch} p_A\, p_B$

The Rest of the Decoding Algorithm

- **The process now repeats: parity nodes passing messages to bit nodes and bit nodes passing messages to parity nodes.**

- **At the last step, a final estimate is computed at each bit node by computing the normalized product of all of its estimates.**

- **Then a hard decision is made on each bit by comparing the final estimate with the threshold 0.5.**

Final Estimate Made by Bit Nodes

- **The following table illustrates how the final estimate is made by a bit node involved in 3 parity equations A, B, and C.**

Estimate received from channel:	p_{ch}
Estimate received from parity node A:	p_A
Estimate received from parity node B:	p_B
Estimate received from parity node C:	p_C
FINAL ESTIMATE:	$K\, p_{ch} p_A\, p_B\, p_C$

Decoding of Simple Example

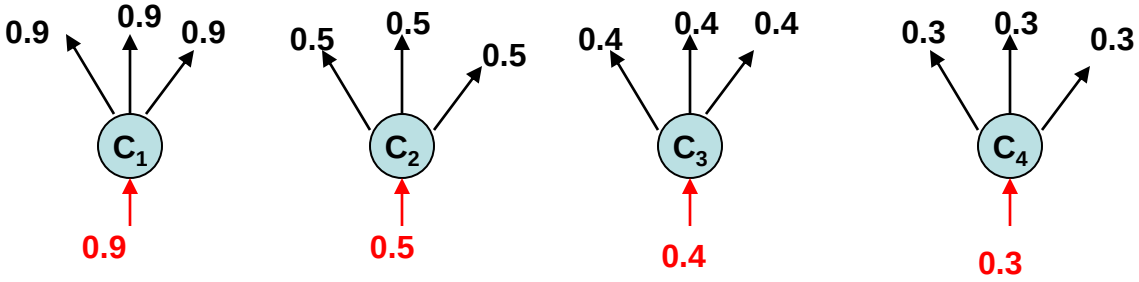
- **Suppose the following Pr[C_i=1], i=1, 2, …, 12 are obtained from channel:**

  0.9 0.5 0.4 0.3 0.9 0.9 0.9 0.9 0.9 0.9 0.9 0.9

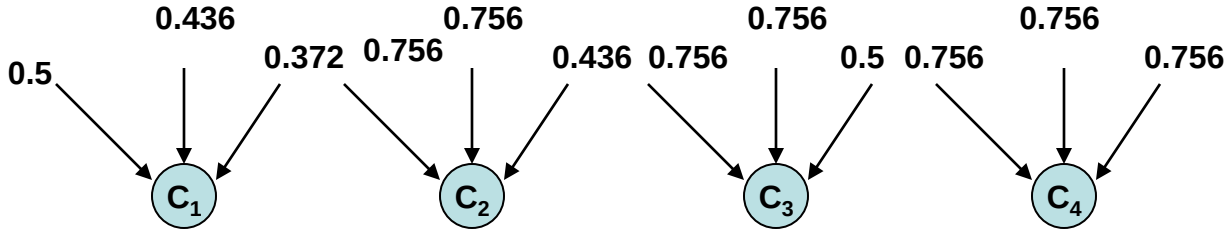
- **We now watch the decoder decode.**

Decoding of Simple Example: First 4 Bit Nodes Only

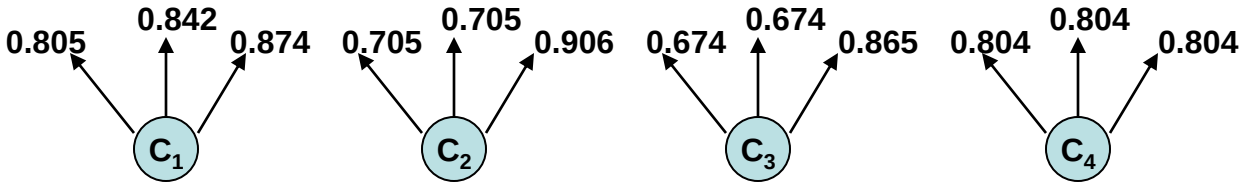
- **Initial broadcast from first 4 bit nodes:**



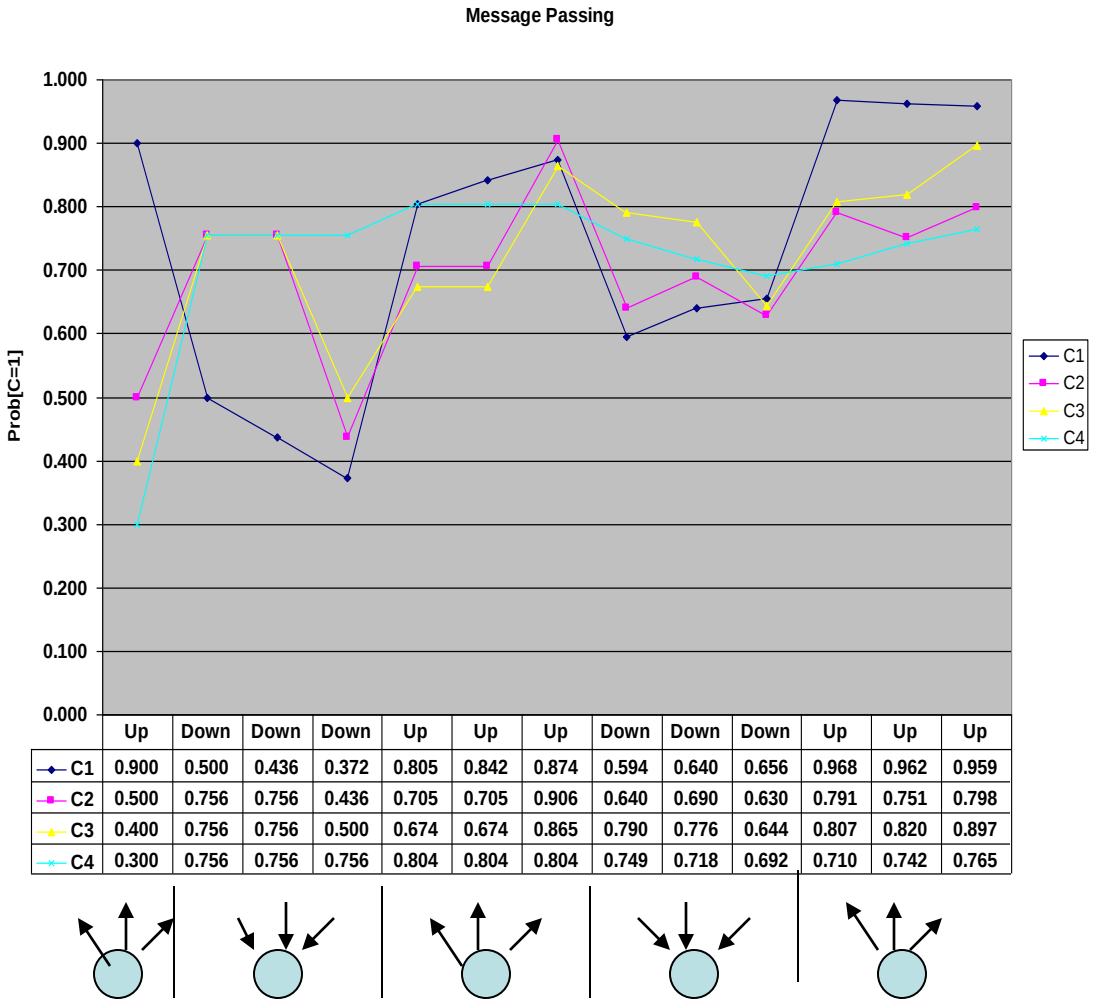
- **Transmission from parity nodes to these 4 bit nodes:**



- **Next transmission from the first 4 bit nodes:**



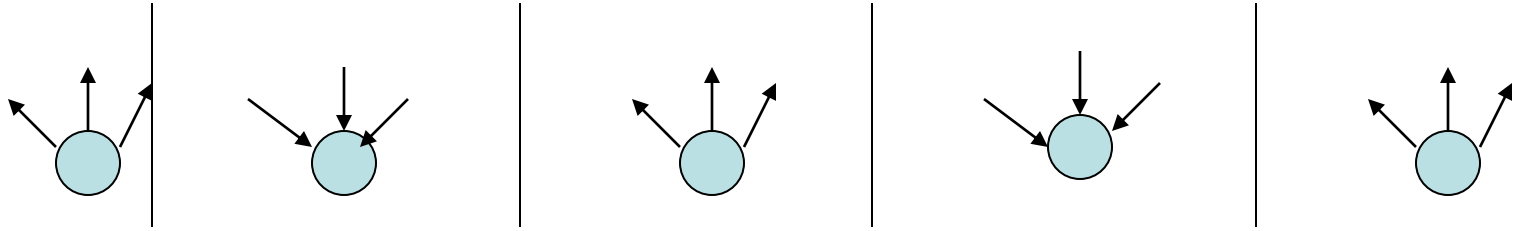
Message Passing for First 4 Bit Nodes for More Iterations



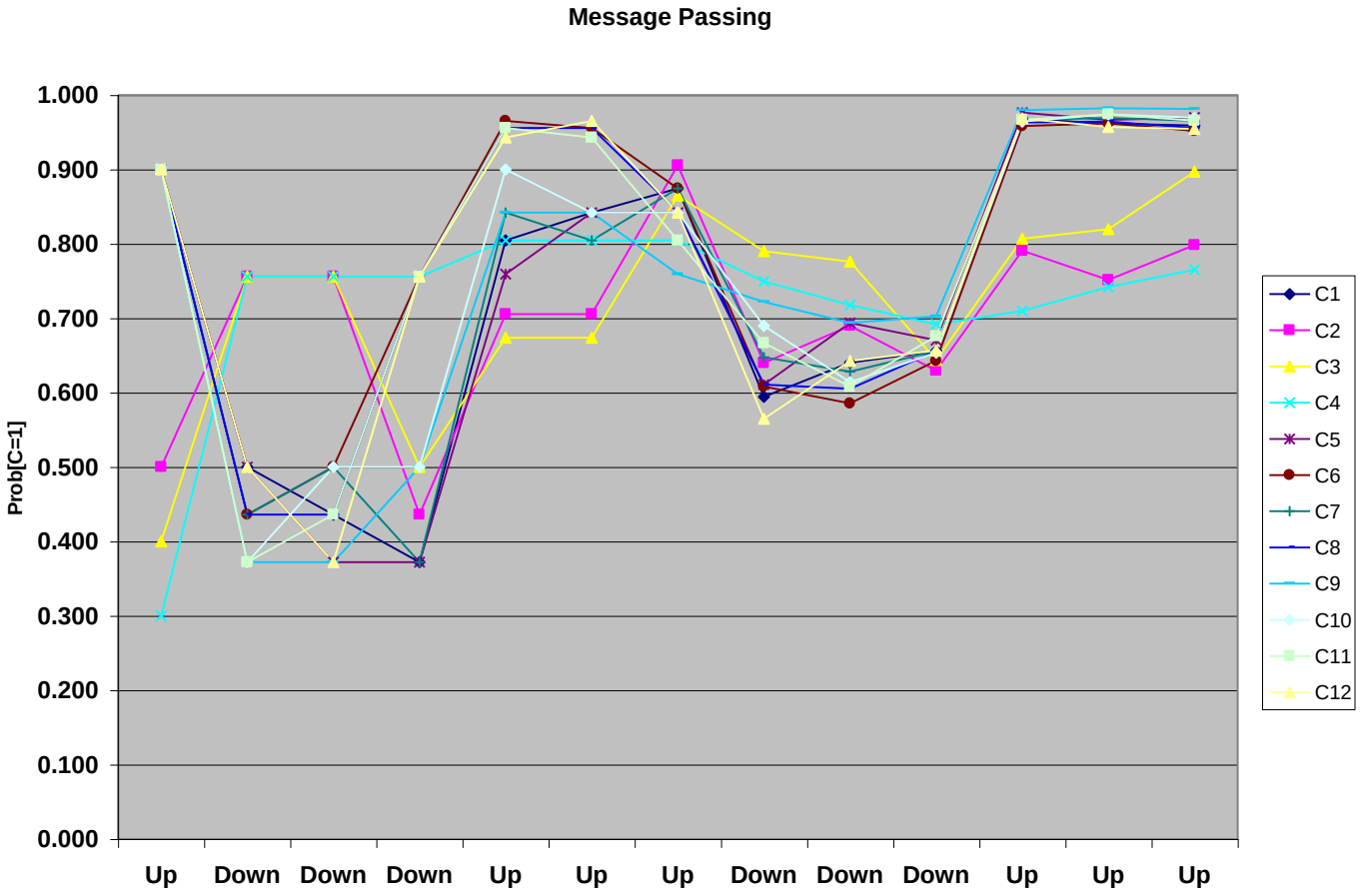
	Up	Down	Down	Down	Up	Up	Up	Down	Down	Down	Up	Up	Up
C1	0.900	0.500	0.436	0.372	0.805	0.842	0.874	0.594	0.640	0.656	0.968	0.962	0.959
C2	0.500	0.756	0.756	0.436	0.705	0.705	0.906	0.640	0.690	0.630	0.791	0.751	0.798
C3	0.400	0.756	0.756	0.500	0.674	0.674	0.865	0.790	0.776	0.644	0.807	0.820	0.897
C4	0.300	0.756	0.756	0.756	0.804	0.804	0.804	0.749	0.718	0.692	0.710	0.742	0.765

Messages Passed To and From All 12 Bit Nodes

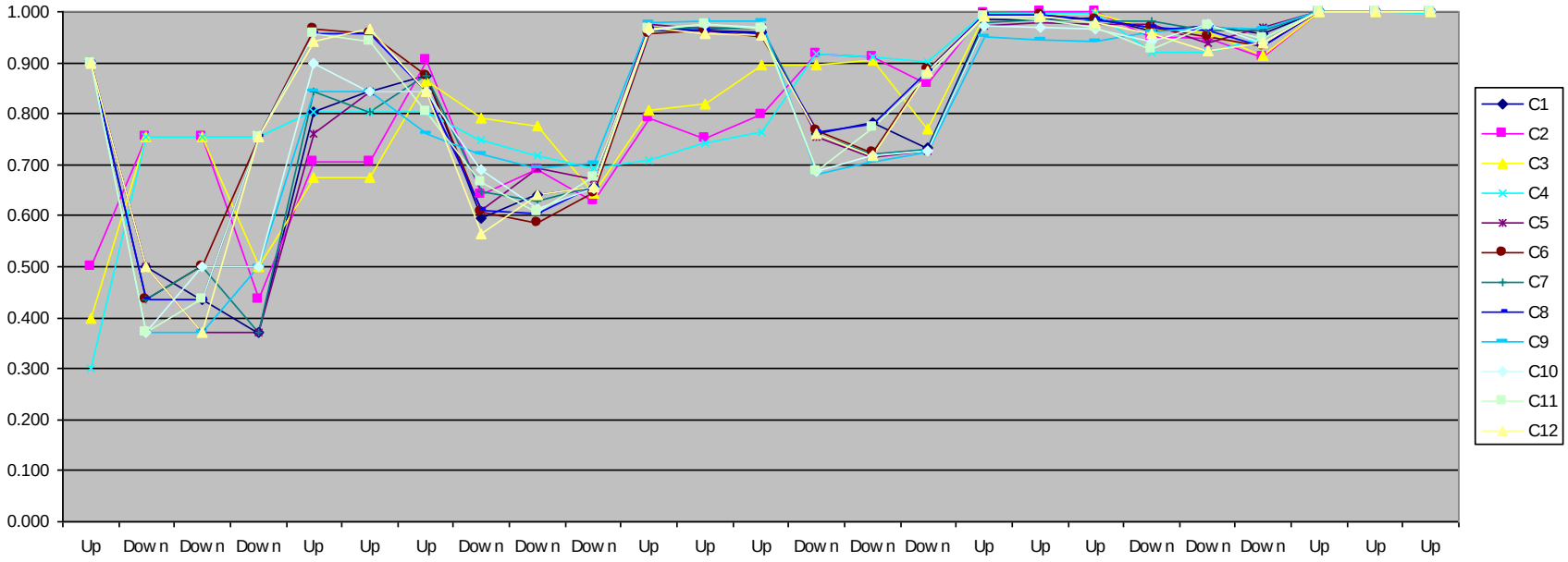
	Up	Down	Down	Down	Up	Up	Up	Down	Down	Down	Up	Up	Up
C1	0.900	0.500	0.436	0.372	0.805	0.842	0.874	0.594	0.640	0.656	0.968	0.962	0.959
C2	0.500	0.756	0.756	0.436	0.705	0.705	0.906	0.640	0.690	0.630	0.791	0.751	0.798
C3	0.400	0.756	0.756	0.500	0.674	0.674	0.865	0.790	0.776	0.644	0.807	0.820	0.897
C4	0.300	0.756	0.756	0.756	0.804	0.804	0.804	0.749	0.718	0.692	0.710	0.742	0.765
C5	0.900	0.500	0.372	0.372	0.759	0.842	0.842	0.611	0.694	0.671	0.976	0.966	0.970
C6	0.900	0.436	0.500	0.756	0.965	0.956	0.874	0.608	0.586	0.643	0.958	0.962	0.952
C7	0.900	0.436	0.500	0.372	0.842	0.805	0.874	0.647	0.628	0.656	0.967	0.969	0.965
C8	0.900	0.436	0.436	0.756	0.956	0.956	0.843	0.611	0.605	0.656	0.963	0.964	0.956
C9	0.900	0.372	0.372	0.500	0.842	0.842	0.759	0.722	0.694	0.703	0.980	0.982	0.981
C10	0.900	0.372	0.500	0.500	0.900	0.842	0.842	0.690	0.614	0.654	0.964	0.974	0.970
C11	0.900	0.372	0.436	0.756	0.956	0.943	0.805	0.667	0.608	0.676	0.967	0.974	0.965
C12	0.900	0.500	0.372	0.756	0.943	0.965	0.842	0.565	0.642	0.657	0.969	0.957	0.955



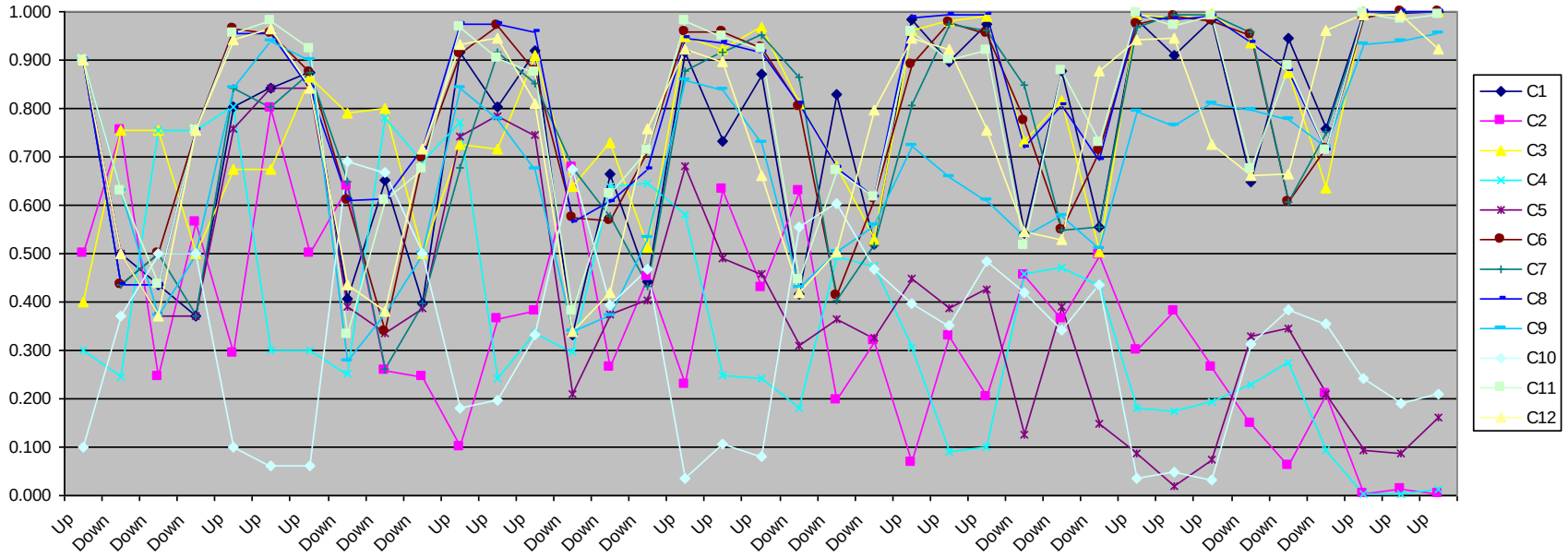
Messages Passed To and From All 12 Bit Nodes



More Iterations
All 12 Bit Nodes



More Interesting Example
All 12 Bit Nodes



Computation at Bit Nodes

- **If estimates of probabilities are statistically independent you should multiply them.**

- **But you need to normalize the product. Otherwise the product is smaller than every single estimate.**

- **For example, with three independent estimates all equal to 0.9, the unnormalized product is:**

$$(0.9)^3 = 0.729$$

where the correct normalized product is:

$$(0.9)^3 \; / \; [(0.1)^3 + (0.9)^3] = 0.9986$$

Derivation of Correct Normalization

- Assume we have 3 independent estimates, p_a, p_b, and p_c from which we compute the new estimate p' from the formula:

$$p' = K\, p_a\, p_b\, p_c.$$

- But the same normalization must hold for (1-p'):

$$(1-p') = K(1- p_a)(1- p_b)(1- p_c)$$

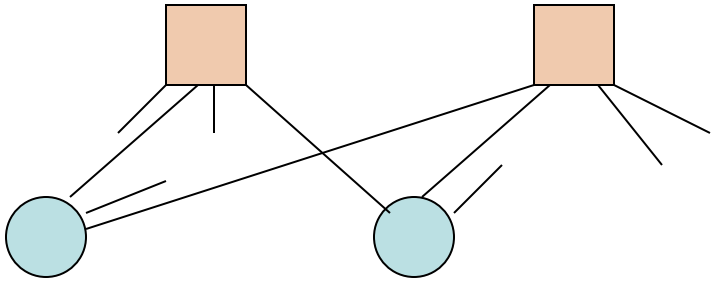
- From the first equation $(1-p') = 1- K\, p_a\, p_b\, p_c$.

- Setting $(1- K\, p_a\, p_b\, p_c)$ equal to $K(1- p_a)(1- p_b)(1- p_c)$ and solving for K we obtain:

$$K = 1 / [(1- p_a)(1- p_b)(1- p_c) + p_a\, p_b\, p_c]$$

Assumption of Independence

- **Note that in our example, parts of the graph looks like:**



- **This is called a cycle of length 4.**
- **Cycles cause estimates to be dependent and our combining formulas are incorrect.**
- **As a result short cycles should be avoided in the design of codes.**

Computation at Parity Nodes

- **When a parity equation involves many bits, an alternative formula is used.**

- **Details are omitted here but can be found in the literature.**

Rate of a Regular LDPC Code

- **Assume a LDPC is designed where:**

  (1) every bit is in J parity checks, and
  (2) every parity check checks K bits.

- **Since the number of 1's in a parity check matrix is the same whether we count by rows or columns, we have**

  J (# of columns) = K (# of rows)
  or J (n) = K (n-k).

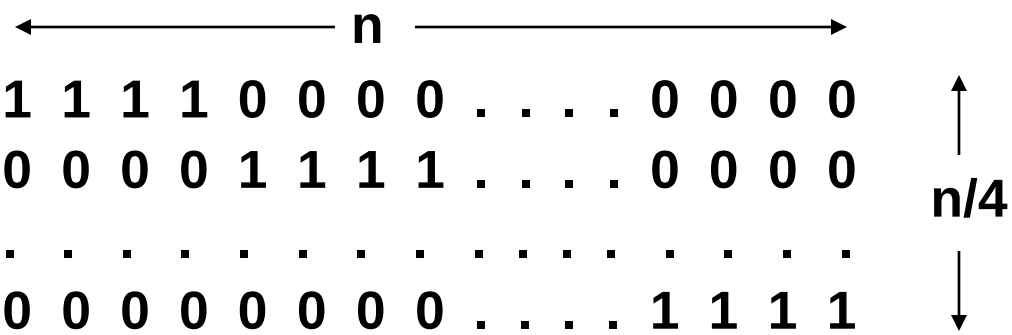
- **Solving for k / n, we have k/n = (1- J / K), the rate of the code.**

- **Higher rate codes can be obtained by puncturing lower rate codes.**

Design of a Parity Matrix for a Regular LDPC Code

- **The following procedure was suggested by Gallager. We illustrate it for a code with J = 3 and K = 4 .**

1. Construct the first n/4 rows as follows:



2. Construct the next n/4 rows by permuting the columns of the first n/4 rows.

3. Repeat 2 using another permutation of the columns.

Irregular LDPC Codes

- **Irregular LDPC codes have a variable number of 1's in the rows and in the columns.**
- **The optimal distributions for the rows and the columns are found by a technique called density evolution.**
- **Irregular LDPC codes perform better than regular LDPC codes.**
- **The basic idea is to give greater protection to some digits and to have some of the parity equations give more reliable information to give the decoding a jump start .**

Paper on Irregular LDPC Codes Luby et al (ISIT 1998)

ISIT 1998, Cambridge, MA, USA, August 16 – August 21

Improved Low-Density Parity-Check Codes Using Irregular Graphs and Belief Propagation

Michael G. Luby[1]
International Computer Science Institute
Berkeley, CA
Email luby@icsi.berkeley.edu

Michael Mitzenmacher
Digital Systems Research Center
Palo Alto, CA
Email michaelm@pa.dec.com

M. Amin Shokrollahi
International Computer Science Institute
Berkeley, CA
Email amin@icsi.berkeley.edu

Daniel A. Spielman
Department of Mathematics, M.I.T.
Cambridge, MA
Email spielman@math.mit.edu

Abstract — **We construct new families of low-density parity-check codes, which we call *irregular codes*. When decoded using belief propagation, our codes can correct more errors than previously known low-density codes. Our improved performance comes from using codes based on irregular random bipartite graphs, based on the work of [2]. Previously studied low-density codes have been derived from regular bipartite graphs. Initial experimental results for our irregular codes suggest that, with improvements, irregular codes may be able to match turbo code performance.**

I. Introduction

We have constructed new families of low-density codes, which we call *irregular codes*. The terminology comes from the fact that the parity-check matrices of our codes yield highly irregular bipartite graphs. These codes have significantly improved performance over previously known codes of this type. Using belief propagation, they can correct a substantially higher number of errors, albeit at the expense of a slightly slower running time. Further information can be found in an extended version of this paper, available as TR-97-044 of the International Computer Science Institute in Berkeley.

II. Irregular Graphs: Intuition and Example

We offer some intuition as to why using irregular graphs should improve performance. Consider trying to build a regular low-density code that transmits at a fixed rate. From the point of view of a message node, it is best to have high degree, since the more information it gets from its check nodes, the more accurately it can judge what its correct value should be. In contrast, from the point of view of a check node, it is best to have low degree, since the lower the degree of a check node, the more valuable the information it can transmit to its neighbors. These two competing requirements must be appropriately balanced. Previous work has shown that for regular graphs, low degree graphs yield the best performance [1]. If one allows irregular graphs, however, there is significantly more flexibility in balancing these competing requirements. Message nodes with high degree will tend to their correct value quickly. These nodes then provide good information to the check nodes, which subsequently provide better information to lower degree message nodes. Irregular graph constructions thus lead to a wave effect, where high degree message nodes tend to get corrected first, and then message nodes with slightly smaller degree, and so on down the line.

[1] Parts of this research were done while at Digital Systems Research Center. Research partially supported by NSF operating grant NCR-9416101.

Code Rate 1/2	
Left Degrees	$\lambda_3 = 0.44506$, $\lambda_4 = 0.26704$, $\lambda_6 = 0.14835$, $\lambda_{17} = 0.07854$, $\lambda_{33} = 0.04046$, $\lambda_{64} = 0.02055$
Right Degrees	$\rho_7 = 0.35282$, $\rho_8 = 0.29548$, $\rho_{10} = 0.10225$, $\rho_{20} = 0.18321$, $\rho_{64} = 0.04179$, $\rho_{65} = 0.02445$

Table 1: Sample code parameters.

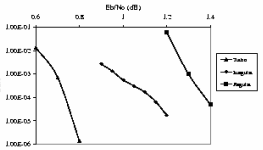


Figure 1: From left to right, rate 1/2 turbo codes, irregular codes, and regular codes.

This intuition (which we observe in our experiments) unfortunately does not provide clues as to how to construct appropriate irregular graphs. Moreover, because belief propagation is not yet well understood mathematically, creating the proper irregular graphs appears a daunting challenge. We meet this challenge by using irregular graphs that have been proven to be effective for erasure codes that function in a similar manner [2]. In the area of erasure codes, the mathematical framework has been established to both design irregular graphs and prove their effectiveness.

We provide an example of the irregular graphs used in Table 1. In the table, λ_i (ρ_i) denotes the fraction of nodes of degree i on the left (right) hand side of the graph. Note that given a vector λ and ρ one can construct a graph with (approximately) the correct node fractions for any number of nodes.

References

[1] D. J. C. MacKay and R. M. Neal, "Good Error Correcting Codes Based on Very Sparse Matrices," available from http://wol.ra.phy.cam.ac.uk/mackay.

[2] M. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical Loss-Resilient Codes," *Proc. 29th Symp. on Theory of Computing*, 1997, pp. 150–159.

Paper on Irregular LDPC Codes Luby et al (ISIT 1998)

- Code Rate ½

- Left degrees

  Λ_3 =.44506 λ_5 =. 26704
  Λ_9 =.14835 λ_{17}=. 07854
  λ_{33}=.04046 λ_{65}=. 02055

- Right degrees

  ρ_7 =.38282 ρ_8 =. 29548
  ρ_{19}=.10225 ρ_{20}=. 18321
  ρ_{84}=.04179 ρ_{85}=. 02445

Code Rate 1/2	
Left Degrees	$\lambda_3 = 0.44506$, $\lambda_5 = 0.26704$,
	$\lambda_9 = 0.14835$, $\lambda_{17} = 0.07854$,
	$\lambda_{33} = 0.04046$, $\lambda_{65} = 0.02055$
Right Degrees	$\rho_7 = 0.35282$, $\rho_8 = 0.29548$,
	$\rho_{19} = 0.10225$, $\rho_{20} = 0.18321$,
	$\rho_{84} = 0.04179$, $\rho_{85} = 0.02445$

Table 1: Sample code parameters.

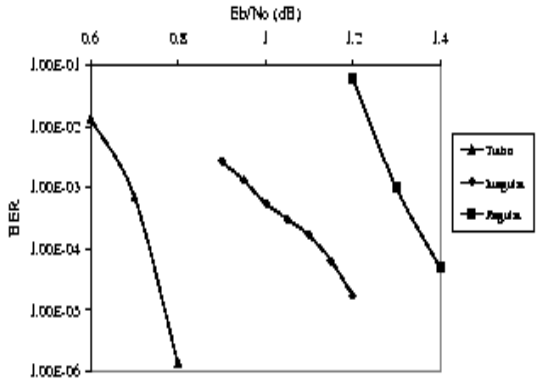
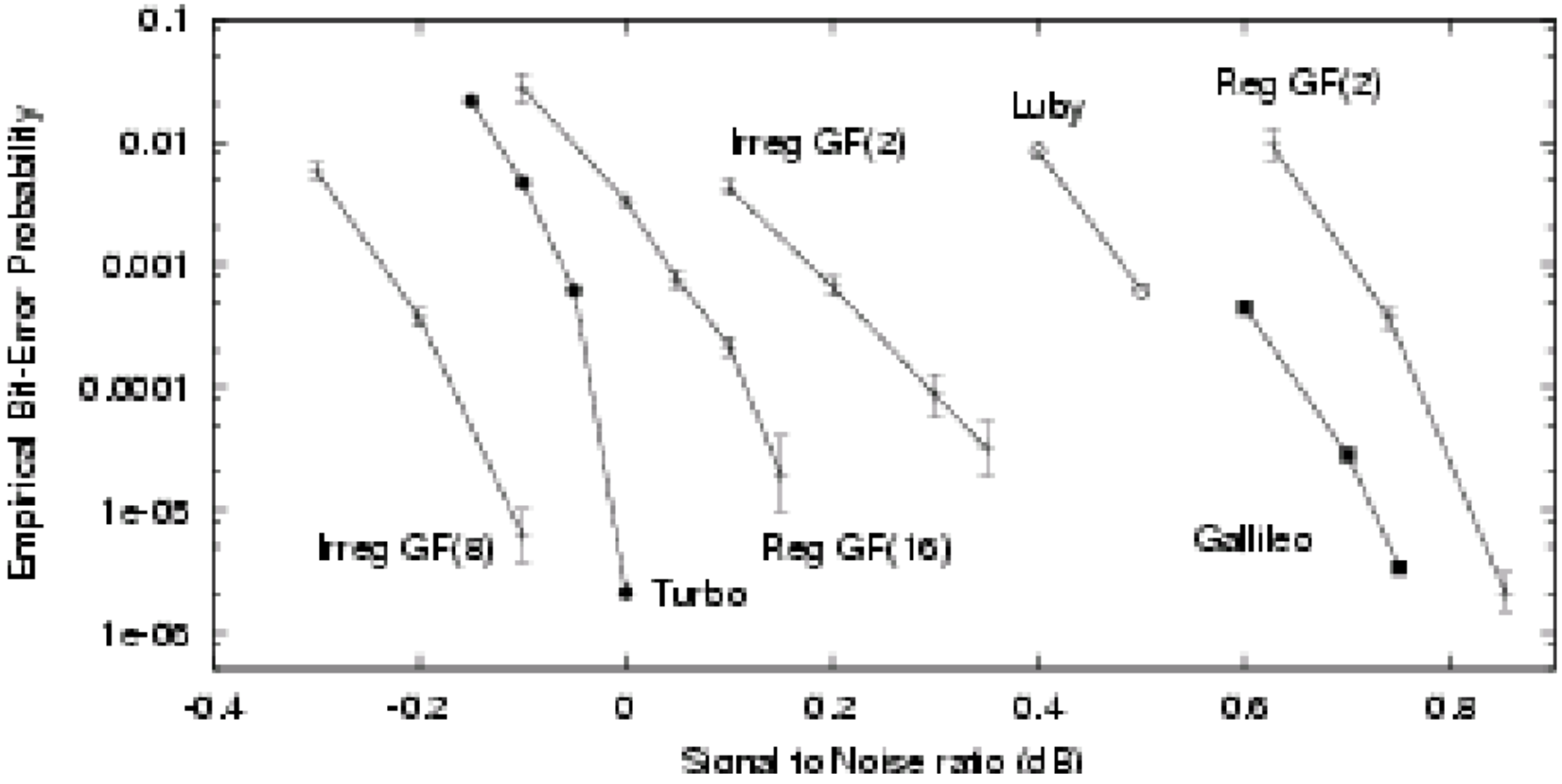


Figure 1: From left to right, rate 1/2 turbo codes, irregular codes, and regular codes.

From MacKay's Website



From MacKay's Website

- The figure shows the performance of various codes with rate 1/4 over the Gaussian Channel. From left to right:
- Irregular low density parity check code over GF(8), blocklength 48000 bits (Davey and MacKay, 1999);
- JPL turbo code (JPL, 1996) blocklength 65536;
- Regular LDPC over GF(16), blocklength 24448 bits (Davey and MacKay, 1998);
- Irregular binary LDPC, blocklength 16000 bits (Davey, 1999);
- M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi and D.A. Spielman's (1998) irregular binary LDPC, blocklength 64000 bits;
- JPL's code for Galileo: a concatenated code based on constraint length 15, rate 1/4 convolutional code (in 1992, this was the best known code of rate 1/4); blocklength about 64,000 bits;
- Regular binary LDPC: blocklength 40000 bits (MacKay, 1999).

Conclusions

- **The inherent parallelism in decoding LDPC codes suggests their use in high data rate systems.**
- **A comparison of LDPC codes and turbo codes is complicated and depends on many issues: e.g., block length, channel model, etc.**
- **LDPC codes are well worthwhile investigating. Some issues to be resolved are:**
  - **Performance for channel models of interest**
  - **Optimization of irregular LDPC codes (for channels of interest).**
  - **Implementation in VLSI.**
  - **Patent issues.**