# Chapter 7

## Information Theory and Coding

### Contents

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

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

![Graph showing entropy of a binary source as $p(1) = \alpha$ varies](image)

• 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

\[
\begin{bmatrix}
    p(y_1|x_1) & p(y_2|x_1) & p(y_3|x_1) \\
    p(y_1|x_2) & p(y_2|x_2) & p(y_3|x_2) \\
    p(y_1|x_3) & p(y_2|x_3) & p(y_3|x_3)
\end{bmatrix}
\]

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.

\[
\begin{align*}
H(X) &= - \sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \\
H(Y) &= - \sum_{j=1}^{m} p(y_j) \log_2 p(y_j) \\
H(Y|X) &= - \sum_{i=1}^{n} \sum_{j=1}^{m} p(x_i, y_j) \log_2 p(y_j|x_i) \\
H(X, Y) &= - \sum_{j=1}^{m} p(x_i, y_j) \log_2 p(x_i, y_j)
\end{align*}
\]

- Note that:

\[
H(X, Y) = H(X|Y) + H(Y) = H(Y|X) + H(X)
\]

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

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

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

$$C = \max[H(Y) - H(Y \mid X)]$$

$$= H(Y) + \underbrace{p \log_2 p + q \log_2 q}_{\text{max}=1} - H(Y \mid X)$$

$$= 1 + p \log_2 p + q \log_2 q$$

$$= 1 - H(p)$$
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

\[
rH(x) = 3.5 \left[ -0.1 \log_2 0.1 - 0.9 \log_2 0.9 \right] \\
= 3.5 \cdot 0.469 = 1.642 \text{ bps}
\]

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

<table>
<thead>
<tr>
<th>Source symbol</th>
<th>Symbol probability ( P(\cdot) )</th>
<th>Code word</th>
<th>( l_i )</th>
<th>( P(\cdot)l_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>0.729</td>
<td>0</td>
<td>1</td>
<td>0.729</td>
</tr>
<tr>
<td>AAB</td>
<td>0.081</td>
<td>100</td>
<td>3</td>
<td>0.243</td>
</tr>
<tr>
<td>ABA</td>
<td>0.081</td>
<td>101</td>
<td>3</td>
<td>0.243</td>
</tr>
<tr>
<td>BAA</td>
<td>0.081</td>
<td>110</td>
<td>3</td>
<td>0.243</td>
</tr>
<tr>
<td>ABB</td>
<td>0.009</td>
<td>11100</td>
<td>5</td>
<td>0.045</td>
</tr>
<tr>
<td>BAB</td>
<td>0.009</td>
<td>11101</td>
<td>5</td>
<td>0.045</td>
</tr>
<tr>
<td>BBA</td>
<td>0.009</td>
<td>11110</td>
<td>5</td>
<td>0.045</td>
</tr>
<tr>
<td>BBB</td>
<td>0.001</td>
<td>11111</td>
<td>5</td>
<td>0.005</td>
</tr>
</tbody>
</table>
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.
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

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

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

$$2^{\frac{C_c}{B}} = e^{(\frac{C_c}{B}) \ln 2} \simeq 1 + \frac{C_c}{B} \ln 2$$
7.4. COMMUNICATIONS IN NOISY ENVIRONMENTS

- 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 graph]

- 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\textsuperscript{1}, where it is referred to as the power-bandwidth efficiency plane.

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

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

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

  errors,

  where $d_m$ is the minimum distance between two code words.
7.5. FORWARD ERROR CORRECTION CODING

- 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](image)

- 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

\[
\begin{array}{c}
\underbrace{a_1 \ a_2 \ \cdots \ a_k} & \underbrace{c_1 \ c_2 \ \cdots \ c_r} \\
\text{source symb.} & \text{parity check symb.}
\end{array}
\]

- Choose the \( r = n - k \) check symbols such that

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

or

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

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

\[
\mathbf{T} = [a_1 \ a_2 \ \cdots \ a_k \ c_1 \ \cdots \ c_r]^T
\]
In place of vector $T$, suppose $R = [R]$ is a received sequence (vector) of length $n$

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

- To decode $R$ and correct the error, we start by writing

$$R = T \oplus E,$$

where $E$ is a length $n$ error pattern induced by the channel

- We need to determine $E$ from $R$ using $H$

- Let

$$S = [S] = HR = HT \oplus HE = HE,$$

since $HT = 0$

- The matrix/vector $S$ is known as the *syndrome*

- We observe that $HE$, for the case of a single error, returns

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

and the $i$ th column of $H$ ($i$ th row of $E$) is where the error occurs
Example 7.1: A (6, 3) Code

• Given

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

• Assume that we receive 111011

• The syndrome is

\[ S = HR \]

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

• 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
<table>
<thead>
<tr>
<th>Code word length</th>
<th>$n = 2^m - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message block</td>
<td>$k = 2^m - 1 - m$</td>
</tr>
<tr>
<td>Parity-check block</td>
<td>$n - k = m$</td>
</tr>
<tr>
<td>Error correcting capability</td>
<td>$e = t = 1$</td>
</tr>
<tr>
<td>Minimum Hamming distance</td>
<td>$d_m = 3$</td>
</tr>
</tbody>
</table>

- The parity check matrix is of dimension $2^{n-k} - 1$ by $n - k$

- The $i$th column of the matrix $H$ 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

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

- Assume that we receive 1110001

- The syndrome is

\[
S = HR
\]

\[
= \begin{bmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 
\end{bmatrix} \begin{bmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
1 \\
\end{bmatrix} = \begin{bmatrix}
1 \\
1 \\
1 
\end{bmatrix}
\]
• 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 contain 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

- Cyclic block code generation
The decoder is a bit more complicated, but still easy to implement.

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

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

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.


---

*ECE 5630 Communication Systems II* 7-27
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 constraint 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.

\[
\begin{align*}
    v_1 &= S_1 \oplus S_2 \oplus S_3 \\
    v_2 &= S_1 \\
    v_3 &= S_1 \oplus S_2
\end{align*}
\]

- 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.
• 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.

Input Seq: 1010
Output Seq: 111 101 011 101
for zero initial initial states

Rate 1/3 constraint length $K = 3$ code tree
7.5. FORWARD ERROR CORRECTION CODING

- 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

<table>
<thead>
<tr>
<th>State</th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$C$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) State Transitions

<table>
<thead>
<tr>
<th>Previous $S_1$</th>
<th>Previous $S_2$</th>
<th>Input</th>
<th>Current $S_1$</th>
<th>Current $S_2$</th>
<th>Current $S_3$</th>
<th>State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$A$</td>
<td>(000)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$C$</td>
<td>(111)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$A$</td>
<td>(100)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$C$</td>
<td>(011)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$B$</td>
<td>(010)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$D$</td>
<td>(110)</td>
</tr>
</tbody>
</table>

(c) Encoder Output for State Transition $x 	o y$

<table>
<thead>
<tr>
<th>Transition</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A 	o A$</td>
<td>(000)</td>
</tr>
<tr>
<td>$A 	o C$</td>
<td>(111)</td>
</tr>
<tr>
<td>$B 	o A$</td>
<td>(100)</td>
</tr>
<tr>
<td>$B 	o C$</td>
<td>(011)</td>
</tr>
<tr>
<td>$C 	o B$</td>
<td>(101)</td>
</tr>
<tr>
<td>$C 	o D$</td>
<td>(010)</td>
</tr>
<tr>
<td>$D 	o B$</td>
<td>(001)</td>
</tr>
<tr>
<td>$D 	o D$</td>
<td>(110)</td>
</tr>
</tbody>
</table>

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.

![Trellis Diagram]

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 = 1101010111)

<table>
<thead>
<tr>
<th>Path</th>
<th>Corresponding symbols</th>
<th>Hamming distance</th>
<th>Survivor?</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAAA</td>
<td>0000000000</td>
<td>6</td>
<td>No</td>
</tr>
<tr>
<td>ACBA</td>
<td>1111011000</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>ACDB</td>
<td>111010001</td>
<td>5</td>
<td>Yes²</td>
</tr>
<tr>
<td>AACB</td>
<td>0001111001</td>
<td>5</td>
<td>No²</td>
</tr>
<tr>
<td>AAAC</td>
<td>000000111</td>
<td>5</td>
<td>No</td>
</tr>
<tr>
<td>ACBC</td>
<td>1111010111</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>ACDD</td>
<td>1110101110</td>
<td>6</td>
<td>No</td>
</tr>
<tr>
<td>AACD</td>
<td>0001110100</td>
<td>4</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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

²If 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)

<table>
<thead>
<tr>
<th>Path</th>
<th>Previous survivor's distance</th>
<th>New segment</th>
<th>Added distance</th>
<th>New distance</th>
<th>Survivor?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACBAA</td>
<td>4</td>
<td>AA</td>
<td>3</td>
<td>7</td>
<td>Yes</td>
</tr>
<tr>
<td>ACDBA</td>
<td>5</td>
<td>BA</td>
<td>2</td>
<td>7</td>
<td>No</td>
</tr>
<tr>
<td>ACBCB</td>
<td>1</td>
<td>CB</td>
<td>1</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>AACDB</td>
<td>4</td>
<td>DB</td>
<td>2</td>
<td>6</td>
<td>No</td>
</tr>
<tr>
<td>ACBAC</td>
<td>4</td>
<td>AC</td>
<td>0</td>
<td>4</td>
<td>Yes</td>
</tr>
<tr>
<td>ACDBC</td>
<td>5</td>
<td>BC</td>
<td>1</td>
<td>6</td>
<td>No</td>
</tr>
<tr>
<td>ACBCD</td>
<td>1</td>
<td>CD</td>
<td>2</td>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>AACCD</td>
<td>4</td>
<td>DD</td>
<td>1</td>
<td>5</td>
<td>No</td>
</tr>
</tbody>
</table>

¹An underscore indicates the previous survivor.
Performance Comparisons

- The BEP performance of a convolutional code can be approximated using the weight structure/spectrum bound

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

where $d_{\text{free}}$ 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

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

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

$$p = Q \left( \sqrt{\frac{2k R E_b}{N_0}} \right)$$

where $R$ is the code rate

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

---

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

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

  - $P_b$ vs $E_b/N_0$ dB
  - $E_b/N_0 = 3.5$ dB for $K = 3$
  - $E_b/N_0 = 4$ dB for $K = 5$

  - BPSK uncoded
  - Convolved, coded; hard decoder; rate $= 0.5$
  - Convolved, coded; hard decoder; rate $= 0.333$
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\textsuperscript{4}, but were re-discovered by MacKay and Neal in 1996\textsuperscript{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\textsuperscript{6}.

- \textit{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 \textit{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.


7.5. FORWARD ERROR CORRECTION CODING

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

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

- 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

---

- 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:

7.5.5 Turbo Codes

See Z&T pages 681-683.
### 7.5.6 MATLAB Support for FEC Coding

#### Error-Control Coding

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

**MATLAB Comm toolbox™FEC coding functions**
Example 7.5: Rate 1/2 $K = 7$ Convolutional Code Simulation

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

```matlab
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
decimals = 0;
N_RecBits = 0;
while decimals < 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
    decimals = decimals + sum(xor(msg(1:end-42), decoded(42+1:end)));
    N_RecBits = N_RecBits + length(msg(1:end-42));
end

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

- Collect BEP simulation data:

    >> [Pe,decimals,N_RecBits] = convCoder_test(0,2500);
    SNR = 0.00, decimals = 744413, BEP = 3.72e-01
    >> [Pe,decimals,N_RecBits] = convCoder_test(1,2500);
    SNR = 1.00, decimals = 508620, BEP = 2.54e-01
    >> [Pe,decimals,N_RecBits] = convCoder_test(2,2500);
    SNR = 2.00, decimals = 237292, BEP = 1.19e-01
    >> [Pe,decimals,N_RecBits] = convCoder_test(3,2500);
7.5. FORWARD ERROR CORRECTION CODING

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

```matlab
function Pb = conv_Pb_bound(R,dfree,Ck,SNRdB,hard_soft)
    % Pb = conv_Pb_bound(R,dfree,Ck,SNRdB,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 + ...
    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.