Chapter 7

Information Theory and Coding

Contents

7.1 Introduction 7-3
7.2 Information Theory 7-3
7.2.1 Entropy 7-4
7.2.2 Discrete Channel Models 7-5
7.2.3 Joint and Conditional Entropy 7-6
7.2.4 Channel Capacity 7-6
7.3 Source Coding 7-8
7.3.1 Shannon-Fano Source Coding 7-11
7.3.2 Huffman Source Coding 7-11
7.4 Communications in Noisy Environments 7-11
7.5 Forward Error Correction Coding 7-15
7.5.1 Block Codes 7-15
7.5.2 Convolutional Codes 7-29
7.5.3 Low Density Parity Check (LDPC) Codes 7-36
7.5.4 Trellis-Coded Modulation (TCM) 7-37
7.5.5 Turbo Codes 7-39
7.5.6 MATLAB Support for FEC Coding 7-40

7.1 Introduction

Z&T Chapter 12 is devoted to information theory and coding. The motivation for this study is original work of Claude Shannon in the late 1940's. Information theory provides a means to evaluate communication system performance compared to a *theoretically best* system for a given bandwidth and SNR.

- We can measure the information contained in a message and determine how to best transfer the information from the source to the destination
- Coding is a major application area of information theory
- The result is provided by *Shannon's coding theorem* is that if a source has information at a rate less than the channel capacity, there exists a coding procedure such the source can be transmitted with arbitrary small probability of error.

7.2 Information Theory

- What is information?
- For some event x_j with corresponding probability $p(x_j)$, the information is defined as

$$I(x_j) = \log_a \left(\frac{1}{p(x_j)} \right) = -\log_a p(x_j)$$

- Consistent with common sense, more information is conveyed by events having a smaller probability

- If the base $a = 2$, the units associated with information is the binary unit or *bit*

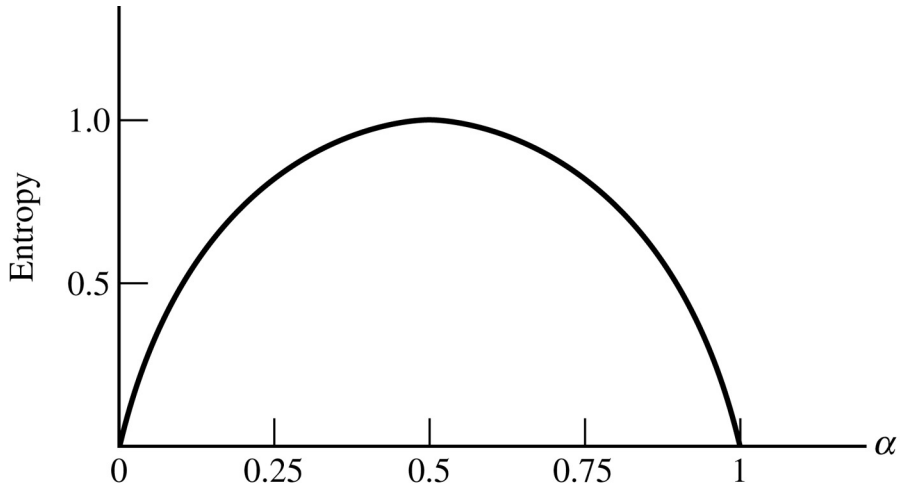
7.2.1 Entropy

- The average information provided by a source or output, is defined as the *entropy*

$$H(X) = E\{I(x_j)\} = -\sum_{j=1}^{n} p(x_j) \log_2 p(x_j)$$

- The entropy of a binary source having $p(1) = \alpha$ and $p(0) = 1 - \alpha = \beta$ is

$$H(\alpha) = -\alpha \log_2 \alpha - (1 - \alpha) \log_2 (1 - \alpha)$$



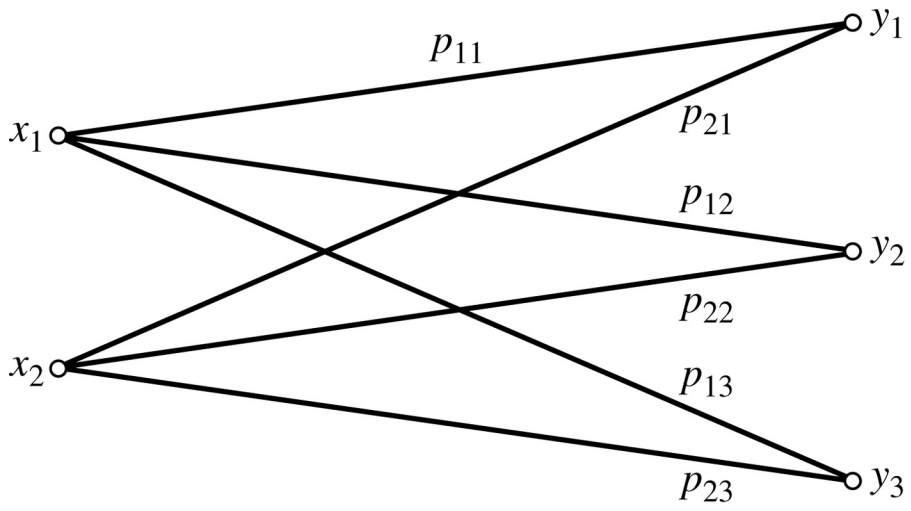
Entropy of a binary source as $p(1) = \alpha$ varies

- We have the maximum entropy when $\alpha = 1/2$; Is this reasonable?

- For an n outcome source $p_k = 1/n$ for $k = 1, \ldots, n$, which in turn gives the maximum entropy

7.2.2 Discrete Channel Models

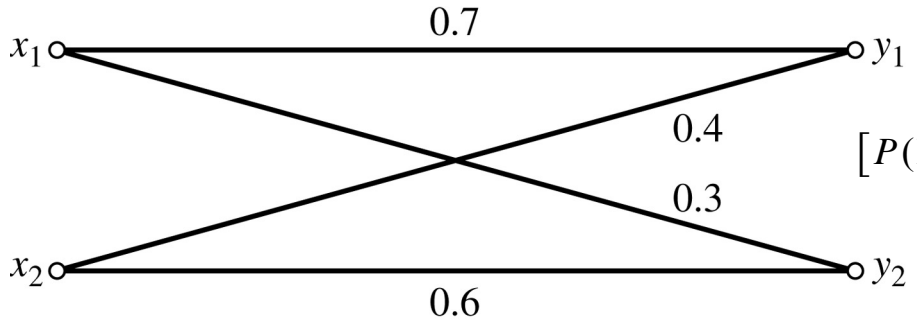
- In discussions of information theory and coding, a discrete memoryless channel (DMC) is often assumed
- The DMC is described in terms of conditional (transition) probabilities that relate the channel input to the channel output state



Two-input three-output DMC

- The probabilities can be placed into a *channel matrix*

$$\left[P(Y|X)\right] = \begin{bmatrix} p(y_1|x_1) & p(y_2|x_1) & p(y_3|x_1) \\ p(y_1|x_2) & p(y_2|x_2) & p(y_3|x_2) \\ p(y_1|x_3) & p(y_2|x_3) & p(y_3|x_3) \end{bmatrix}$$



$$\left[P(Y|X)\right] = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}$$

Binary channel

7.2.3 Joint and Conditional Entropy

- Beyond entropy defined earlier, we can also define conditional entropy and joint entropy

- These additional forms are useful in defining channel capacity, defined shortly

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$
$$H(Y) = -\sum_{j=1}^{m} p(y_j) \log_2 p(y_j)$$
$$H(Y|X) = -\sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log_2 p(y_j|x_i)$$
$$H(X,Y) = -\sum_{j=1}^{m} p(x_i, y_j) \log_2 p(x_i, y_j)$$

- Note that:

$$\begin{aligned} H(X,Y) &= H(X|Y) + H(Y) \\ &= H(Y|X) + H(X) \end{aligned}$$

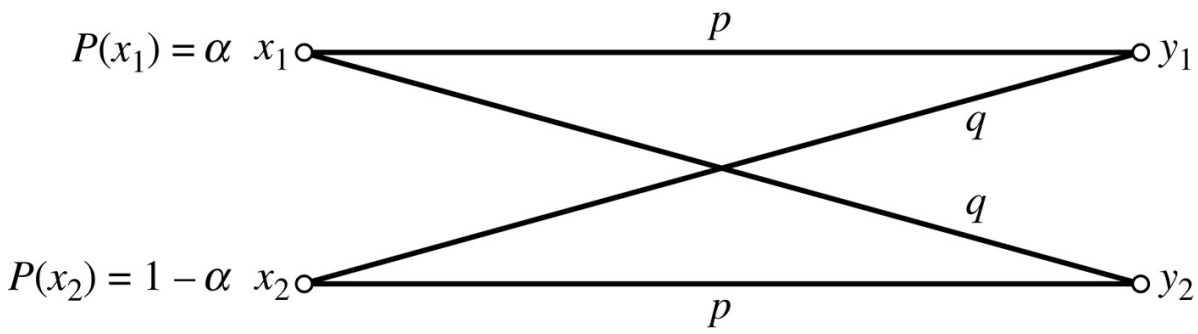
7.2.4 Channel Capacity

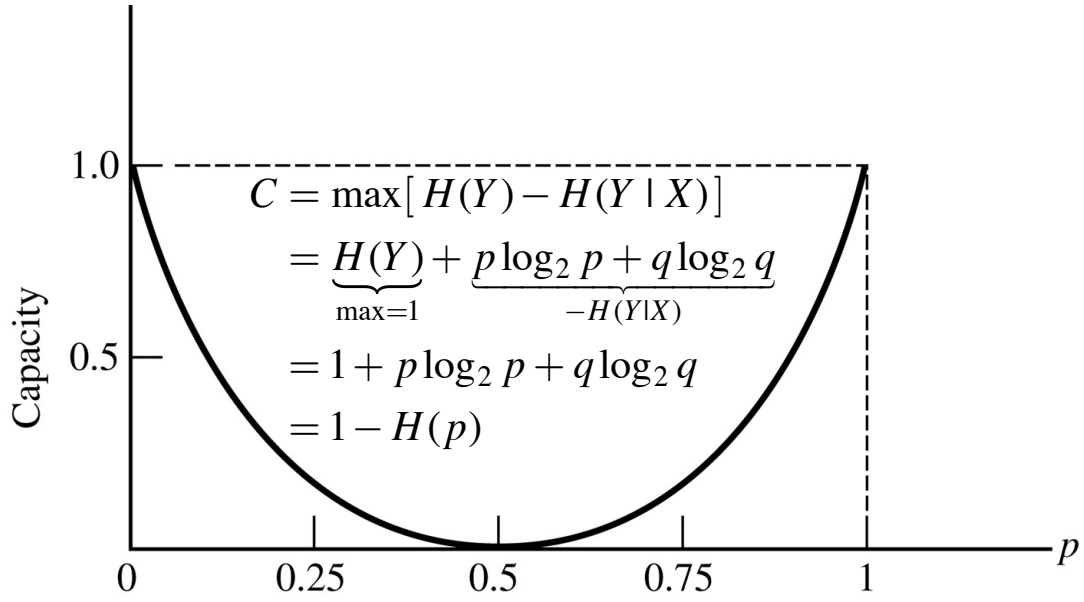
- Using the definitions of joint and conditional entropy, we can define the *mutual information* $I(X;Y)$

$$I(X;Y) = H(X) - H(X|Y) \geq 0$$

- If X is the channel input and Y is the channel output, the *decrease* in average uncertainty of the transmitted signal when it is received, is the *mutual information*
- The channel capacity, C, is defined as the maximum value of mutual information, that is the maximum average information per symbol that can be transmitted through the channel upon each use

$$C = \max[I(X;Y)]$$



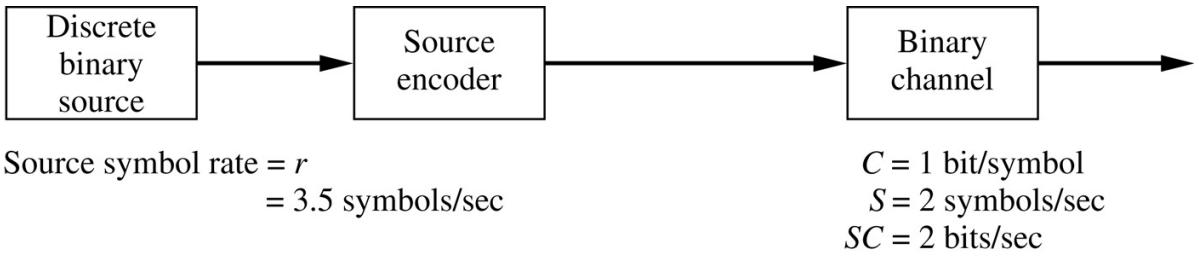


Capacity of the binary symmetric channel (BSC)

- C is a function of both the source probabilities and the channel transition probabilities

7.3 Source Coding

- Match the source to the channel via a data compression coding scheme
- As an introductory example consider the following transmission scheme block diagram



- From what we know about the source entropy $H(X)$, the source information rate is given by

$$R_s = rH(X) \text{ bps}$$

where r is the symbol rate in symbols per second $H(X)$ has units of bits, actually bits per symbol

- The Shannon *noiseless coding theorem* states that

> *Given a channel and a source that generates information at a rate less than the channel capacity, it is possible to code the source output in a such a manner that it can be transmitted through the channel*

- In the above block diagram we assume a binary source with outputs A and B having probability 0.9 and 0.1 respectively
- The source rate is 3.5 symbols/sec

- The channel capacity is one bit per symbol, since we assume that we have a binary symmetric channel with $p = 1$
- The available symbol rate for the channel is $S = 2$ symbols/sec
- Presently the source symbol rate being $3.5 > 2$ symbols/sec, means that we cannot send the source symbols directly over the channel with negligible error
- Note that the *source information rate*, $rH(x)$, is

$$\begin{aligned} rH(x) &= 3.5\big[-0.1\log_2 0.1 - 0.9\log_2 0.9\big] \\ &= 3.5 \cdot 0.469 = 1.642 \text{ bps} \end{aligned}$$

- The source information rate is **less** than the channel capacity, so transmission is possible, we just need to devise a source code
- One simple approach is with order n extension of the original source, that is n-symbol groups of source symbols are formed and then assigned code words of increasing length as the symbol probability decreases

Third-Order extension ($n = 3$) for the source coding example

Source symbol	Symbol probability $P(\cdot)$	Code word	l_i	$P(\cdot)l_i$
AAA	0.729	0	1	0.729
AAB	0.081	100	3	0.243
ABA	0.081	101	3	0.243
BAA	0.081	110	3	0.243
ABB	0.009	11100	5	0.045
BAB	0.009	11101	5	0.045
BBA	0.009	11110	5	0.045
BBB	0.001	11111	5	0.005

- The average word length here is $\bar{L} = 1.598$

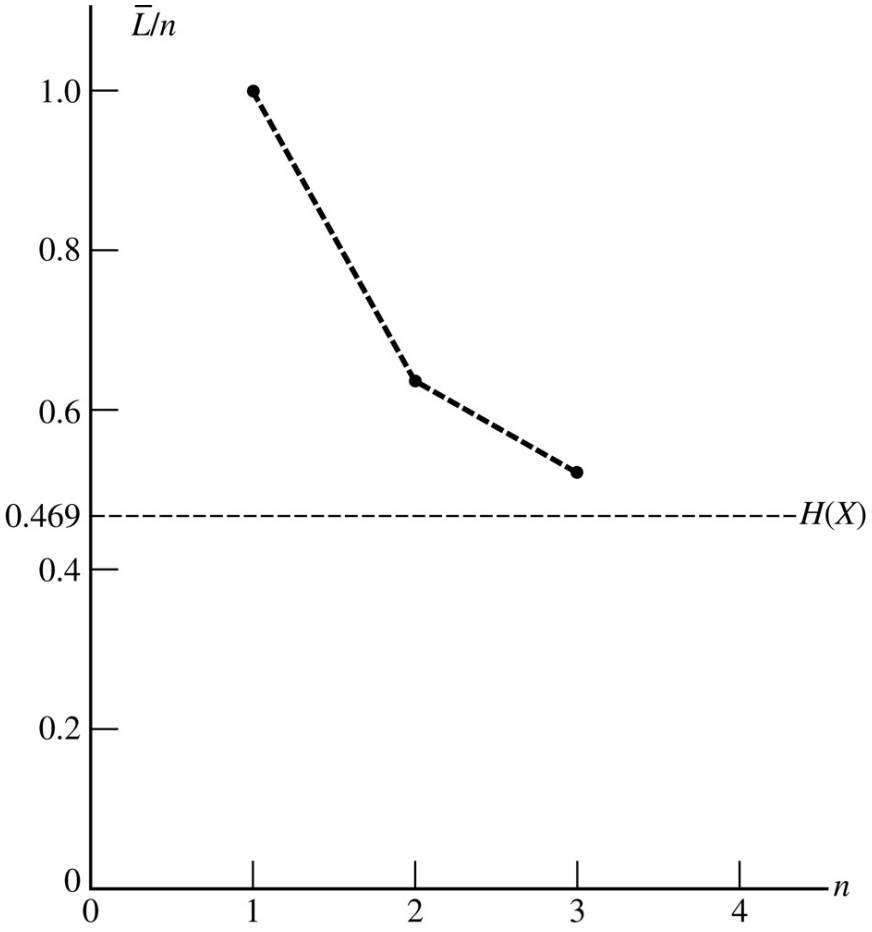
$$\frac{\bar{L}}{n} = \frac{1}{n}\sum P(\cdot)l_i = \frac{1.598}{3} = 0.5333 \text{ code symb/src symb}$$

- The symbol rate at the encoder output is

$$r\frac{\bar{L}}{n} = 3.5(0.5333) = 1.864 \text{ code symb/s}$$
$$< S = 2 \text{ chan symb/s}$$

so transmission is now possible!

- In general $\bar{L}/n$ exceeds the source entropy, but approaches it as n becomes large



$\bar{L}/n$ in example as n steps from one to three

7.3.1 Shannon-Fano Source Coding

See Z&T page 632.

7.3.2 Huffman Source Coding

See Z&T page 632–634 .

7.4 Communications in Noisy Environments

- We now take the first steps towards channel coding
- The Shannon fundamental theorem of information theory states that:

  Given a discrete memoryless source (each symbol perturbed by noise independently of all other symbols) with capacity C and a source with positive rate R, where $R < C$, there exists a code such that the output of the source can be transmitted over the channel with an arbitrary small probability of error.

Shannon-Hartley Law

- The AWGN channel has capacity in bits/s given by

$$C_c = B \log_2 \left(1 + \frac{S}{N}\right)$$

  where B is the channel bandwidth in Hz and S/N is the signal-to-noise power ratio

- With C_c a trade-off between channel bandwidth and SNR is established
- We now investigate what happens to the capacity when we try to make the bandwidth very large
- We know that $E_b = ST_b$ (recall S is signal power) and at capacity the bit rate $R_b = 1/T_b = C_c$ bits/s, so

$$E_b = ST_b = \frac{S}{C_c}$$

- The noise power in bandwidth B is

$$N = N_0 B$$

so

$$\frac{S}{N} = \frac{E_b}{N_0} \cdot \frac{C_c}{B}$$

- Rewriting the Shannon-Hartley law we have

$$\frac{C_c}{B} = \log_2\left(1 + \frac{E_b}{N_0} \cdot \frac{C_c}{B}\right)$$

- Solving for E_b/N_0 we have

$$\frac{E_b}{N_0} = \frac{B}{C_c}\left(2^{C_c/B} - 1\right)$$

- We now consider $B \gg C_c$ using the expansion $e^x \simeq 1 + x,\ |x| \ll 1$:

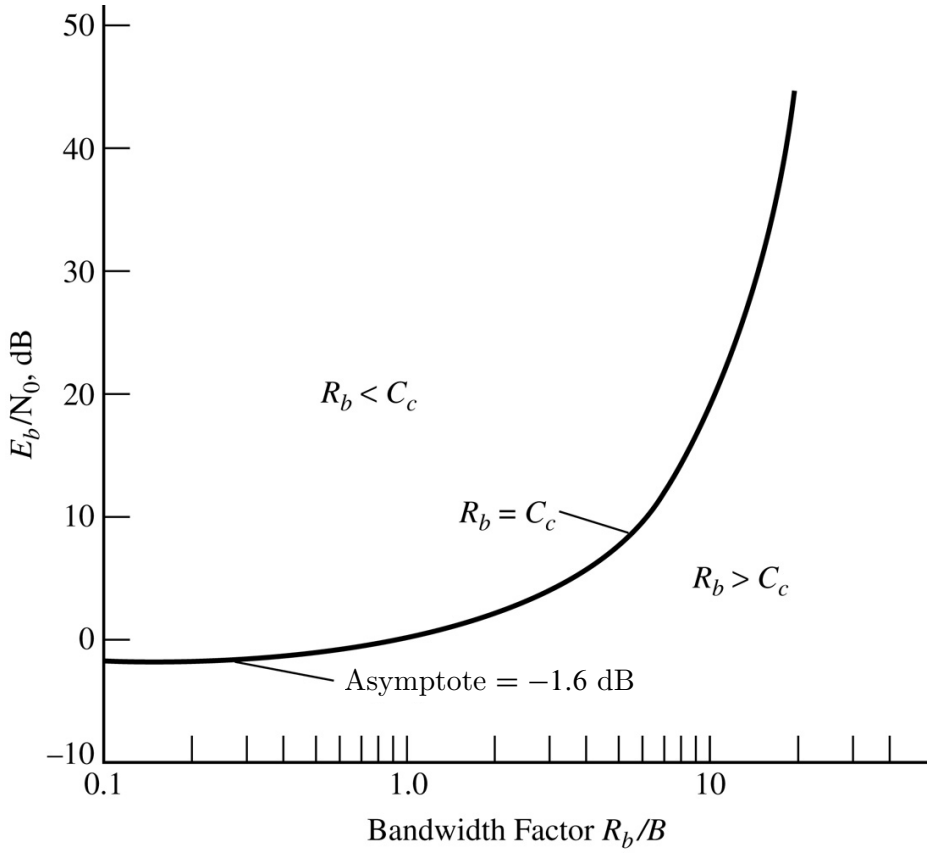
$$2^{C_c/B} = e^{(C_c/B)\ln 2} \simeq 1 + \frac{C_c}{B}\ln 2$$

- Under the $B \gg C_c$ assumption we then have

$$\frac{E_b}{N_0} \simeq \frac{B}{C_c}\left(1 + \frac{C_c}{B}\ln 2 - 1\right) = \ln 2 = -1.6 \text{ dB}$$

- We conclude that in an ideal system, where $R_b = C_c$, the limiting value for E_b/N_0, as B grows without bound, is -1.6 dB

- The plot below shows this relationship along with regions where $R_b < C_c$ exist as we plot E_b/N_0 versus R_b/B

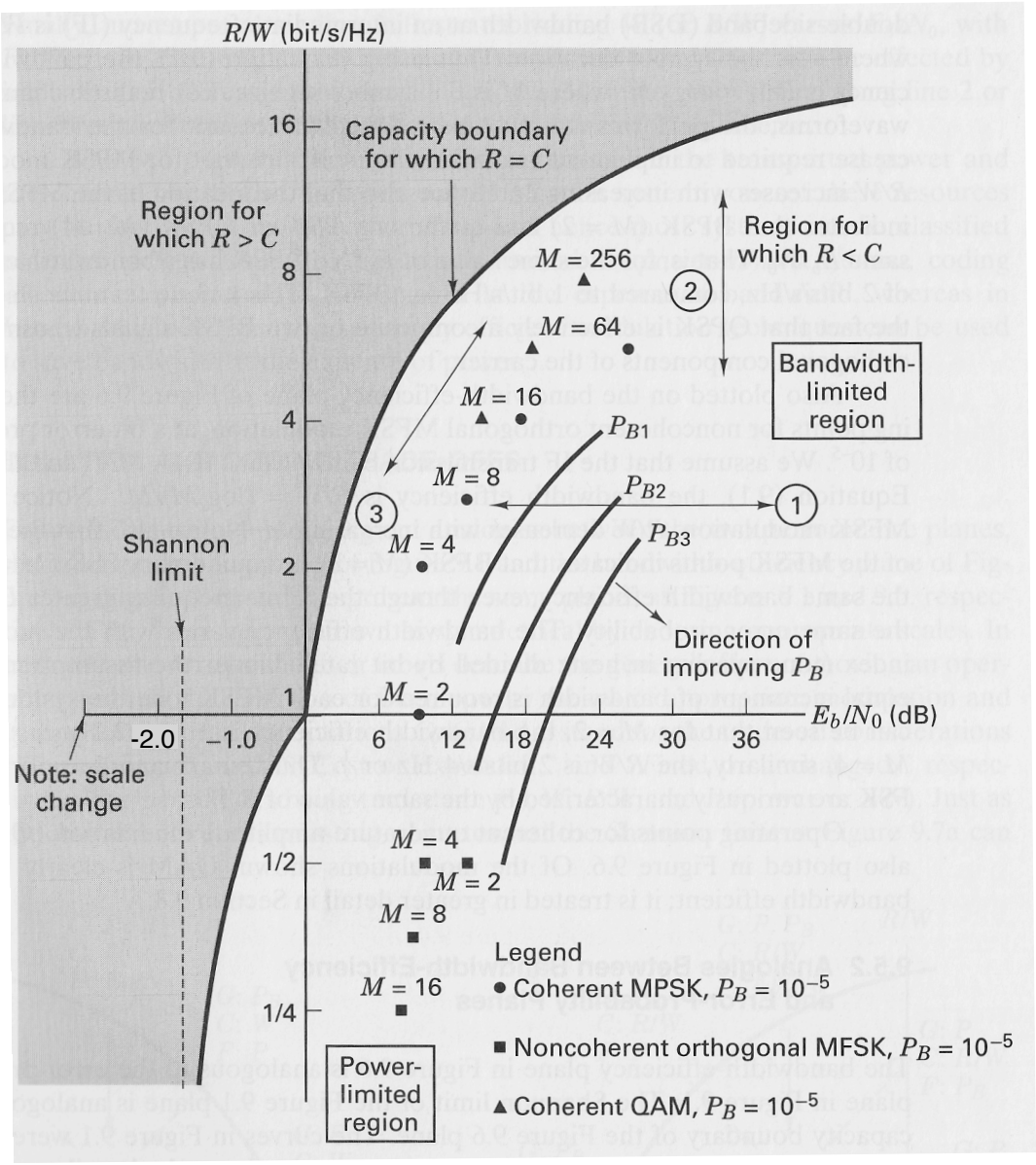


AWGN channel capacity

- Above and to the left of the curve is where realizable systems must operate, in particular for R_b/B large E_b/N_0 is also large

- For $R_b < C_c$ and $B \gg R_b$, we need E_b/N_0 just greater than -1.6 dB, i.e., $S \simeq R_b(\ln 2)N_0$ W

- Another way of plotting this function is found in Sklar[1], where it is referred to as the power-bandwidth efficiency plane



Power-bandwidth efficiency plane

- Comparisons between MPSK, MQAM, and MFSK are also provided

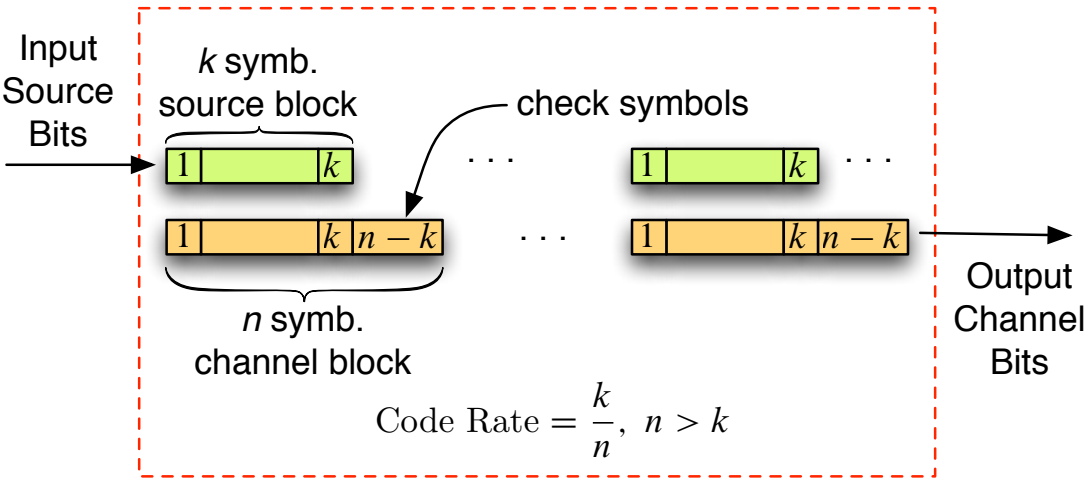
[1]B. Sklar, *Digital Communications: Fundamentals and Applications*, second edition, Prentice Hall, New Jersey, 2001.

7.5 Forward Error Correction Coding

- How do we move closer to Shannon's limit as seen in the two previous figures?
- One approach is through the design of the modulation scheme itself
- Another approach, and one of the main topics of this chapter, is via forward error correction (FEC) code design
- Coding can be used to combat the effects of channel noise
- Two fundamental classes of codes for FEC are *block* and *convolutional* codes

7.5.1 Block Codes

- With block coding the serial source symbols are grouped into k-symbol blocks and then $n-k$ check symbols, to make code words of length $n > k$; the code is denoted (n, k)



- The check symbols allow for correction or at least detection of errors

- The desire is to achieve the desired error correcting with code rate as close to one as possible

Hamming Distances and Error Correction

- We can view error correction/detection from a geometric point of view

- Consider a code word composed of n bits, 1s and 0s

- The *Hamming distance*, d_{ij}, between two such code words s_i and s_j is defined as the number of positions in which s_i and s_j differ

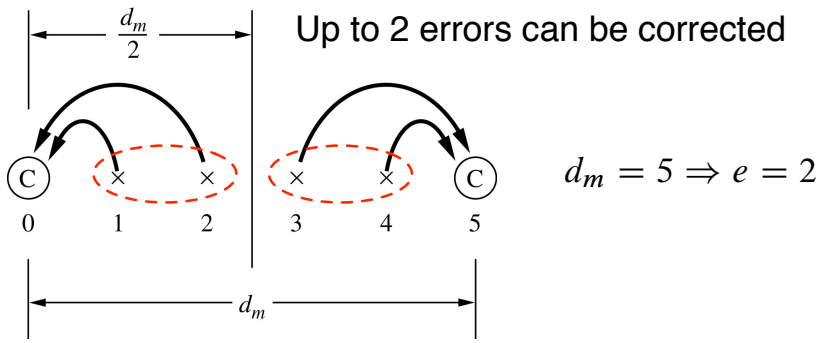
$$d_{ij} = w(s_i \oplus s_j),$$

where $\oplus$ denotes modulo-2 addition (XOR) and $w(\ )$ is the *Hamming weight*, which counts the number of 1s of the code word in its argument

- The geometric view tells us that if two code words are distance 5 apart, a minimum-distance decoder can correct as many as

$$e = \left\lfloor \frac{d_m - 1}{2} \right\rfloor \text{ errors},$$

where d_m is the minimum distance between two code words



Geometric view of two code words and the impact of errors

Single-Parity Check Codes

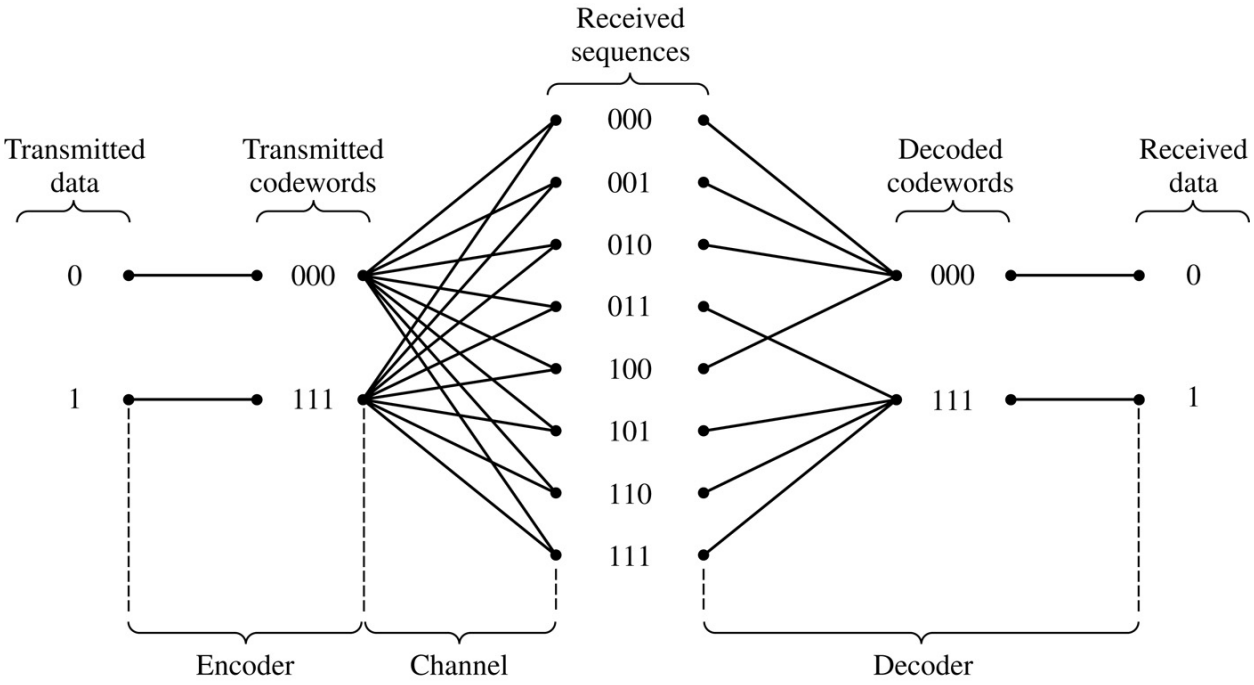
- A very simple block code that can only detect errors, is when one check symbol is added to the k information symbols to form a $k + 1$ length block, thus the code rate $= k/(k + 1)$
- The added symbol is a *parity-check symbol* which is used to make the code word Hamming distance either even or odd
- If the word contains an *even* number of errors error detection will not occur, but for an *odd* number of errors (in particular a single error), we know the word contains an error

Repetition Codes

- Another very simple block code, that is capable of correcting errors, is the repetition code where each symbol is repeated n times
- Hence we have $n - 1$ check symbols making

$$\text{code rate} = \frac{1}{n}$$

- For $n = 3$ one error can be corrected ($e = (3-1)/2 = 1$)



Repetition code example

- The BEP with this code is

$$P_b = \sum_{i=e+1}^{n} \binom{n}{i} p^i (1-p)^{n-i},$$

  where p is the BSC channel error probability

- The information rate increase ($R_c = nR_s$) associated with the repetition code, makes its use limited
  - When the uncoded $P_E = p$ is exponential with E_b/N_0 we cannot overcome the bandwidth expansion
  - When P_E is algebraic (think Rayleigh fading), the repetition is effective, i.e., like diversity combining

Parity Check for Single Errors

- Practical codes for digital communication try to strike a balance between error correction and maintaining a high information rate (code rate close to one)
- One simple code that does this is the single error correction parity-check codes
- To each k-symbol block we add $r = n - k$ parity check symbols

$$\underbrace{a_1\ a_2\ \cdots\ a_k}_{\text{source symb..}}\ \underbrace{c_1\ c_2\ \cdots\ c_r}_{\text{parity check symb.}}$$

- Choose the $r = n - k$ check symbols such that

$$\begin{aligned} 0 &= h_{11}a_1 \oplus h_{12}a_2 \oplus \cdots \oplus h_{1k}a_k \oplus c_1 \\ 0 &= h_{21}a_1 \oplus h_{22}a_2 \oplus \cdots \oplus h_{2k}a_k \oplus c_2 \\ &\vdots \qquad\qquad\qquad \vdots \\ 0 &= h_{r1}a_1 \oplus h_{r2}a_2 \oplus \cdots \oplus h_{rk}a_k \oplus c_r \end{aligned}$$

or

$$\mathbf{HT} = \big[H\big]\big[T\big] = \big[0\big] = \mathbf{0}$$

where $\mathbf{H}$ is the *parity check* matrix

$$\mathbf{H} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1k} & 1 & 0 & \cdots & 0 \\ h_{21} & h_{22} & \cdots & h_{2k} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{r1} & h_{r2} & \cdots & h_{rk} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

and $\mathbf{T}$ is the *code word* vector

$$\mathbf{T} = \begin{bmatrix} a_1 & a_2 & \cdots & a_k & c_1 & \cdots & c_r \end{bmatrix}^T$$

- In place of vector $\mathbf{T}$, suppose $\mathbf{R} = \left[R\right]$ is a received sequence (vector) of length n

$$\mathbf{HR} \neq \mathbf{0} \Rightarrow \text{at least one error is present}$$

- To decode $\mathbf{R}$ and correct the error, we start by writing

$$\mathbf{R} = \mathbf{T} \oplus \mathbf{E},$$

  where $\mathbf{E}$ is a length n error pattern induced by the channel

- We need to determine $\mathbf{E}$ from $\mathbf{R}$ using $\mathbf{H}$

- Let

$$\mathbf{S} = \left[S\right] = \mathbf{HR} = \mathbf{HT} \oplus \mathbf{HE} = \mathbf{HE},$$

  since $\mathbf{HT} = \mathbf{0}$

- The matrix/vector $\mathbf{S}$ is known as the *syndrome*

- We observe that $\mathbf{HE}$, for the case of a single error, returns

$$\mathbf{E} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

  and the i th column of $\mathbf{H}$ (i th row of $\mathbf{E}$) is where the error occurs

Example 7.1: A (6, 3) Code

- Given

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

- Assume that we receive 111011
- The syndrome is

$$\mathbf{S} = \mathbf{HR}$$

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

- The syndrome is column 3 of the parity check matrix, so the error is in the third symbol, meaning the decoded word is 110011
- The syndrome of 110011 is **0**, as expected

Hamming Codes

- The Hamming (n, k) code was discovered by Richard Hamming in 1950
- For positive integer $m \geq 3$ we have

Code word length	$n = 2^m - 1$
Message block	$k = 2^m - 1 - m$
Parity-check block	$n - k = m$
Error correcting capability	$e = t = 1$
Minimum Hamming distance	$d_m = 3$

- The parity check matrix is of dimension $2^{n-k} - 1$ by $n - k$
- The ith column of the matrix $\mathbf{H}$ is made the binary representation of i, making the syndrome for a single error the binary representation of the position in error

Example 7.2: A (7, 4) Code

- Given

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

- Assume that we receive 1110001
- The syndrome is

$$\mathbf{S} = \mathbf{HR}$$

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

- The error is thus in position 7 as $(111)_b = 7$, so the decoded word is 1110000
- Note that the parity checks are in columns 1, 2, and 4, as these columns contains a single 1 each

---

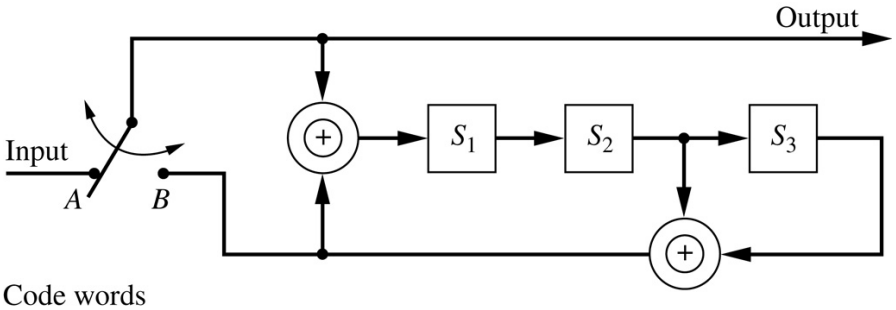
Cyclic Block Codes

- A linear block code is a cyclic code if a cyclic shift of any code word produces another valid code word, e.g.,

$$x_1 x_2 \cdots x_{n-1} x_n \quad \text{and} \quad x_n x_1 \cdots x_{n-2} x_{n-1}$$

- The motivation here is encoder and decoder implementation ease

- An (n, k) cyclic code can be easily generated using an $n - k$ stage shift register with feedback
  - *Position A*: Shift $k = 4$ information bits into the decoder (position A)
  - *Position B*: Shift out remaining $n-k$ bits, the register also zeros at the end of this operation, ready to load a new set of information bits



Code words

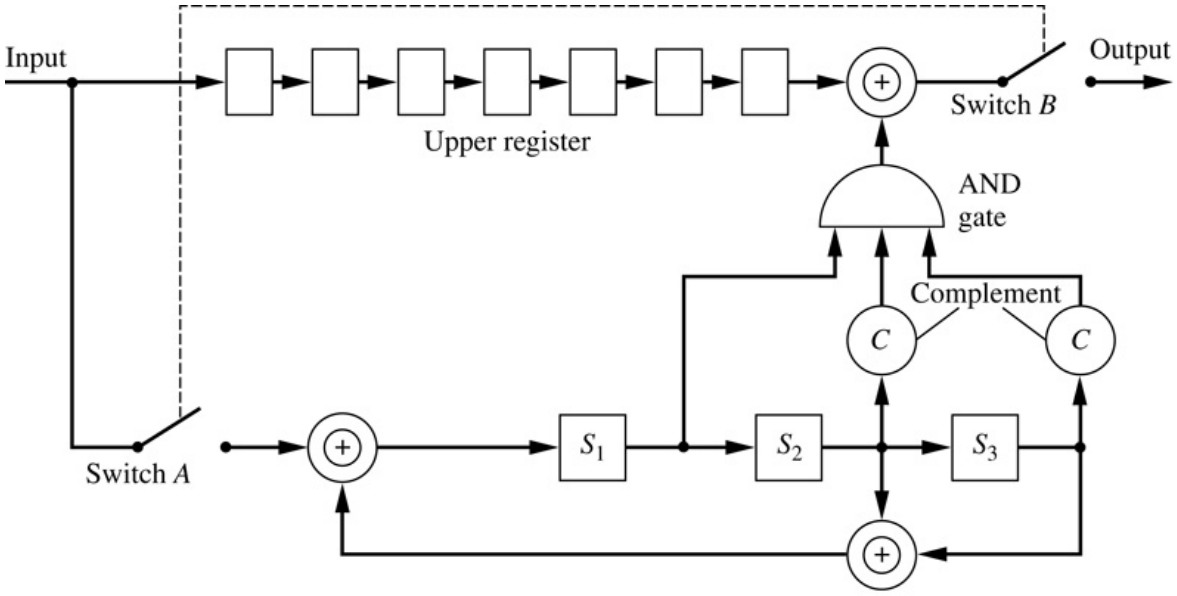
0000000
1000101
1100010
1110100
1111111
0001011
0011101
0111010
1011000
1101001
1001110
0110001
0100111
0010110
0101100
1010011

Register contents for input word 1101

Shift	Register content	Output
1	100	1
2	110	1
3	111	0
4	111	1
	Switch set to position *B*	
5	011	0
6	001	0
7	000	1

Cyclic block code generation

- The decoder is a bit more complicated, but still easy to implement



Received word 1101001 (no errors)

	Shift	Input	Lower register content	Output
Switch *A* closed Switch *B* open	1	1	100	
	2	1	110	
	3	0	111	
	4	1	111	
	5	0	011	
	6	0	001	
	7	1	000	
Switch *A* open Switch *B* closed	8		000	1
	9		000	1
	10		000	0
	11		000	1
	12		000	0
	13		000	0
	14		000	1

Received word 1101011 (one errors)

Shift	Input	Lower register content	Output
1	1	100	
2	1	110	
3	0	111	
4	1	111	
5	0	011	
6	1	101	
7	1	010	
8		101	1
9		110	1
10		111	0
11		011	1
12		001	0
13		100	0
14		010	1

AND gate output inverts upper register output

Cyclic (7, 4) code decoding

- Examples of popular cyclic codes include:
  - Golay: $e = 3$,
  - Bose-Chaudhuri-Hocquenghem (BCH): $e < 2^{m-1}$, $m \geq 3$,
  - Reed Solomon (RS): non-binary with each symbol carrying 2^m bits, $n = 2^m - 1$, parity block $n - k = 2e$
- The RS code is good at dealing with burst errors and is part of the playback standard for compact disk digital audio

Performance Comparison

- The performance of all block codes revolves around q_u and q_c, the uncoded and coded symbol error probabilities, respectively
- We are also interested in P_{eu} and P_{ec}, the uncoded and coded word error probabilities, respectively
- Assuming independent errors,

$$P_{eu} = 1 - (1 - q_u)^k$$

- If a code can correct up to e errors, then

$$P_{ec} = \sum_{i=e+1}^{n} \binom{n}{i} (1 - q_c)^{n-i} q_c^i$$

- For *perfect codes* (Hamming codes and Golay (23,12)) the above expression for P_{ec} is exact

- In general it is possible that for certain error sequences, in which more than e errors occur, they may also be corrected
- The above P_{ec} expression still forms a tight bound in most cases
- Word error probabilities are only useful when n-symbol words carry and equal number of information bits
- In Z&T the Torrieri[2] BEP bounds for block codes is discussed, in particular the expression

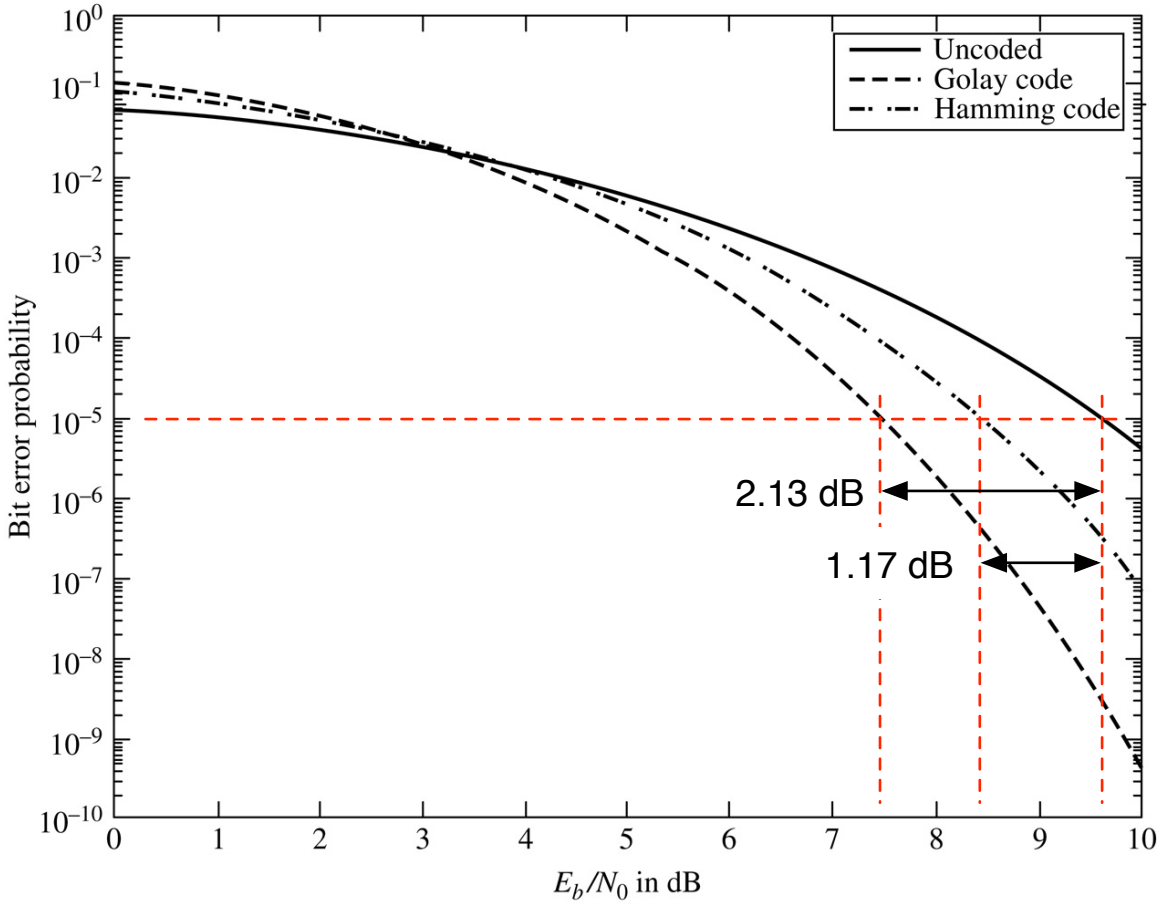
$$P_b = \frac{q}{2(q-1)}\left[\sum_{i=e+1}^{d}\frac{d}{n}\binom{n}{i}P_s^i(1-P_s)^{n-i} + \frac{1}{n}\sum_{i=d+1}^{n} i\binom{n}{i}P_s^i(1-P_s)^{n-i}\right]$$

  where P_s is the channel symbol error probability (q_u for the binary case or just p for the BSC), e the number of correctable errors, $d = 2e+1$, q is the code alphabet size ($q = m = 2$ in the binary case, 2^m for RS codes)
- Finding the exact BEP can be difficult

[2]D.J. Torierri, *Principles of Secure Communication Systems* second edition, Artech House, 1992

Example 7.3: **Golay and Hamming Code Example**

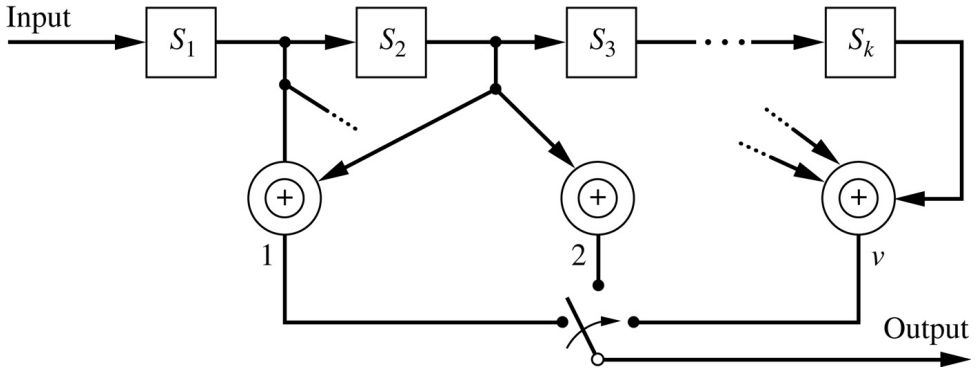
- Compare the performance of a (15,11) Hamming code with a (23,12) Golay code, and an uncoded system, all in AWGN
- The Hamming code can correct just one error, while the Golay code can correct up to 3 errors
- The Torrieri expressions are used to compute the BEP
- See Z&T p. 656 for the MATLAB code (convert to Python easy)



(15,11) Hamming and (23,12) Golay code BEP performance

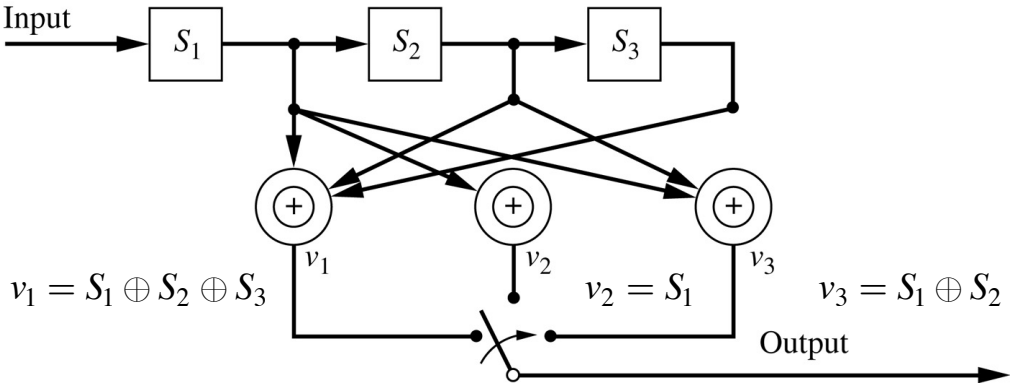
7.5.2 Convolutional Codes

- Convolutional codes are generated with a *constriant span*, $k = K$, in place of the block length and parity symbols
- For each new information symbol multiple code symbols are produced at the output via a commutator



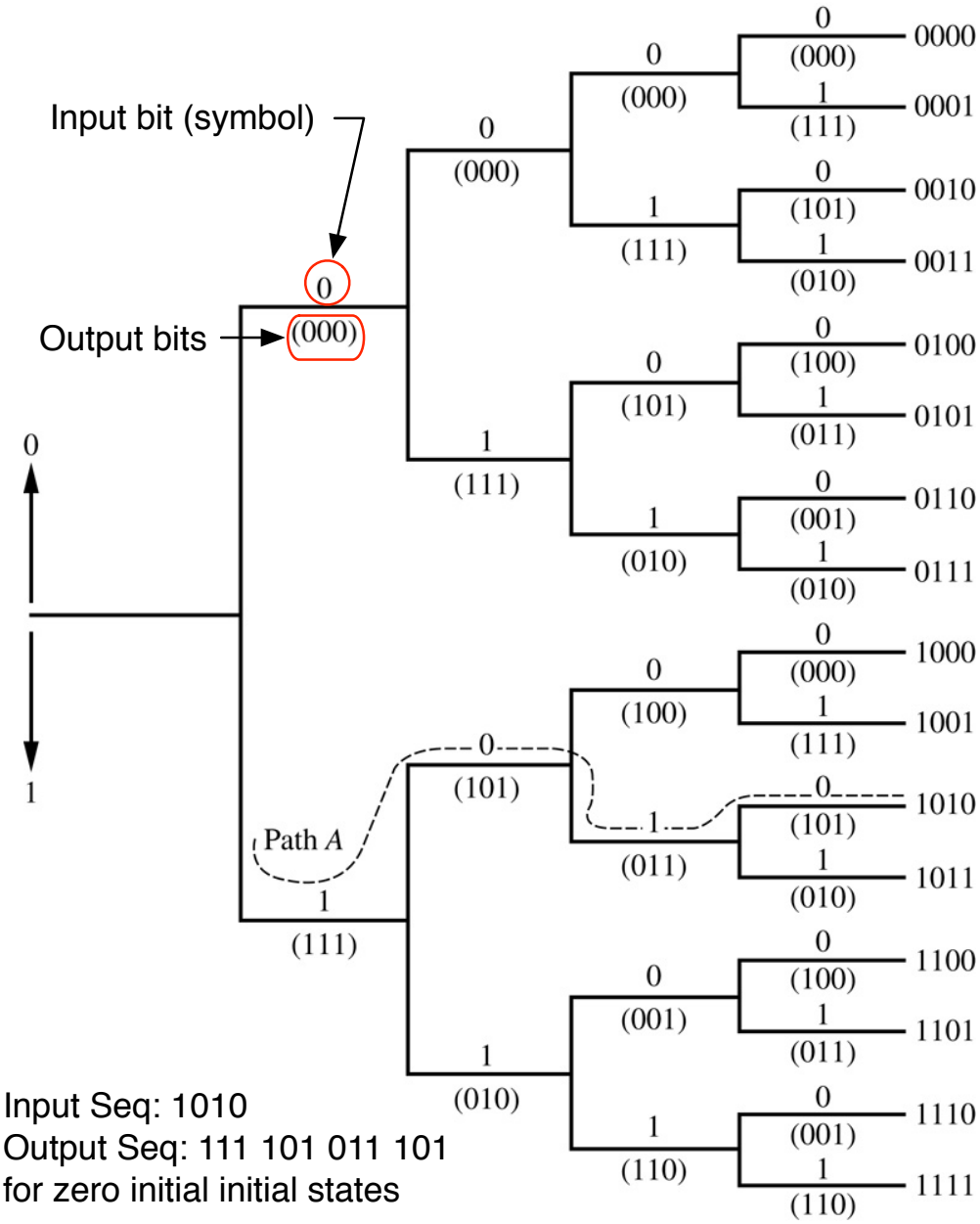
General convolutional coder

- With v outputs for every input, the code rate becomes $1/v$
  - Rate k/v convolutional codes are also possible, and there is also a technique known as *puncturing* which removes selected output symbols
- Consider a rate 1/3 constraint length 3 coder



Rate 1/3 constraint length $K = 3$ coder

- To decode convolutional codes the Viterbi algorithm (VA) is most often employed
- The encoder introduces memory into the output sequence that can be traced using a *code tree*, but the tree continues to grow in size (2^N branches, N binary input symbols) as the *traceback* distance increases



Rate 1/3 constraint length $K = 3$ code tree

- The use of a trellis structure, which is part of the VA, makes the decoding process manageable
- To understand the VA we first consider the state as (S_1, S_2) and consider what happens when a new bit enters the encoder

(a) Definition of States

State	S_1	S_2
A	0	0
B	0	1
C	1	0
D	1	1

(b) State Transitions

Previous				Current				
State	S_1	S_2	Input	S_1	S_2	S_3	State	Output
A	0	0	0	0	0	0	A	(000)
			1	1	0	0	C	(111)
B	0	1	0	0	0	1	A	(100)
			1	1	0	1	C	(011)
C	1	0	0	0	1	0	B	(101)
			1	1	1	0	D	(010)
D	1	1	0	0	1	1	B	(001)
			1	1	1	1	D	(110)

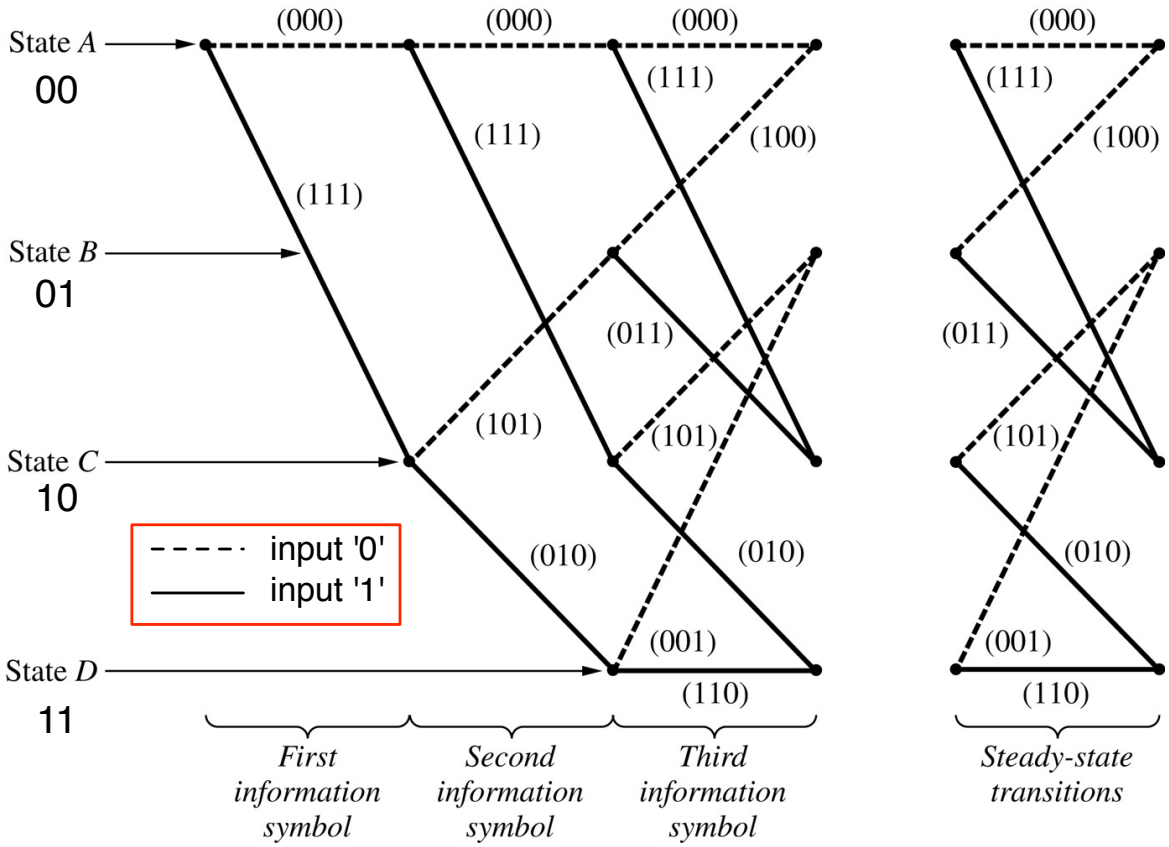
(c) Encoder Output for State Transition $x \rightarrow y$

Transition	Output
$A \rightarrow A$	(000)
$A \rightarrow C$	(111)
$B \rightarrow A$	(100)
$B \rightarrow C$	(011)
$C \rightarrow B$	(101)
$C \rightarrow D$	(010)
$D \rightarrow B$	(001)
$D \rightarrow D$	(110)

Rate 1/2 constraint length $K = 3$ states and state transitions

- The present value of (S_2, S_3) (or previous (S_1, S_2)) along with the current input bit S_1, completely describes the behavior of the encoder and can be used to set up the VA trellis used for decoding

- Using the above tables, we now construct the trellis



Rate 1/3 constraint length $K = 3$ trellis

- The VA uses the trellis by searching backwards to find the most *likely* path that was used to arrive at the current state

- The traceback process, uses metrics formed according to the minimum Hamming distance between the received symbols and a given trellis path

- By keeping as a *survivor* the minimum Hamming distance path to each of the present states, the traceback paths in theory all come from a common path through the trellis, which in turn corresponds to the input bit sequence

- Divergence from the correct path occurs when errors cannot be corrected, but due to the periodicity of the trellis only make for a finite run of decoded errors (e.g., $K = 3$ in this case)

Calculations for Viterbi Algorithm: Step One (Received Sequence = 110101011)

Path[1]	Corresponding symbols	Hamming distance	Survivor?
AAAA	000000000	6	No
ACBA	111101100	4	Yes
ACDB	111010001	5	Yes[2]
AACB	000111101	5	No[2]
AAAC	000000111	5	No
ACBC	111101011	1	Yes
ACDD	111010110	6	No
AACD	000111010	4	Yes

[1]The initial and terminal states are identifted by the first and fourth letters, respectively. The second and third letters correspond to intermediate states.

[2]if two or more paths have the same Hamming distance, it makes no difference which is retained as the survivor.

Calculations for Viterbi Algorithm: Step Two (Received Sequence = 110101011111)

Path[1]	Previous survivor's distance	New segiment	Added distance	New distance	Survivor?
ACBAA	4	*AA*	3	7	Yes
ACDBA	5	*BA*	2	7	No
ACBCB	1	*CB*	1	2	Yes
AACDB	4	*DB*	2	6	No
ACBAC	4	*AC*	0	4	Yes
ACDBC	5	*BC*	1	6	No
ACBCD	1	*CD*	2	3	Yes
AACDD	4	*DD*	1	5	No

[1]An underscore indicates the previous survivor.

Performance Comparisons

- The BEP performance of a convolutional code can be approximated using the *weight structure/spectrum* bound[3]
- The Z&T text provides MATLAB code for calculating this bound

$$P_b < \sum_{k=d_{\text{free}}}^{\infty} c_k P_k,$$

where d_{free} is the free distance of the code, the Hamming weight of the minimum length error event path, P_k is the probability of an error event path of length k occurring, and c_k is weight that gives the number of bit errors associated with the error event path (the c_k's are found in a table)

- The P_k's are calculated as follows

$$P_k = \begin{cases} \sum_{e=(k/2)+1}^{k} \left\{ \binom{k}{e} p_e (1-p)^{k-e} \right. \\ \left. + \frac{1}{2}\binom{k}{k/2} p^{k/2}(1-p)^{k/2} \right\}, & k = \text{even} \\ \sum_{e=(k+1)/2}^{k} \binom{k}{e} p^e (1-p)^{k-e}, & k = \text{odd} \end{cases}$$

- Finally, for the AWGN channel (BPSK in this case)

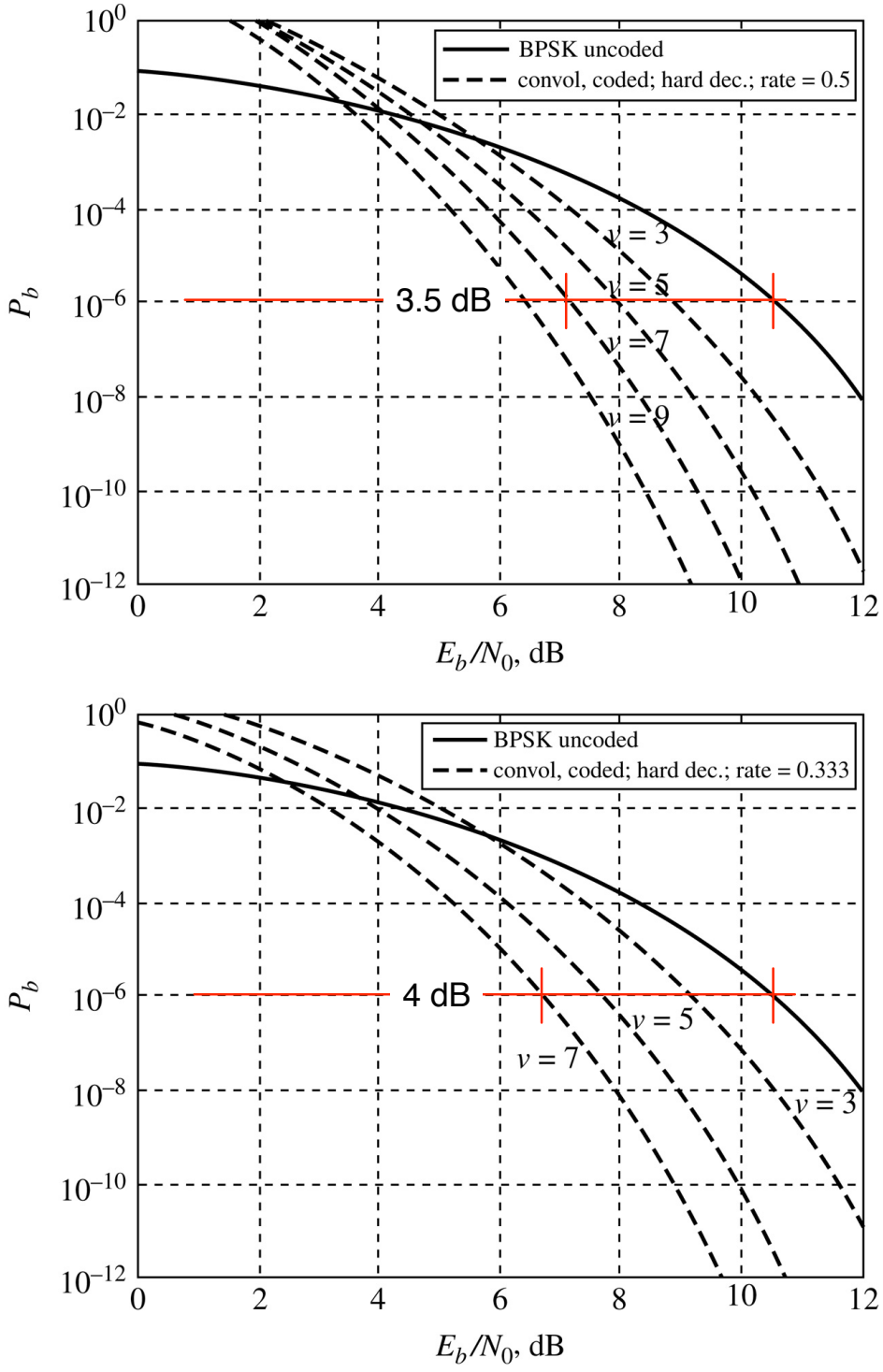
$$p = Q\left(\sqrt{\frac{2kRE_b}{N_0}}\right)$$

where R is the code rate

- Tables of the weights can be found in Ziemer & Peterson

[3] R. Zeimer and R. Peterson, *Introduction to Digital Communications*, second edition, Prentice Hall, New Jersey, 2001

Example 7.4: Rate 1/2 and 1/3 Codes, K a Parmeter



Code performance of rate 1/2 and 1/3 codes

7.5.3 Low Density Parity Check (LDPC) Codes

- An LDPC code is a linear block code having parity check matrix **H** which is sparse
- These codes were originally discovered in 1962[4], but were rediscovered by MacKay and Neal in 1996[5]
- The significance of these codes is that using long LDPC codes we can approach the Shannon limit to within a few tenths of a dB!
- LDPC codes also offer both better performance and lower decoding complexity than other codes[6]
- *Regular* LDPC codes have a block length of n, a parity check matrix that has exactly w_r ones in each row and exactly w_c ones in each column, where $w_c < w_r < n$ (there are also *irregular* LDPC codes)
- The rows of **H** do not have to be linearly independent
- The code dimension is controlled by the rank of **H**
- Sparseness of **H** makes a code more efficient to decode

[4]R. Gallager, "Low-Density Parity-Check Codes," *IRE Trans. Inform. Theory*, pp. 21-28, January 1962.

[5]D. J. C. MacKay and R. M. Neal, "Near Shannon-Limit Performance of Low-Density Parity-Check Codes," Electronics Letters, vol. 32, pp. 1645–1646, Aug. 1996.

[6]J. Barry, E. Lee, and D. Messerschmitt, *Digital Communications*, third edition, Kluwer Academic Publishers, 2004.

- As an example parity check matrix of a regular (12, 6) code is[7]

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

- Decoding LDPC codes is accomplished using *message passing algorithms* (MPA)

- The algorithms are iterative, but not difficult to implement in practice, hence the popularity

7.5.4 Trellis-Coded Modulation (TCM)

Combined coding and modulation to achieve both bandwidth and power efficiency gains. See Z&T pages 668-672.

A practical example taken from SATCOM is combining a rate 1/2 convolutional code with an 8-PSK modulator. The comparison system is QPSK.

- With QPSK we have two bits per symbol

- With 8-PSK TCM the first bit is encoded rate 1/2 to two code bits

- The second bit is sent as the third 8-PSK bit making the symbol complete

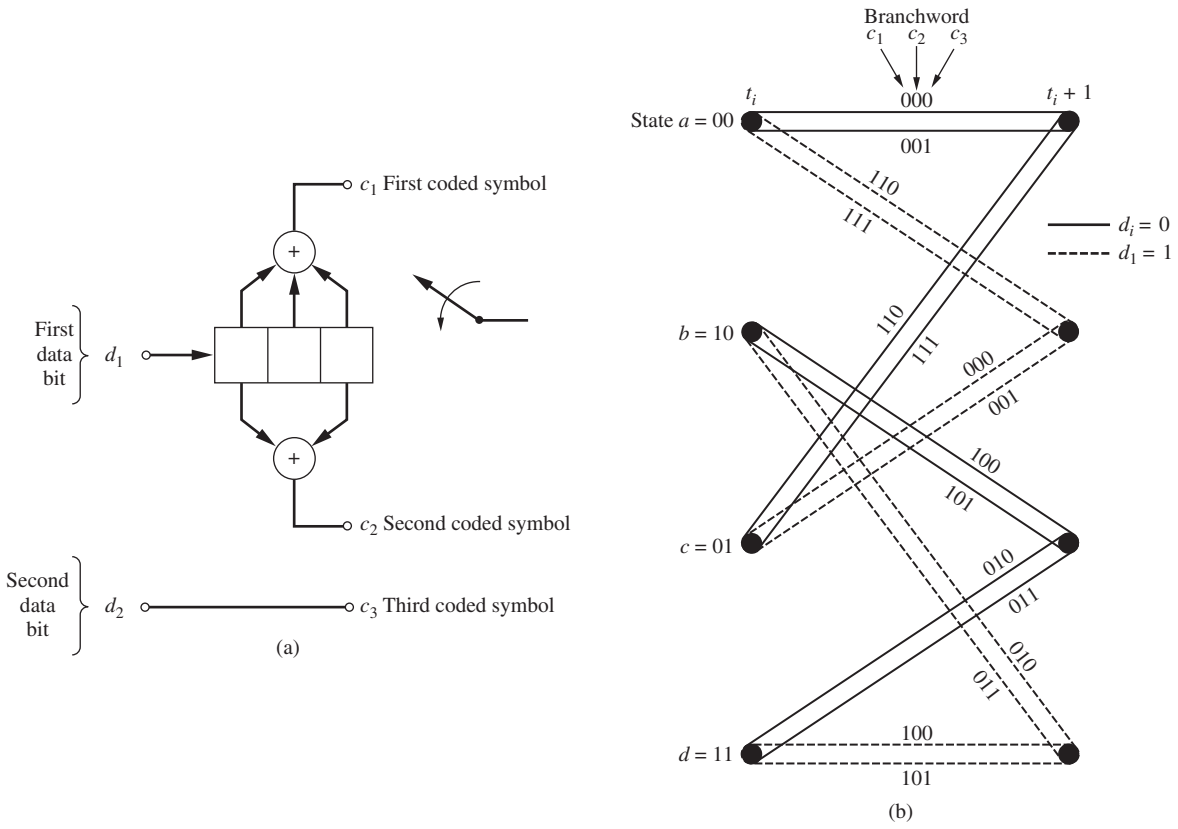
[7] T. Ha, *Theory and Design of Digital Communication Systems*, Cambridge University Press, London, 2011.

- The effective code rate is now $R = 2/3$

- The asymptotic coding gain over QPSK is 3 dB with no change in bandwidth required

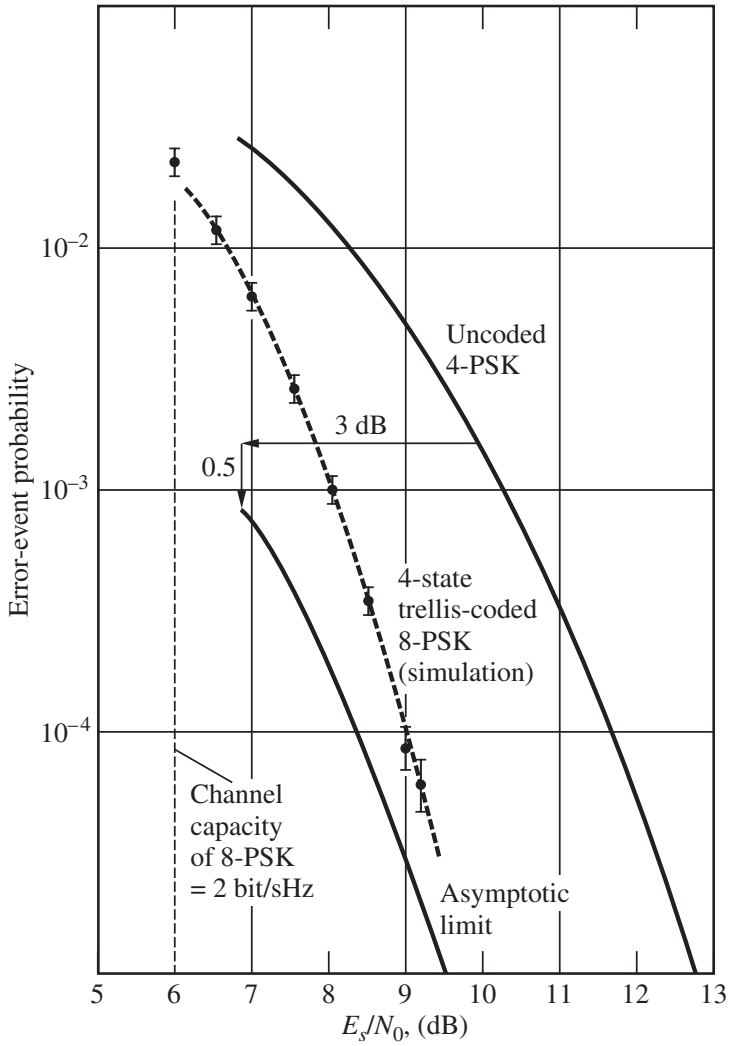
- Very simple decoding is also possible, which gives less coding gain, but requires only a simple 4-state Viterbi algorithm

- The coder and trellis structure found in Z&T are shown below



8-PSK TCM: (a) coder, (b) trellis.

- The error probability of scheme in AWGN is compared with QPSK below:



Performance for a 4-state, 8-PSK TCM signaling scheme. (From G. Ungerboeck, “Trellis-Coded Modulation with Redundant Signal Set, Part I: Introduction,” IEEE Communications 10–2 Magazine, February 1987, Vol. 25, pp. 5–11.)

7.5.5 Turbo Codes

See Z&T pages 681-683.

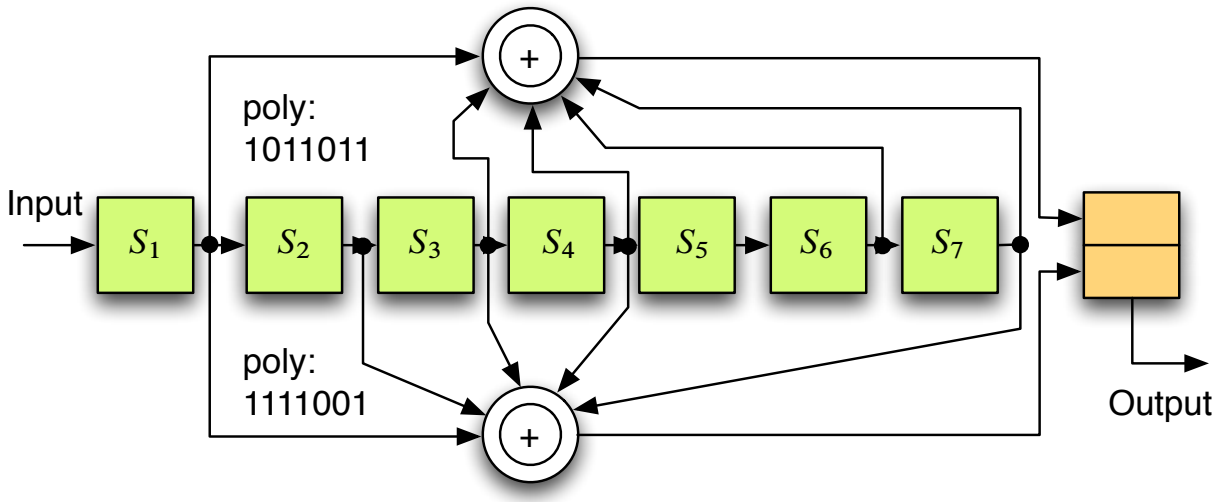
7.5.6 MATLAB Support for FEC Coding

Error-Control Coding

bchdec	BCH decoder
bchenc	BCH encoder
bchgenpoly	Generator polynomial of BCH code
bchnumerr	Number of correctable errors for BCH code
convenc	Convolutionally encode binary data
cyclgen	Produce parity-check and generator matrices for cyclic code
cyclpoly	Produce generator polynomials for cyclic code
decode	Block decoder
dvbs2ldpc	Low-density parity-check codes from DVB-S.2 standard
encode	Block encoder
fec.bchdec	Construct BCH decoder object
fec.bchenc	Construct BCH encoder object
fec.ldpcdec	Construct LDPC decoder object
fec.ldpcenc	Construct LDPC encoder object
fec.rsdec	Construct Reed-Solomon decoder object
fec.rsenc	Construct Reed-Solomon encoder object
gen2par	Convert between parity-check and generator matrices
gfweight	Calculate minimum distance of linear block code
hammgen	Produce parity-check and generator matrices for Hamming code
rsdec	Reed-Solomon decoder
rsdecof	Decode ASCII file encoded using Reed-Solomon code
rsenc	Reed-Solomon encoder
rsencof	Encode ASCII file using Reed-Solomon code
rsgenpoly	Generator polynomial of Reed-Solomon code
syndtable	Produce syndrome decoding table
vitdec	Convolutionally decode binary data using Viterbi algorithm

MATLAB Comm toolbox ™FEC coding functions

Example 7.5: Rate 1/2 $K = 7$ Convolutional Code Simulation



Rate 1/2 $K = 7$ Coder

- To set up the encoder we need to first build the trellis using `poly2trellis()`
- This function takes as its input octal words describing the connections in the block diagram

```
function [Pe,errors,N_RecBits] = convCoder_test(SNRdB,NminErrors)
% [Pe,errors,N_RecBits] = convCoder_test(SNR,Nerrors)
%
% Mark Wickert December 2010

% Convert uncoded SNR = Eb/N0 to channel Eb/N0
% Since the code is rate 1/2 the factor is 3 dB
SNR = SNRdB - 10*log10(2);

%Create a Rate 1/2 conv encoder with K = 7 and code generators
% G1 = 1111001, G2 = 1011011.
% Create in octal form for poly2trellis function
K = 7;
G1_b = '1111001';
G2_b = '1011011';
```

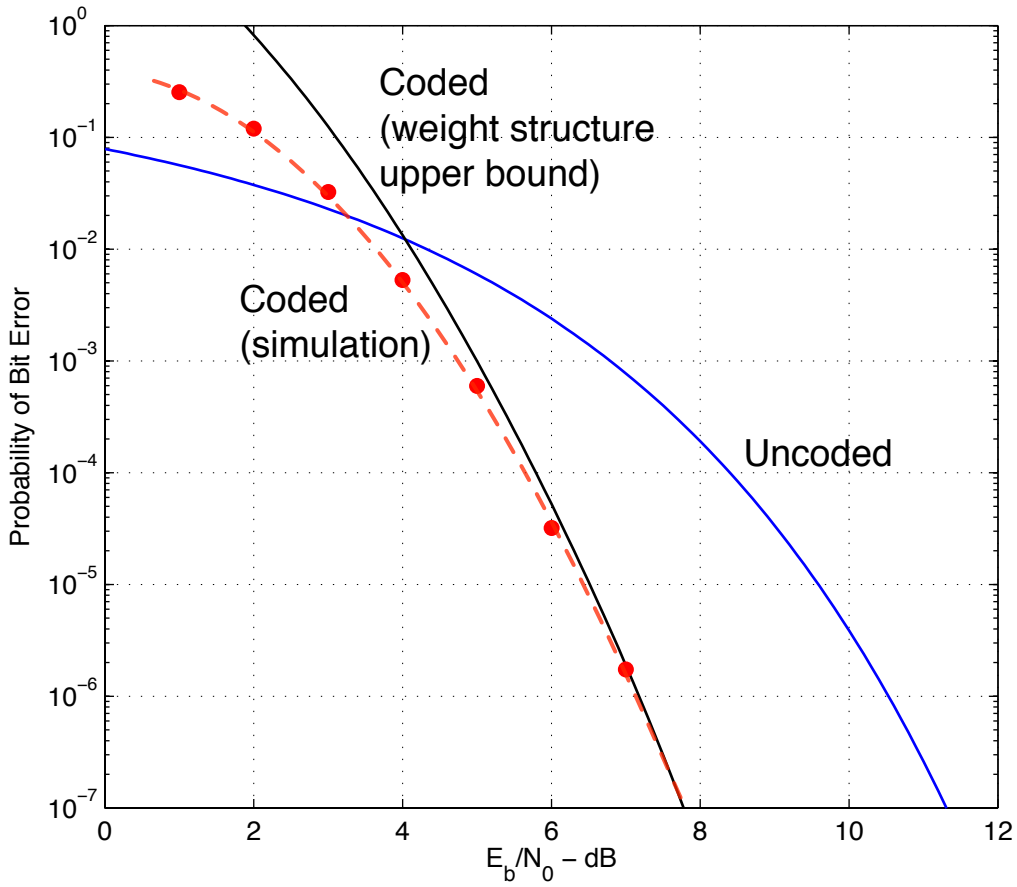
```
G1_o = str2num(dec2base(bin2dec(G1_b),8));
G2_o = str2num(dec2base(bin2dec(G2_b),8));
% Create the trellis structure
trellis = poly2trellis(K,[G1_o G2_o]);
% Initialize error counters
errors = 0;
N_RecBits = 0;
while errors < NminErrors
    % Create a sample input message
    N = 2000000;
    msg = randi([0,1],1,N);
    % Encode msg
    code_bits = convenc(msg,trellis);
    % Baseband BPSK
    code_bits = 2*code_bits - 1;
    % Add noise
    %SNR = 0
    code_bits = real(cpx_AWGN(code_bits,SNR,1));
    % Make hard decisions
    code_bits = sign(code_bits);
    % Convert -1/+1 logic to 0/1 logic
    code_bits = (code_bits+1)/2;
    % Decode using hard (Hamming weight metrics)-decision Viterbi, and a
    % traceback depth of 42 bits.
    TRACEBACK = 42;
    decoded = vitdec(code_bits,trellis,TRACEBACK,'cont','hard');
    % Error detect
    errors = errors + sum(xor(msg(1:end-42), decoded(42+1:end)));
    N_RecBits = N_RecBits + length(msg(1:end-42));
end

Pe = errors/N_RecBits;
my_label = sprintf('SNR = %2.2f, errors = %d, BEP = %2.2e',...
                   SNRdB,errors,Pe);
disp(my_label);
```

- Collect BEP simulation data:

```
>> [Pe,errors,N_RecBits] = convCoder_test(0,2500);
SNR = 0.00, errors = 744413, BEP = 3.72e-01
>> [Pe,errors,N_RecBits] = convCoder_test(1,2500);
SNR = 1.00, errors = 508620, BEP = 2.54e-01
>> [Pe,errors,N_RecBits] = convCoder_test(2,2500);
SNR = 2.00, errors = 237292, BEP = 1.19e-01
>> [Pe,errors,N_RecBits] = convCoder_test(3,2500);
```

```
SNR = 3.00, errors = 64808, BEP = 3.24e-02
>> [Pe,errors,N_RecBits] = convCoder_test(4,100);
SNR = 4.00, errors = 10805, BEP = 5.40e-03
>> [Pe,errors,N_RecBits] = convCoder_test(5,500);
SNR = 5.00, errors = 1008, BEP = 5.04e-04
>> [Pe,errors,N_RecBits] = convCoder_test(6,500);
SNR = 6.00, errors = 509, BEP = 3.64e-05
>> [Pe,errors,N_RecBits] = convCoder_test(7,250);
SNR = 7.00, errors = 254, BEP = 1.74e-06
>> % Obtain the bounding results and uncoded
>> % Uncoded (last parameter == 2)
>> Pb = conv_Pb_bound(1/2,10,[36 0 211 0 1404 0 11633 0],SNRdB,2);
>> % Coded hard decision (last parameter == 0)
>> Pb_h = conv_Pb_bound(1/2,10,[36 0 211 0 1404 0 11633 0],SNRdB,0);
>> % Coded soft decision (last parameter == 1)
>> Pb_s = conv_Pb_bound(1/2,10,[36 0 211 0 1404 0 11633 0],SNRdB,1);
```

- The results are plotted in the figure below



BEP results, uncoded, coded upper bound, and simulation

- MATLAB code for the error bounding formulas is listed below

```
function Pb = conv_Pb_bound(R,dfree,Ck,SNRdB,hard_soft)
% Pb = conv_Pb_bound(R,dfree,Ck,SNR,hard_soft)
%
% Convolution coding bit error probability upper bound
% according to Ziemer & Peterson 7-16, p. 507
%
% Mark Wickert July 2001

Pb = zeros(1,length(SNRdB));
SNR = 10.^(SNRdB/10);

for n=1:length(SNR)
    for k=dfree:(length(Ck)+dfree-1)
        if hard_soft == 0       % Evaluate hard decision bound
            Pb(n) = Pb(n) + Ck(k-dfree+1)*hard_Pk(k,R,SNR(n));
        else                    % Evaluate soft decision bound
            Pb(n) = Pb(n) + Ck(k-dfree+1)*soft_Pk(k,R,SNR(n));
        end
    end
    if hard_soft == 2   % Compute Uncoded Pe
        Pb(n) = gaussQ(sqrt(2*SNR(n)));
    end
end

function Pk = hard_Pk(k,R,SNR)
% Pk = hard_Pk(k,R,SNR)
%
% Calculates Pk as found in Ziemer & Peterson eq. 7-12, p.505
%
% Mark Wickert July 2001

p = gaussQ(sqrt(2*R*SNR));
Pk = 0;

if 2*fix(k/2) == k
    for e=k/2+1:k
        Pk = Pk + ...
        factorial(k)/(factorial(e)*factorial(k-e))*p^e*(1-p)^(k-e);
    end
    Pk = Pk + ...
        1/2*factorial(k)/(factorial(k/2)*factorial(k-k/2))*p^(k/2)*(1-p)^(k/2);
else
    for e=(k+1)/2:k
        Pk = Pk + ...
```

```
        factorial(k)/(factorial(e)*factorial(k-e))*p^e*(1-p)^(k-e);
    end
end

function Pk = soft_Pk(k,R,SNR)
% Pk = soft_Pk(k,R,SNR)
%
% Calculates Pk as found in Ziemer & Peterson eq. 7-13, p.505
%
% Mark Wickert July 2001

Pk = gaussQ(sqrt(2*k*R*SNR));

function p = gaussQ(x)

p = 1/2*erfc(x/sqrt(2));
```

Note: The above MATLAB example can also be run using the Python module `fec_conv.py` in `scikit-dsp-comm`. Functions for calculating the soft decision coding bounds are also included. See the final project Fall 2021, Problem 4, for more details.

Note: Block coding FEC can also be explored in `scikit-dsp-comm` using the module `fec_block.py`. An example notebook can be found on read-the-docs.