Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Comunicações

UNICAMP

FAPESP - São Paulo Coding School

Introduction to Convolutional Codes

Reginaldo Palazzo Jr.

January 2015

A Brief Review of Convolutional Codes

- Convolutional codes were introduced by Peter Elias in 1955. The **main motivation** was **decoding with high reliability at the expense of computation and delay**.
- In 1967 Viterbi proposed an **ML decoding algorithm** with reasonable complexity known as the **Viterbi Algorithm**. Another trellis based decoder is the **BCJR Algorithm** a **MAP decoding algorithm** proposed by Bahl, Cocke, Jelinek, and Raviv in 1974.
- A non-recursive encoder **does not admit a feedback structure** whereas a recursive encoder **does admit a feedback structure**.
- A **non-recursive non-systematic convolutional code** may be considered a **finite impulse response filter (FIR)**, whereas a **recursive convolutional code** may be considered an **infinite impulse response filter (IIR)**.
- **Recursive systematic convolutional codes** were proposed by Berrou et. al. in 1991, and are useful in **concatenated codes such as turbo codes**.

Several algorithms exist for **decoding convolutional codes**:

- For **short constraint length codes**, the **Viterbi algorithm** is decoding used for it provides maximum likelihood performance.
- **Longer constraint length codes** are "easily" decoded with any of the **sequential decoding algorithms**, for instance the **Fano algorithm**. Unlike Viterbi decoding, **sequential decoding is not maximum likelihood** but its complexity increases only slightly with constraint length, allowing the use of long-constraint-length codes.
- Both **Viterbi algorithm** and **Sequential decoding algorithms** return **hard decisions**: the bits that form the most likely codeword. When a confidence measure can be added to each bit then the **Soft Output Viterbi Algorithm** (SOVA) is employed. **Maximum A Posteriori (MAP) soft decisions** for each bit can be obtained by use of the **BCJR algorithm**.

Non-recursive (feed-forward) encoders and Recursive (feedback) encoders:

- Recursive systematic convolutional encoder is used as constituent code in a Turbo code.
- Recursive systematic convolutional (RSC) codes have been used in Turbo codes.
- Other RSC codes and example applications include:
  - LDPC code implementation and as inner constituent code for serial concatenated convolutional codes.
  - Serial concatenated convolutional codes and multidimensional turbo codes.
  - Constituent code in low error rate turbo codes for applications in satellite links.

In control applications, two types are distinguished:

- **Moore Machine**: The FSM uses only entry actions, i.e., the **output depends only on the state**.
- **Mealy Machine**: The FSM uses only input actions, i.e., the **output depends on the input and state**. The use of a Mealy FSM leads often to a reduction of the number of states.

Chapter 1

Channel Coding Theorems for Block and Convolutional Codes[a]

2. Objectives
3. Digital Communication System Model
4. Channel Encoder
5. Channel Decoder
6. Error Probability of a Code with two Codewords
7. Channel Coding Theorems

[a]Most of the content of this chapter may be found in the classical papers and books by Shannon, Viterbi/Omura, Gallager, Shu Lin/Costello, etc.

1 Objectives

- Review some of the concepts and techniques employed in **finding the upperbounds on the average error probability** of a set of codes by use of the **random coding argument** as proposed by Shannon.

- To show that the **reliability function, $E_c(R)$, for time-varying convolutional codes** is significantly superior to the **reliability function, $E_b(R)$, for block codes**, for all R, where R is the transmission rate and $0 < R < C$.

2 Digital Communication System Model

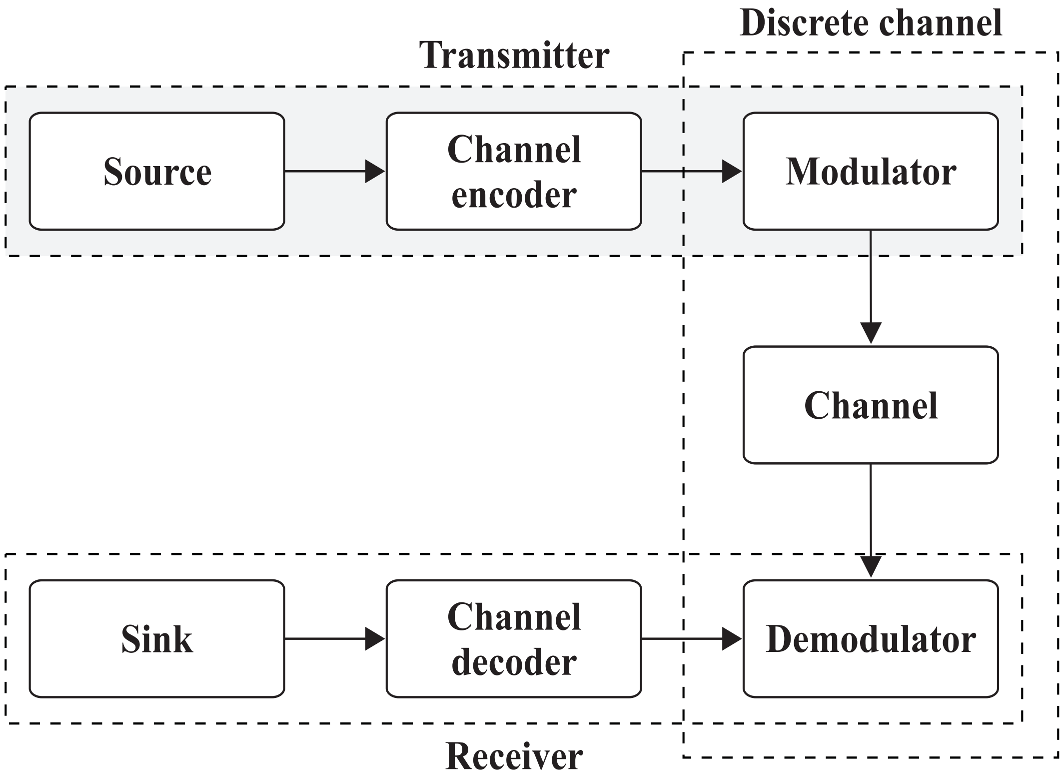
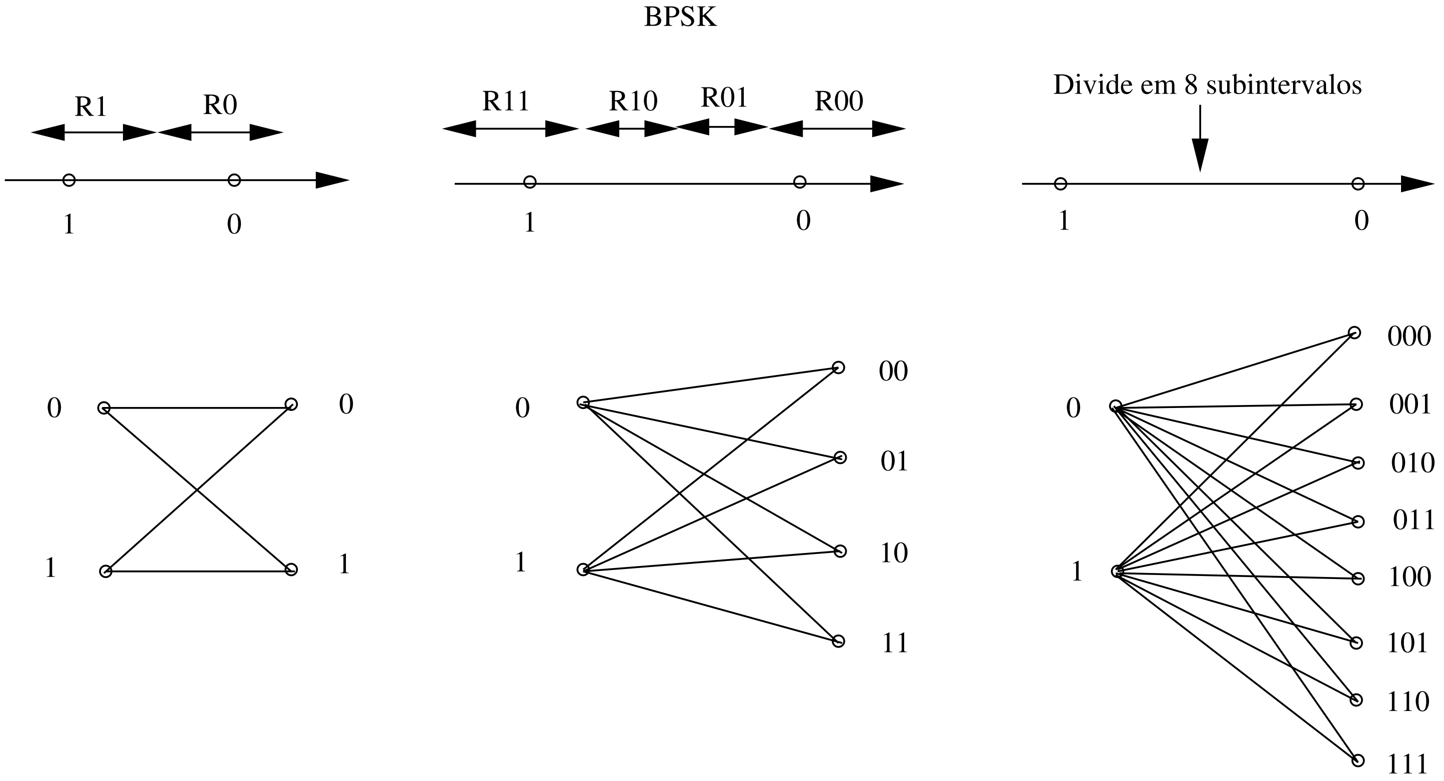
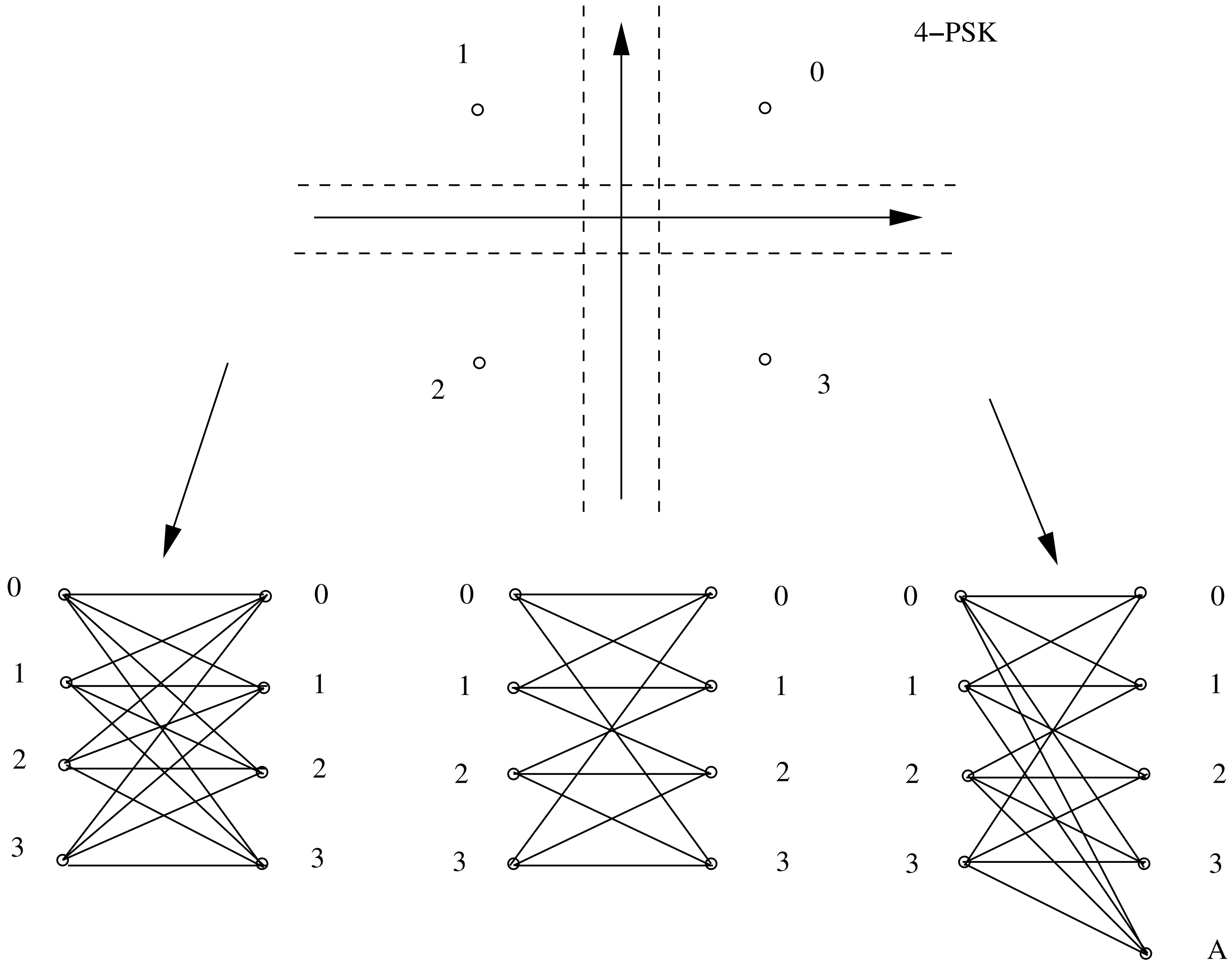


Figure 1: Digital communication system

2.1 Discrete Channels from 1-D signal sets



2.2 Discrete Channels from 2-D signal sets



Discrete channel - BSC channel capacity:

$$\mathcal{C} = 1 - H(p)$$

Continuous channel - AWGN channel capacity:

$$\mathcal{C} = W \ln\left(1 + \frac{P}{N}\right)$$

- Q: Is the **optimum Shannon code** a **random code**?
- A: The **optimum Shannon code** is a **map**.

- The **multi-h codes** and **Ungerboeck codes** have shown the existence of **structured optimum codes** achieving the Shannon bound.

- **Structured optimum discrete codes**: Turbo codes and LDPC codes.
- **Structured optimum continuous codes**: Slepian codes and Lattices codes.

2.3 Shannon bound

The signal power is $P = E_b R$ and the noise power is $N = W N_0$. From the AWGN channel capacity

$$\frac{C}{W} = \log\left(1 + \frac{RE_b}{N_0 W}\right)$$

with the constraint,

$$\frac{R}{W} < \frac{C}{W}.$$

Define **spectral bit rate** r (bits/seg/Hz) as $r = \frac{R}{W}$, yields

$$r < \log\left(1 + r\frac{E_b}{N_0}\right),$$

or

$$\frac{E_b}{N_0} > \frac{2^r - 1}{r}.$$

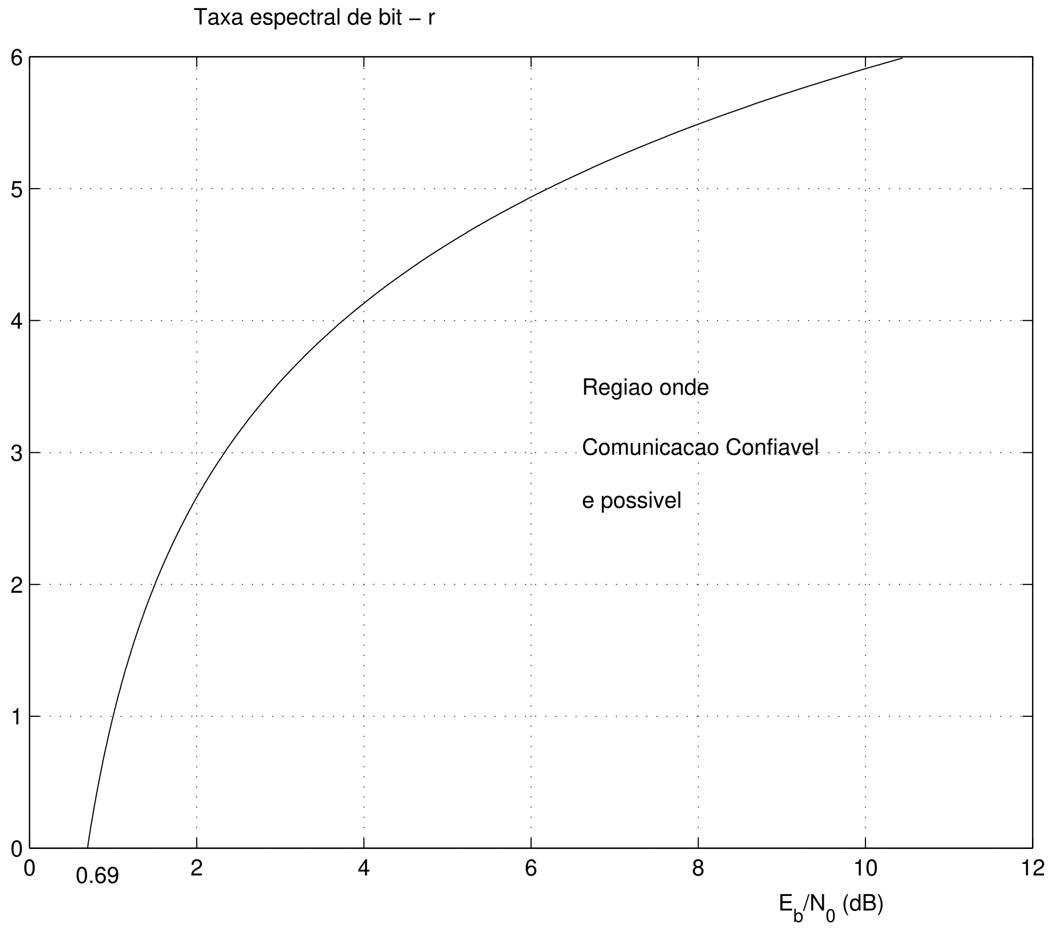


Figure 2: Baseband channel capacity with AWGN

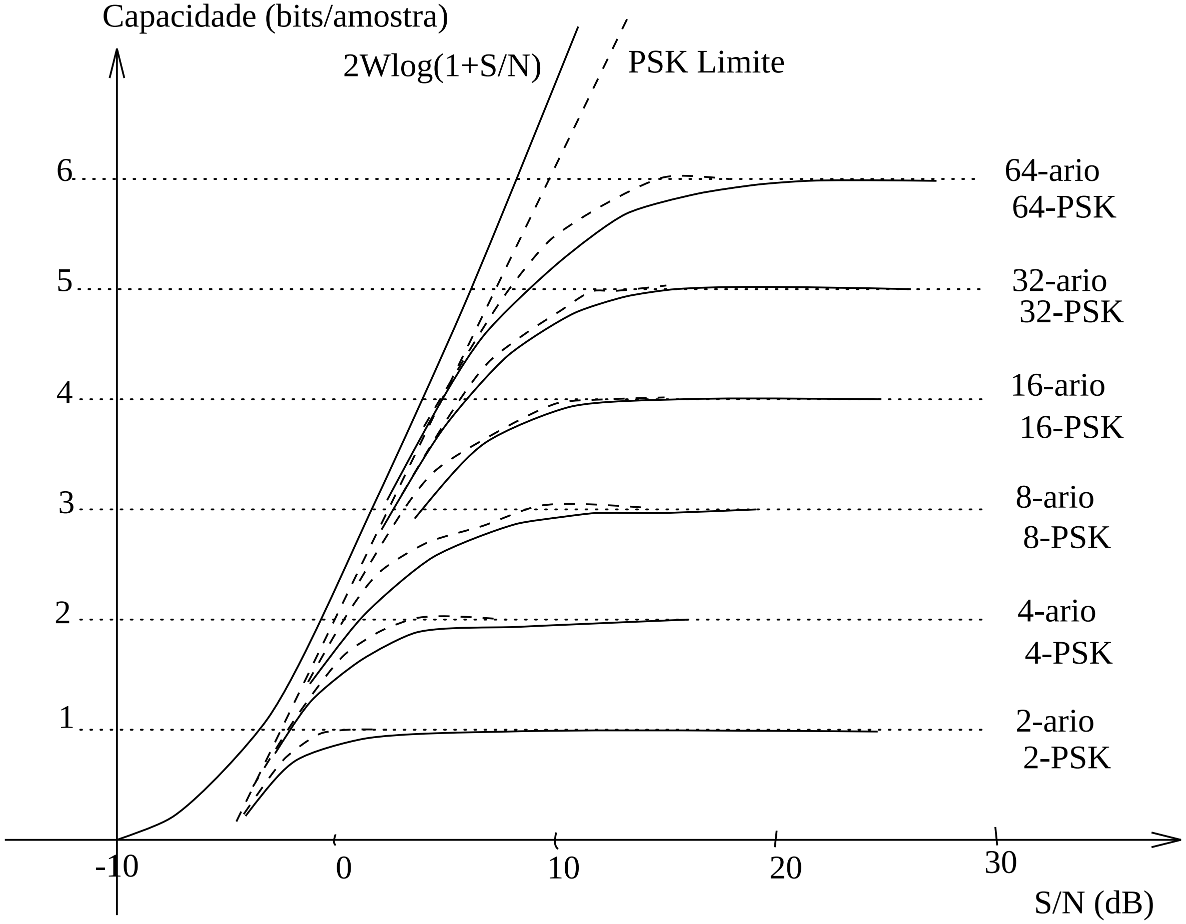


Figure 3: Channel capacities of 1-D and 2-D signal sets

3 Channel Encoder

The **mapping must be one-to-one** between the **set of input sequences** to the channel encoder and the **set of codewords** at the output of the channel encoder.

Input	$\longrightarrow$	Channel Encoder	$\longrightarrow$	Codewords
blocklength L				blocklength N
1		$\leftrightarrow$		$\mathbf{x}_1 = (x_{11}, x_{12}, x_{13}, \ldots, x_{1N})$
2		$\leftrightarrow$		$\mathbf{x}_2 = (x_{21}, x_{22}, x_{23}, \ldots, x_{2N})$
.	.	.	.	.
.	.	.	.	.
$M = 2^L$		$\leftrightarrow$		$\mathbf{x}_M = (x_{M1}, x_{M2}, x_{M3}, \ldots, x_{MN})$

Define R as $R = \frac{\log M}{N} = \frac{\log 2^L}{N} = \frac{L}{N} \log 2$ bits/channel use or nats/channel use.

4 Channel Decoder

Since the mapping is one-to-one, the channel is characterized as shown in Fig. 4(b). As a consequence, the decoder is as shown in Fig. 4(b).

(a) Channel (b) Channel decoder

Figure 4: Channel and channel decoder

The objective of the channel decoder is to "guess" which $\mathbf{x}_m$, $1 \le m \le M$, was transmitted.

If $\mathbf{x}_? = \mathbf{x}_m$, given that $\mathbf{x}_m$ was transmitted, then the decoding was correct. Otherwise, a decoding error has occurred, with probability P_e.

If we fix R, such that $R < C$, and increase N, then P_e decreases exponentially with N.

This implies that we may find codes with N large and R fixed, such that $P_e \to 0$, or to a number as small as we want. To achieve this result the number of codewords may have to increase in order to keep $\frac{\log M}{N}$ constant.

This large number of codewords implies an increase in the decoding complexity. In general, the complexity grows exponentially with the number of codewords.

4.1 Decoding rules for block codes

Decoding Rule: Minimum P_e.

Let $p(\mathbf{y}|\mathbf{x}_m)$ be the probability of receiving a sequence $\mathbf{y}$ (channel output) given that the **message** or **codeword** $\mathbf{x}_m$ was transmitted. Assume the channel is a DMC, that is,

$$p(\mathbf{y}|\mathbf{x}_m) = \prod_{i=1}^{N} p(y_i|x_{mi}).$$

Let $p(m)$ be the probability of **message** m. Then,

$$p(m|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x}_m)p(m)}{p(\mathbf{y})} = \frac{p(\mathbf{y}|\mathbf{x}_m)p(m)}{\sum_m p(m)p(\mathbf{y}|\mathbf{x}_m)}$$

If $\mathbf{y}$ is decoded as m, then $p(m|\mathbf{y})$ is maximum and $(1 - p(m|\mathbf{y}))$ is minimum. Hence, decode the received sequence $\mathbf{y}$ as m such that

$$p(m|\mathbf{y}) \geq p(m'|\mathbf{y}), \quad \forall \quad m' \neq m. \tag{1}$$

Now, equation (1) is equal to

$$\frac{p(\mathbf{y}|m)p(m)}{p(\mathbf{y})} \geq \frac{p(\mathbf{y}|m')p(m')}{p(\mathbf{y})}.$$

Since $p(\mathbf{y}) > 0$ is independent of m, it follows that

1) **Maximum A Posteriori Probability (MAP):** If at least one $p(m) \neq \frac{1}{M}$ for $1 \leq m \leq M$, then

$$p(\mathbf{y}|m)p(m) \geq p(\mathbf{y}|m')p(m'), \quad \forall \quad m' \neq m$$

2) **Maximum Likelihood (ML) Decoding:** If $p(m) = 1/M$, $1 \leq m \leq M$, then

$$p(\mathbf{y}|m) \geq p(\mathbf{y}|m'), \quad \forall \quad m' \neq m$$

A **decoding rule** is defined as a **mapping** of the set $\mathbf{Y}^N$ of the channel output sequence to the set of the M messages. Thus, the **decoding rule** establishes a **partitioning** of the N-dimensional space in M regions.

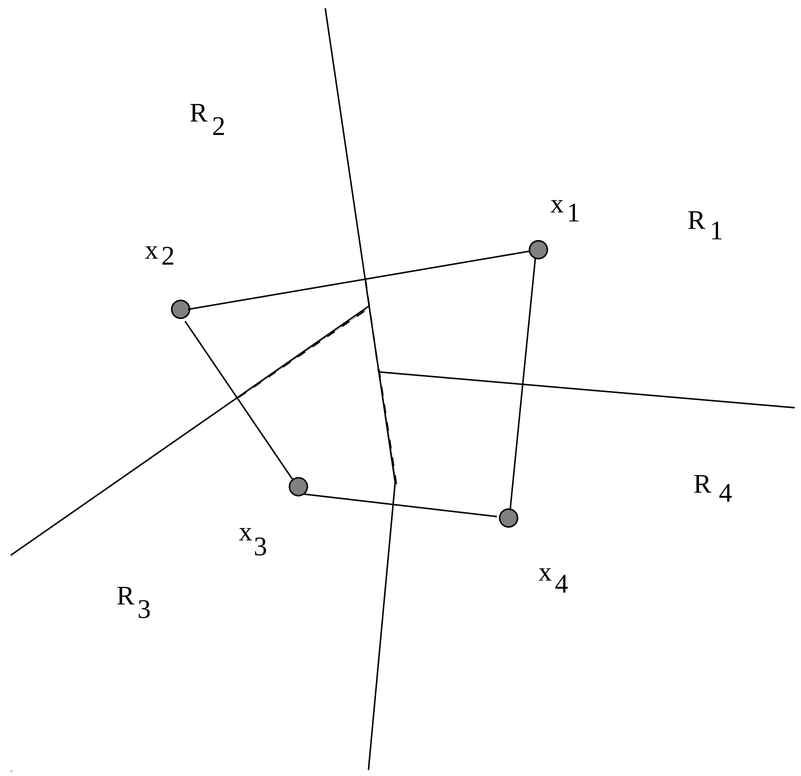


Figure 5: 2-D space partitioned into four regions

Let $\mathbf{Y}_m$ be the set representing all the decoded messages as message m.

Then, $\mathbf{Y}_m^c$ is the complementary set of $\mathbf{Y}_m$. Thus, if m is the output of the source encoder, $\mathbf{x}_m$ will be the transmitted codeword.

Then, an error will occur if the received sequence $\mathbf{y}$ belongs to the set $\mathbf{Y}_m^c$.

The decoding error probability given that message m was transmitted is

$$P_{e,m} = \sum_{\mathbf{y} \in \mathbf{Y}_m^c} p(\mathbf{y}|\mathbf{x}_m).$$

The average error probability is

$$P_e = \sum_m p(m) P_{e,m}.$$

5 Error Probability of a Code with two Codewords

Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be codewords of length N, and the decoding rule is ML, that is,

$$p(\mathbf{y}|\mathbf{x}_1) \geq p(\mathbf{y}|\mathbf{x}_2) \rightarrow \text{decode message 1 correctly,}$$

or equivalently,

$$p(\mathbf{y}|\mathbf{x}_2) \geq p(\mathbf{y}|\mathbf{x}_1) \rightarrow \text{decode message 2 correctly.}$$

The conditional error probability given message $\mathbf{x}_1$ is

$$P_{e,1} = \sum_{\mathbf{y} \in \mathbf{Y}_1^c} p(\mathbf{y}|\mathbf{x}_1). \tag{2}$$

Fig. 6, shows a possible representation of the code with two codewords.

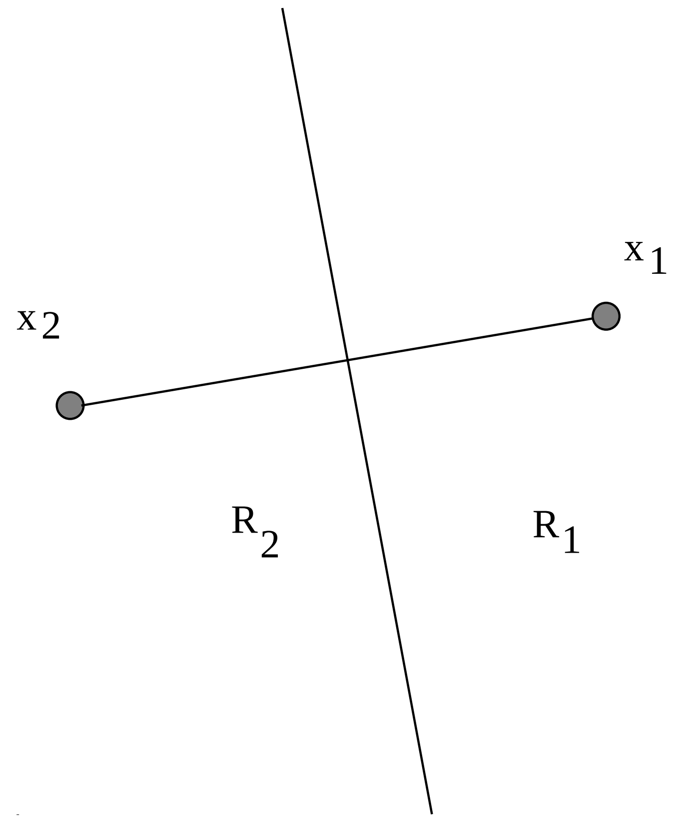


Figure 6: Representation of the code with two codewords

For $\mathbf{y} \in \mathbf{Y}_1^c$ we may upperbound $p(\mathbf{y}|\mathbf{x}_1)$ by

$$p(\mathbf{y}|\mathbf{x}_1) \leq (p(\mathbf{y}|\mathbf{x}_1))^{1-s} (p(\mathbf{y}|\mathbf{x}_2))^s, \qquad 0 < s < 1, \tag{3}$$

Since $\mathbf{y} \in \mathbf{Y}_1^c$ the decision will be in favor of $\mathbf{x}_2$, or $p(\mathbf{y}|\mathbf{x}_2) \geq p(\mathbf{y}|\mathbf{x}_1)$. By taking both sides of the inequality to the power s, with $0 < s < 1$, we have

$$\left[\frac{p(\mathbf{y}|\mathbf{x}_2)}{p(\mathbf{y}|\mathbf{x}_1)}\right]^s \geq 1.$$

Multiplying both sides by $p(\mathbf{y}/\mathbf{x}_1)$, leads to (3). Substituting (3) in

$$P_{e,1} = \sum_{\mathbf{y} \in \mathbf{Y}_1^c} p(\mathbf{y}|\mathbf{x}_1),$$

yields

$$P_{e,1} \leq \sum_{\mathbf{y} \in \mathbf{Y}_1^c} (p(\mathbf{y}|\mathbf{x}_1))^{1-s} (p(\mathbf{y}|\mathbf{x}_2))^s. \tag{4}$$

As $\sum_{\mathbf{y}\in\mathbf{Y}_1^c}(p(\mathbf{y}|\mathbf{x}_1))^{1-s}(p(\mathbf{y}|\mathbf{x}_2))^s \geq \mathbf{0}$, then

$$\sum_{\mathbf{y}}(p(\mathbf{y}|\mathbf{x}_1))^{1-s}(p(\mathbf{y}|\mathbf{x}_2))^s = \sum_{\mathbf{y}\in\mathbf{Y}_1^c}(p(\mathbf{y}|\mathbf{x}_1))^{1-s}(p(\mathbf{y}|\mathbf{x}_2))^s + \sum_{\mathbf{y}\in\mathbf{Y}_1}(p(\mathbf{y}|\mathbf{x}_1))^{1-s}(p(\mathbf{y}|\mathbf{x}_2))^s. \tag{5}$$

Note that for the first summation on the right-hand side of equation (5) to be equal to the left-hand side it is necessary that a positive number be added to it. Consequently,

$$\sum_{\mathbf{y}\in Y^c} p(\mathbf{y}|x_1)^{1-s}p(\mathbf{y}|x_2)^s \leq \sum_{\mathbf{y}\in Y^c} p(\mathbf{y}|x_1)^{1-s}p(\mathbf{y}|x_2)^s. \tag{6}$$

By using this inequality in (2), we have

$$P_{e,1} \leq \sum_{\mathbf{y}} (p(\mathbf{y}|\mathbf{x}_1))^{1-s} (p(\mathbf{y}|\mathbf{x}_2))^{s}, \qquad 0 < s < 1.$$

Following the same procedure to determine $P_{e,2}$, we arrive at

$$P_{e,2} \leq \sum_{\mathbf{y}} (p(\mathbf{y}|\mathbf{x}_2))^{1-r} (p(\mathbf{y}|\mathbf{x}_1))^{r}, \qquad 0 < r < 1. \tag{7}$$

If we consider $1-s=r$, then substituting this term in (7), we conclude that

$$P_{e,2} \leq \sum_{\mathbf{y}} (p(\mathbf{y}|\mathbf{x}_1))^{1-s} (p(\mathbf{y}|\mathbf{x}_2))^{s}, \qquad 0 < s < 1,$$

that is, the upperbound is equal to $P_{e,1}$ as well as to $P_{e,2}$. In this way, if the channel is a DMC, then

$$P_{e,m} \leq \sum_{y_1}\sum_{y_2}\cdots\sum_{y_N}\left\{\prod_{i=1}^{N}\left(p(y_i|x_{1i})\right)^{1-s}\left(p(y_i|x_{2i})\right)^{s}\right\}$$

$$= \left\{\sum_{y_1}\left(p(y_1|x_{11})\right)^{1-s}\left(p(y_1|x_{21})\right)^{s}\right\}\cdot\left\{\sum_{y_N}\left(p(y_N|x_{1N})\right)^{1-s}\left(p(y_N|x_{2N})\right)^{s}\right\}.$$

$$P_{e,m} \leq \prod_{n=1}^{N}\left\{\sum_{y_n}\left(p(y_n|x_{1n})\right)^{1-s}\left(p(y_n|x_{2n})\right)^{s}\right\}, \qquad m=1,2$$

Define $g_n(s) = \sum_{y_n}\left(p(y_n|x_{1n})\right)^{1-s}\left(p(y_n|x_{2n})\right)^{s}$, then

$$P_{e,m} \leq \prod_{n=1}^{N} g_n(s), \qquad m=1,2, \qquad 0<s<1.$$

For certain channels $p(y_n|x_{1n})$ and/or $p(y_n|x_{2n})$ may be zero implying that $g_n(s)$ is undetermined when $s = 0$ or $s = 1$. In these cases we define

$$g_n(0) = \lim_{s \to 0_+} g_n(s) = \sum_{y_n \text{ for which } p(y_n|x_{2n}) \neq 0} p(y_n|x_{1n}),$$

and

$$g_n(1) = \lim_{s \to 1^-} g_n(s) = \sum_{y_n \text{ for which } p(y_n|x_{1n}) \neq 0} p(y_n|x_{2n}).$$

Therefore, the least upperbound is given by

$$P_{e,m} \leq \min_{0 \leq s \leq 1} \left\{ \prod_{n=1}^{N} g_n(s) \right\}, \qquad m = 1, 2.$$

Example 1. *Consider the BSC channel as shown in Fig. 7. The codewords are given, respectively, by* $\mathbf{x}_1 = (0,0,...,0)$ *and* $\mathbf{x}_2 = (1,1,...,1)$.

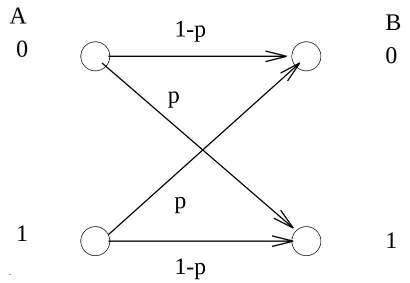


Figure 7: BSC channel

The function $g_n(s)$ *is given by* $g_n(s) = \sum_{y_n} (p(y_n|x_{1n}))^{1-s} (p(y_n|x_{2n}))^s$, $1 \le n \le N$. *Hence,*

$$g_n(s) = p(0|0)^{1-s} p(0|1)^s + p(1|0)^{1-s} p(1|1)^s = (1-p)^{1-s} p^s + p^{1-s}(1-p)^s.$$

If $s = 1/2$, *then* $g_n(1/2) = \sqrt{4p(1-p)}$. *Consequently,*

$$P_{e,m} \le \left(\sqrt{4p(1-p)}\right)^N, \qquad m = 1,2$$

Note that the upperbound on $P_{e,m}$ *is achieved if, and only if, the codewords differ in all of the N positions.*

6 Channel Coding Theorems

6.1 Block Codes

$$\alpha \overbrace{\exp\{-N[E_{sp}(R)]\}}^{\text{sphere packing}} \quad \leq \quad \overline{P}_b \quad \leq \quad \beta \overbrace{\exp\{-N[E_b(R)]\}}^{\text{upperbound}}$$

where

$$E_{sp}(R) = \sup_{\rho \geq 0} \left\{ \max_Q \{E_0(\rho, Q)\} - \rho R \right\} \quad \text{and} \quad E_b(R) = \max_{0 \leq \rho \leq 1} \left\{ \max_Q \; \{E_0(\rho, Q) - \rho R\} \right\}$$

$$E_0(\rho, Q) = -\ln \left\{ \sum_{j=0}^{J-1} \left(\sum_{k=0} Q(k)[P(j|k)]^{\frac{1}{1+\rho}} \right)^{1+\rho} \right\}; \quad R_0 = E_0(1, Q)$$

From this theorem we have

- N increases $\rightarrow$ dimension grows $\rightarrow$ spheres generate a surface $\rightarrow$ errors decrease;
- Well packed spheres $\rightarrow$ R increases up to channel capacity C.
- This **sphere packing** is called **Code**.

6.2 Convolutional Codes

$$\overline{P}_b < (2^k - 1)\frac{2^{KkE_c(R)/R}}{\left(1 - 2^{\varepsilon k E_c(R)/R}\right)^2}, \qquad \varepsilon > 0$$

where

$$E_c(R) = \begin{cases} R_0, & 0 \le R \le R_0(1-\varepsilon) \\ \\ \max_Q \{E_0(\rho, Q)\}, & 0 \le \rho \le 1 \end{cases}$$

$$R = (1-\varepsilon)\left\{\max_Q \{E_0(\rho, Q)/\rho\}\right\}, \quad R_0(1-\varepsilon) \le R \le C(1-\varepsilon)$$

where ε is a positive number arbitrarily small.

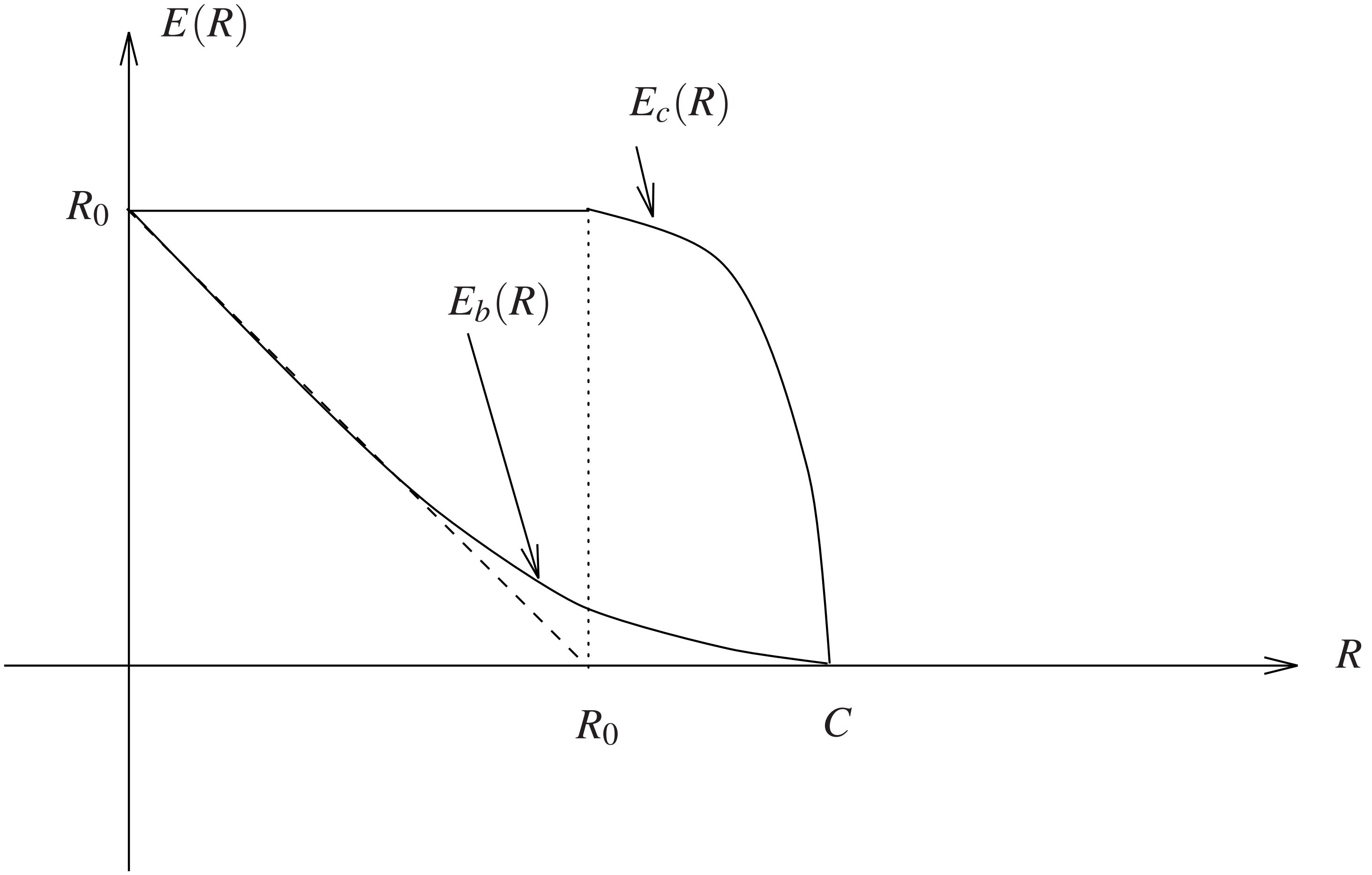


Figure 8: Reliability function versus Transmission rate

Chapter 2

Topological Structure of Convolutional Codes[a].

8. Objectives
9. Cutoff Rate of Codes with two Codewords
10. Tree Codes
11. Trellis Codes
12. Convolutional Codes
13. Distance Properties of Convolutional Codes

---

[a]Most of the content of this chapter may be found in the CISM Lecture Notes, No. 216, by Professor James L. Massey

7 Objectives

- To **describe the topological structure** of convolutional codes.
- To **characterize the encoder**, a FSM, as a **trellis** structure (the number of nodes from a certain depth in the tree converges to the number of state).
- The **decoding problem** as **finding the shortest path in this trellis**. The **Viterbi algorithm** as an **efficient and exhaustive search procedure** derived from the topology associated with the code. The **sequential decoding algorithms** may be seen as **heuristic search procedures** with the aim of reducing the average number of computations.
- To **emphasize the topological structure** rather than the **algebraic structure** of the convolutional codes we consider the tree, trellis and convolutional codes.

8 Cutoff Rate of Codes with two Codewords

A DMC channel has finite input and output alphabets and it acts independently on each input symbol. Another characteristic is that its statistics is time invariant.

Let $\mathcal{A}$ and $\mathcal{B}$ be the input and output alphabets of a DMC channel, respectively, that is, $\mathcal{A}$ and $\mathcal{B}$ are sets consisting of M and N discrete values,

$$\mathcal{A} = \{a_1, a_2, ..., a_M\} \quad \text{and} \quad \mathcal{B} = \{b_1, b_2, ..., b_N\}.$$

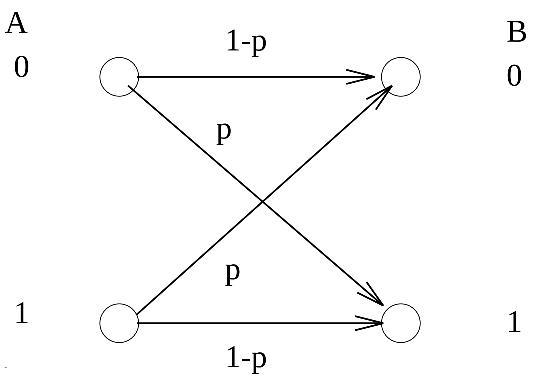
The channel is completely characterized once the conditional probabilities $P(b_j/a_i)$, for every $j = 1, 2, ..., N$, and $i = 1, 2, ..., M$. These conditional probabilities are known as **transition probabilities**.

Example 2. *Consider the binary symmetric channel, BSC, identified by the input alphabet* $\mathcal{A} = \{0, 1\}$*, the output alphabet* $\mathcal{B} = \{0, 1\}$*, and by the transition probabilities* $P(b_j/a_i)$*, for* $a_i, b_j \in \{0, 1\}$*, as shown in Fig. 9(a).*

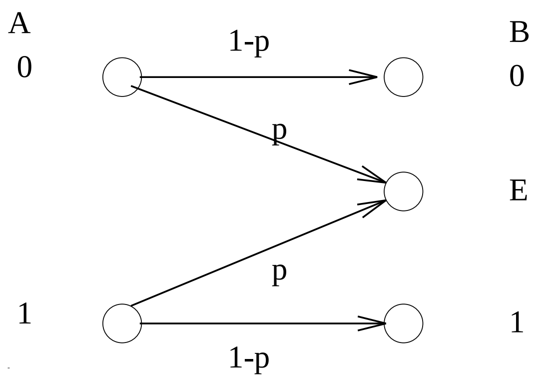
$$P(0|0) = P(1|1) = 1 - p, \qquad P(0|1) = P(1|0) = p$$

As another example, consider the binary erasure channel, BEC, identified by the input alphabet $\mathcal{A} = \{0, 1\}$*, the output alphabet* $\mathcal{B} = \{0, 1, E\}$*, and by the transition probabilities* $P(b_j/a_i)$*, for* $a_i \in \{0, 1\}$ *and* $b_j \in \{0, E, 1\}$*, as shown in Fig. 9(b).*

$$P(0|0) = P(1|1) = 1 - p, \qquad P(E|0) = P(E|1) = p$$



(a) BSC channel



(b) BEC channel

Figure 9: Binary input channels

Let $\mathcal{C} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$ be an (N,R) block code for a DMC channel with cardinality $m = 2^{NR}$, where $\mathbf{x}_j = (x_{j1}, x_{j2}, ..., x_{jN})$, denotes a codeword for $1 \leq j \leq m$, with $x_{jk} \in \mathcal{A}$ and $1 \leq k \leq N$.

ML decoder: if $\mathbf{y} = (y_1, y_2, y_3, \cdots, y_N)$ is the received vector, the decoder selects an estimate of the subscript, say j, of the transmitted codeword that maximizes

$$P(\mathbf{y}/\mathbf{x}_j) = \prod_{n=1}^{N} P(y_n | x_{jn}).$$

Consider the code has only two codewords, $\mathcal{C} = \{\mathbf{x}_1, \mathbf{x}_2\}$, and the parameters are $(N, R = 1/N)$.

Let $Q(.)$ be the probability distribution of the input alphabet $\mathcal{A}$ of the DMC channel, that is, the probability of the symbol a_k is $Q(a_k)$, for $1 \leq k \leq 2$.

For each code $\{\mathbf{x}_1, \mathbf{x}_2\}$ we specify the joint probability of $\{\mathbf{x}_1, \mathbf{x}_2\}$ by

$$P(\mathbf{x}_1, \mathbf{x}_2) = \prod_{n=1}^{N} Q(x_{1n})Q(x_{2n}),$$

whose interpretation is the following: $P(\mathbf{x}_1, \mathbf{x}_2)$ denotes the probability of a code $\{\mathbf{x}_1, \mathbf{x}_2\}$ where each symbol of the codewords is independently selected with respect to the probability distribution $Q(.)$.

The set of codes $\mathbb{C} = \{\mathcal{C}_1, \mathcal{C}_2, ...\mathcal{C}_W\}$, where $W = 2^{2N}$ denotes the number of possible codes with two codewords each with length N, with the corresponding set of joint probabilities $\mathbb{P} = \{P_{\mathcal{C}_1}, P_{\mathcal{C}_2}, ..., P_{\mathcal{C}_W}\}$ form a **set with a measure**, or equivalently, an **ensemble of codes**, that is,

$$\mathbb{M} = \{(\mathcal{C}_1; P_{\mathcal{C}_1}), (\mathcal{C}_2; P_{\mathcal{C}_2}), ..., (\mathcal{C}_W; P_{\mathcal{C}_W})\}.$$

Let $P_e(\mathbf{x}_1, \mathbf{x}_2)$ denote the decoding error probability when the decoder employs the ML decision rule for the code $\{\mathbf{x}_1, \mathbf{x}_2\}$.

Thus, $\overline{P}_e$ for the ensemble of codes with $m = 2$ codewords each having length N, is given by

$$\overline{P}_e = \sum_{\mathbf{x}_1 \in \mathcal{A}^N} \sum_{\mathbf{x}_2 \in \mathcal{A}^N} P_e(\mathbf{x}_1, \mathbf{x}_2) P(\mathbf{x}_1, \mathbf{x}_2).$$

From the independence of the joint probability distribution $P(.,.)$ associated with the code $\{\mathbf{x}_1, \mathbf{x}_2\}$. From Chapter 1, it can be shown that $\overline{P}_e$ of the ensemble of codes is upperbounded by

$$\overline{P}_e < 2^{-NR_0}, \tag{8}$$

where the cut-off rate, R_0, is

$$R_0 = -\log \left\{ \min_{Q} \sum_{y \in \mathcal{B}} \left\{ \sum_{x \in \mathcal{A}} Q(x) \sqrt{P(y|x)} \right\}^2 \right\}, \tag{9}$$

and $Q(.)$ is the probability distribution that minimizes R_0.

The interpretation of equation (8) is **there exists at least one code whose error probability $P_e(\mathbf{x}_1, \mathbf{x}_2)$ is not worse than the $\overline{P}_e$**.

Therefore, it is possible to find a code whose decoding error probability using the ML decision criterion is less than 2^{-NR_0}, for N large. **The cutoff rate R_0 is an important parameter for it is the rate at which the decoder operates**.

In the next sections, the **tree and trellis codes** are **characterized topologically** whereas the **convolutional codes** are **characterized algebraically**, via sequential linear systems.

We derive an upperbound on $\overline{P}_e$ for the ensemble of these codes by using the "**random coding**" **argument**. It will be noticed that the cutoff rate R_0 comes naturally into the derivation and determination of the upperbounds on $\overline{P}_e$ when ML decoding is used.

9 Tree Codes

Definition 1. *Let (D,M)-μ-ary be a tree such that: 1) μ branches leave each node up to depth D from the root node of the tree; 2) only one branch leaves each node from depth $\mathcal{D}$ such that $D \leq \mathcal{D} \leq D+M$ with D, M, and μ positive integers such that $D \geq 1$, $M \geq 0$, and $\mu \geq 2$.*

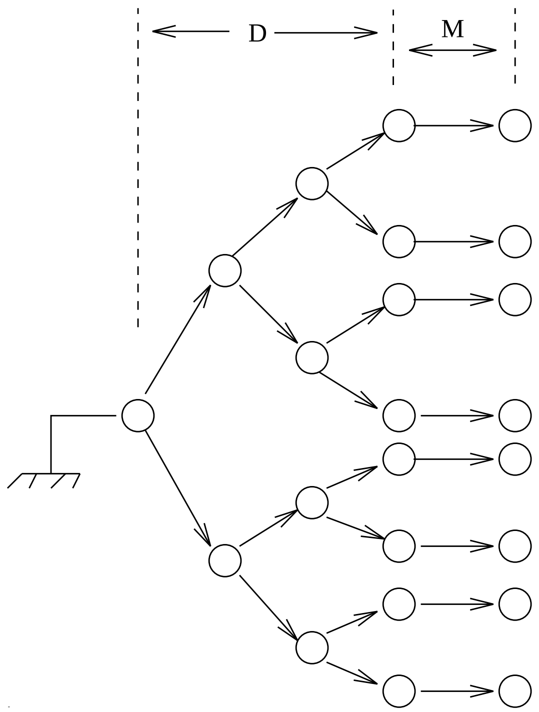


Figure 10: $A = (3,1)$-binary tree

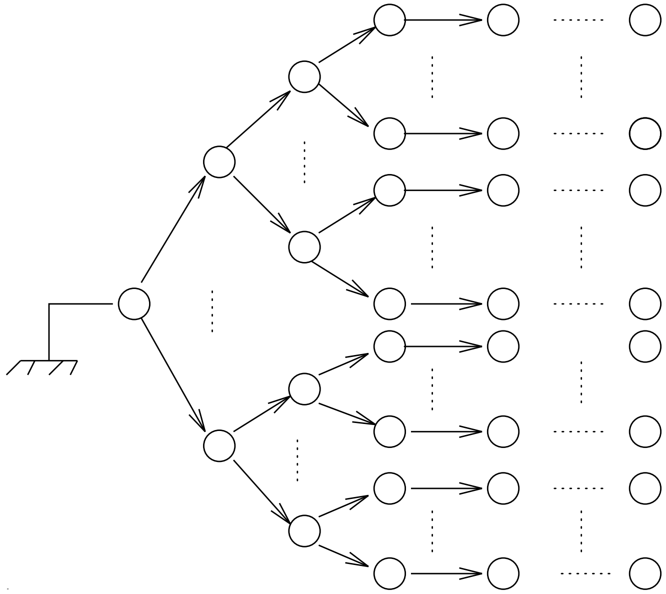


Figure 11: $A = (D,M)$-μ-ary tree

A (D,M)-μ-ary tree has μ^D terminal nodes at depth $D+M$. We associate to each branch of this tree a **branch codeword** with length N, that is, each branch codeword consists of N symbols from the input alphabet of the DMC channel.

Let R be the rate of a (D,M)-μ-ary tree code. Thus, $\mu = 2^{NR}$ denotes the number of branches that leave each node up to depth $D-1$.

Let γ be the constraint length of this tree. γ is defined as the total number of digits contained in the juxtaposition of the branches starting at depth $(D-1)$ up to depth $(D+M)$ of this tree code, $\gamma = (M+1)N$.

A tree code is identified by a 4-tuple (N,R,D,M). Note **tree codes** are a special class of **block codes** for a DMC channel, where the number of codewords is $m_B = \mu^D$, each codeword has length $N_B = (D+M)N$.

As a block code, its rate R_B is given by $R_B = \frac{\log(m_B)}{N_B} = \frac{\log(\mu^D)}{(D+M)N} = \left(\frac{D}{D+M}\right)\left(\frac{\log\mu}{N}\right)$,

$$R_B = \left(\frac{D}{D+M}\right) R.$$

Note that for $D \gg M$, we have $R_B \approx R$.

Once the **tree codes have been characterized as a block code**, the next step is to **derive an upperbound on the average error probability**, $\overline{P}_e$, when the ML decoding decision criterion is employed.

Consider a set of (N,R,D,M) tree codes for a given DMC channel. Assume each digit of the codewords is chosen independently with probability $Q(.)$, from the input symbols to the channel. Each codeword of this tree code so obtained consists of $(D+M)$ branch codewords of the (D,M)-μ-ary tree code.

Define E_i, $1 \leq i \leq D$, as the event in which some path that, in the direction to the terminal node, diverging from the correct path in a node at depth $(D-i)$ from the root node, is more likely to provide the same received sequence as the one the corresponding segment of the correct path would do. Then,

$$P_e \leq P(E_1 \cup E_2 \cup .... \cup E_D),$$

where the inequality is necessary due to the fact that we may decode correctly even when the correct path assumes the same value as the other path with respect to having the same chance of being decoded correctly. By the union bound

$$P_e \leq P(E_1) + P(E_2) + ... + P(E_D). \tag{10}$$

Note that there are $(\mu-1)\mu^{i-1}$ paths diverging from the correct path in the node at depth $(D-i)$ from the root node up to the terminal nodes and that each one of these paths consists of $(M+i)$ branches, or equivalently, of $(M+i)N$ symbols of the input alphabet of the channel.

The probability of one of these paths to be more likely than the corresponding segment of the correct path is simply the error probability of a code containing two codewords of length $(M+i)N$. Thus, using the bound on the average error probability for codes containing only two codewords of length $(M+i)N$ and the union bound, we have

$$\overline{P(E_i)} \leq (\mu-1)\mu^{i-1}2^{-(M+i)NR_0}. \tag{11}$$

Now, averaging $P_e \leq \sum_{i=1}^{D} P(E_i)$, and substituting equation (11) in (10), we arrive at

$$\overline{P}_e \leq (\mu-1)2^{-(M+1)NR_0}\left[\sum_{i=1}^{D}\mu^{i-1}2^{-(i-1)NR_0}\right]. \tag{12}$$

Note that the expression in brackets, equation (12), is a geometric series that converges for $\mu < 2^{NR_0}$, and if we let $D \to \infty$, then

$$\overline{P}_e \leq \frac{\mu-1}{1-\mu 2^{-NR_0}}2^{-(M+1)NR_0}, \qquad \text{for} \qquad \mu < 2^{NR_0}.$$

Finally, by upperbounding $(\mu - 1)$ by μ, knowing that $\mu = 2^{NR}$, and that $\gamma = (M+1)N$, we conclude that the **average error probability for the ensemble of** (N,R,D,M) **tree codes ML decoded** in a DMC channel satisfies

$$\overline{P}_e < \alpha 2^{-\gamma R_0}, \qquad \text{for} \qquad R < R_0$$

where

$$\alpha = \frac{1}{2^{-NR} - 2^{-NR_0}},$$

is a constant for any value of R, however fixed, such that $R < R_0$.

Note that an (N,R,D,M) **tree code** for a DMC channel is a **special case** of a type of (N_B, R_B) **block code**, where $N_B = (D+M)N$ and $R_B = [D/(D+M)]R$. Note also that we may consider an (N,R) **block code** for a DMC channel as a **particular case of an** $(N,R,D=1,M=0)$ **tree code**. **Whether the block codes encompasses the tree codes or not, it is only a matter of choice**. According to the objectives to be achieved (analyses of convolutional codes) the preference is to think of the block codes as being a particular case of the tree codes.

10 Trellis Codes

The class of trellis codes is one of the most important class of tree codes whose encoding process introduces a dependency between the information symbols to be encoded.

Let $\{\alpha_i\}_{i=0}^{D-1}$ be a μ-ary information sequence such that each symbol α_i is associated with a tree branch leaving each node up to depth $D-1$.

For each sequence $\alpha_0^{(i)}, \alpha_1^{(i)}, \cdots, \alpha_{D-1}^{(i)}$ there exists a path in the tree starting at the root node and ending in a node at depth D.

An **encoding procedure** for a (D,M,m)-μ-ary tree code where the nodes are identified by labels is as follows:

- Let $m \in [M, D+M]$. Label each node in the (D,M)-μ-ary tree by one of the m values of the previous information digits.
- Assume from depth D up to depth $D+M$ only one digit "0" is associated with each one of the branches.

Definition 2. *If a tree has memory m, in the sense that for any two nodes at the same depth result in the same encoded sequence when the information sequence is input to the encoder from each one of these nodes (states), then we have to keep just one of these sequences. When this occurs, the resulting code is said to be a* (D, M, m)*-*μ*-ary trellis code.*

The 5-tuple (N, R, D, M, m)-μ-ary defines a trellis code for the DMC channel where for each trellis branch we associate a **branch codeword** consisting of N channel input symbols.

Example 3. *Consider the binary tree code and the corresponding trellis code specified by* $(N, R, 3, 2, 2)$ *as shown in Fig. 12. It is also shown in the same figure the binary tree code and the corresponding trellis code specified by* $(N, R, 3, 1, 2)$ *where the tree code has tail depth* $M = 1$ *(on the left) and the trellis code with two terminal states, denoted by 00 and 01 (on the right).*

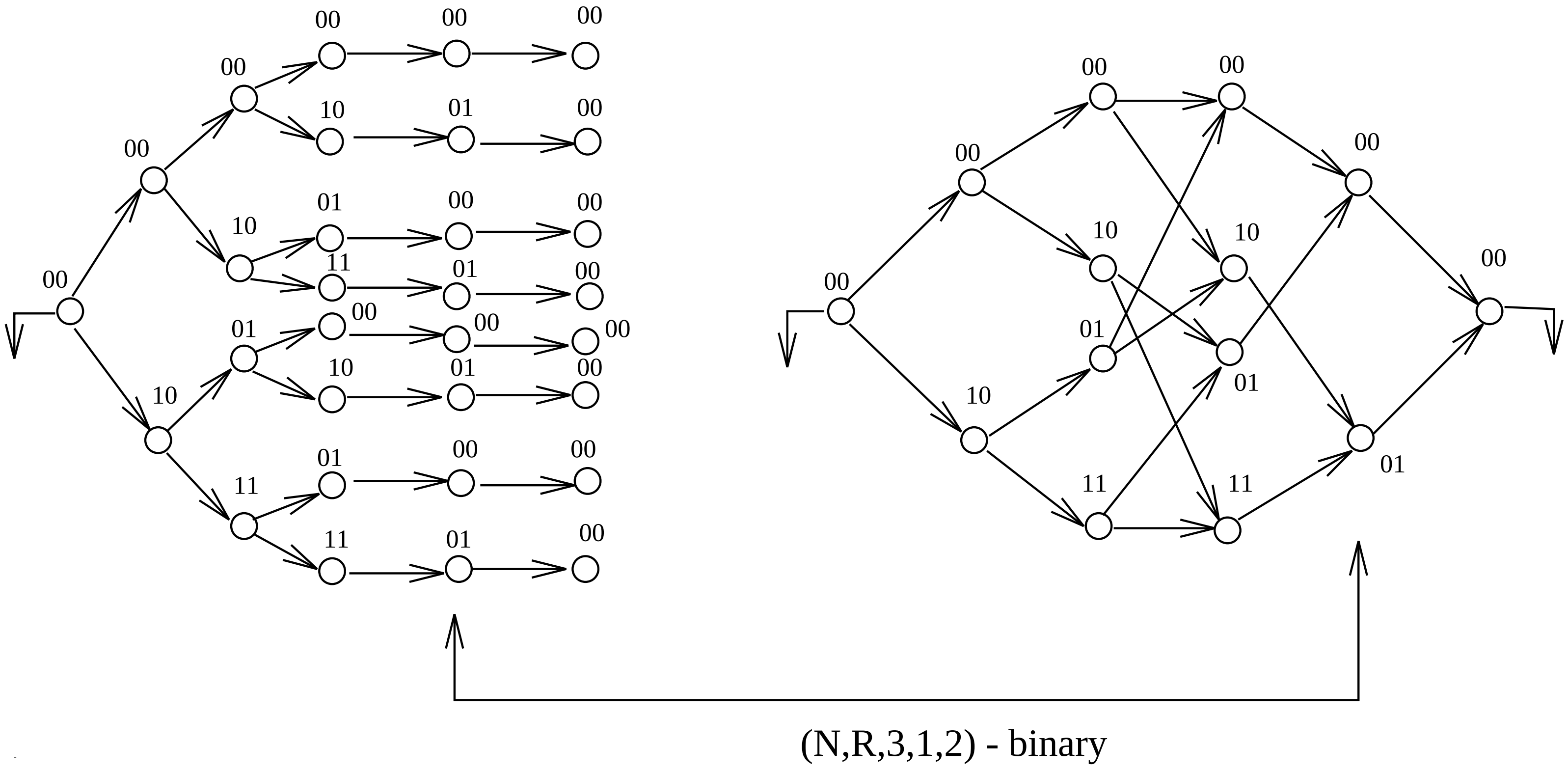


Figure 12: Binary trellis codes (N,R,3,2,2) and (N,R,3,1,2)

The set of (N,R,D,M,m)-μ-ary trellis codes is a special class of the (N,R,D,M) tree codes. The constraint length, γ, of the trellis code has the same value as the constraint length of the tree code, $\gamma = (M+1)N$. Note that $R_B = [D/(D+M)]R$ relates **nominal rate** R of the trellis code with the rate R_B of the tree code measured in bits/channel use.

Our aim is to establish an upperbound on the average error probability, $\overline{P}_e$, with ML decoding for the DMC channel over an ensemble of trellis codes where there is a probability of each such a code being selected.

This probability may be easily determined by the following hypothesis: each digit of a codeword of a (D,M,m)-μ-ary trellis code is selected independently with the distribution $Q(.)$.

Note that **when $m = M$, there exists only one terminal node in the trellis**, this terminal node is labeled "zero", that is, $(00...0)$. Hence, **every path in the trellis starts and ends in the root node**.

Following the same steps and arguments as in the tree codes case, it can be shown that

$$\overline{P}_e \leq \alpha D 2^{-(m+1)NR_0}, \qquad \text{for} \qquad R < R_0 \tag{13}$$

as the upperbound on $\overline{P}_e$ with ML decoding of the set of $(N,R,D,M,m=M)$ trellis codes.

As can be seen in equation (13), **this upperbound depends on the parameter** D whereas the **upperbound on the tree codes does not**.

Now, we **consider the general case**, that is, **the condition** $m = M$ **does not apply**. Under this hypothesis, the **trellis has** $\mu^{(m-M)}$ **terminal nodes**, where $M \leq m \leq D+M$. **When** $m = M$**, there exists only one terminal node**.

Assume that a path going to the terminal node diverging from the correct path at depth $(D-i)$ is more likely to be selected than the corresponding segment of the correct path. **If this path returns to the correct path, then this error event is included in the event** E_i **for the trellis code with** $m = M$.

If this path does not return to the correct path, then this error event is included in the event E_i for the tree code with tail length M.

Thus, $\overline{P}_e$, with ML decoding of the ensemble of (N,R,D,M,m) trellis codes, is upperbounded by the sum of $\overline{P}_e$ for the ensemble of the $(N,R,D,M,m=M)$ trellis codes and of the $\overline{P}_e$ for the ensemble of the (N,R,D,M) tree codes,

$$\overline{P}_e < \alpha D 2^{-(m+1)NR_0} + \alpha 2^{-(M+1)NR_0},$$

which may be written as

$$\overline{P}_e < \alpha \left\{ 1 + D 2^{-(m-M)NR_0} \right\} 2^{-\gamma R_0}, \quad \text{for} \quad R < R_0.$$

This is the upperbound on $\overline{P}_e$ with ML decoding for the ensemble of (N,R,D,M,m) trellis codes.

Now, taking a closer look to this upperbound, we may observe that the dependency on D, not desirable, for the trellis codes when $m = M$ may be surpassed. For suppose that $D.2^{-(m-M)NR_0} \leq 1$, or equivalently, $m - M \geq (\log D)/NR_0$. Hence,

$$\overline{P}_e < 2.\alpha.2^{-\gamma R_0}.$$

The conclusion is, if we choose a value for m a little bit greater than M, it is more than sufficient for the upperbound on $\overline{P}_e$ of the ensemble of the trellis codes be very close to the upperbound on $\overline{P}_e$ of the ensemble of the tree codes.

We have observed earlier that the (N,R,D,M,m) trellis code for the DMC channel is a special case of the (N,R,D,M) tree code. On the other hand, observe that we may think of an (N,R,D,M) tree code as being a special case of the $(N,R,D,M,m = D+M)$ trellis code since when $m = D+M$, the trellis degenerates to a tree with $\mu^{(m-M)} = \mu^D$ terminal nodes.

11 Convolutional Codes

As we have seen in the previous sections, **the ensemble of (N,R,D,M)-μ-ary tree codes for a DMC channel coincide with the set of $(N,R,D,M,m=D+M)$ trellis codes**. Hence, **it is sufficient to consider only the set of trellis codes**.

Let $\left\{\alpha_j^{(i)}\right\}_{j=0}^{D-1}$ be a sequence of information symbols to be encoded. As seen previously, each $\alpha_j^{(i)}$ is associated with a branch in the trellis that goes from the root node up to a node at depth D in the tree. Let $\left\{\beta_j^{(i)}\right\}_{i=0}^{D+M-1}$ be the corresponding encoded sequence where $\alpha_j^{(i)} = \mathbf{0}$ for $D \leq i \leq D+M$.

The mathematical representation of the encoder is the following:

$$\beta_i = \Gamma(\alpha_i, \alpha_{i-1}, \alpha_{i-2}, \ldots, \alpha_{i-m}, i),$$

where for notational easiness the superscript i in the information and encoded symbols is not considered.

Note that $\Gamma(.,.)$ is an arbitrary function of $\alpha_{i-1}, \alpha_{i-2}, ..., \alpha_{i-m}$ and of the time $t = i$. However, when β_i is specified by

$$\beta_i = \Gamma(\alpha_i, \alpha_{i-1}, \alpha_{i-2}, ..., \alpha_{i-m}),$$

the encoder is said to be **time-invariant** or **fixed**. Note also that, as $\Gamma(.)$ is an arbitrary function, it may be **linear** or **nonlinear**. When $\Gamma(.)$ is **linear**, we have a special class of the trellis codes called **convolutional codes**. A convolutional encoder is a linear trellis encoder. In order to complete the mathematical structure of the encoder, an algebraic structure in the **information symbols** as well as in the **encoded symbols** have to be included.

Suppose the input alphabet of a DMC channel belongs to $GF(q)$, where q is a prime number or a power of a prime number. Let $\mu = q^L$ be such that each α_i is an L-tuple over $GF(q)$.

The rate of this code is given by

$$R = \frac{\log \mu}{N} = \frac{L}{N} \log q, \quad \text{bits/channel use}$$

The (N,R,D,M,m) convolutional code with

$$\beta_i = \Gamma(\alpha_i, \alpha_{i-1}, \alpha_{i-2}, \ldots, \alpha_{i-m}, i)$$

is such that $\alpha_i \in GF(q)^L$ and $\beta_i \in GF(q)^N$, and $\Gamma(.,i)$ are linear functions given by $\Gamma(.,i) : GF(q)^{(m+1)L} \rightarrow GF(q)^N$.

Such functions may be represented by

$$\beta_i = \alpha_i G_0(i) \oplus \alpha_{i-1} G_1(i) \oplus \ldots \oplus \alpha_{i-m} G_m(i), \qquad (14)$$

where the $G_j(i)$s are $L \times N$ matrices whose elements are over $GF(q)$.

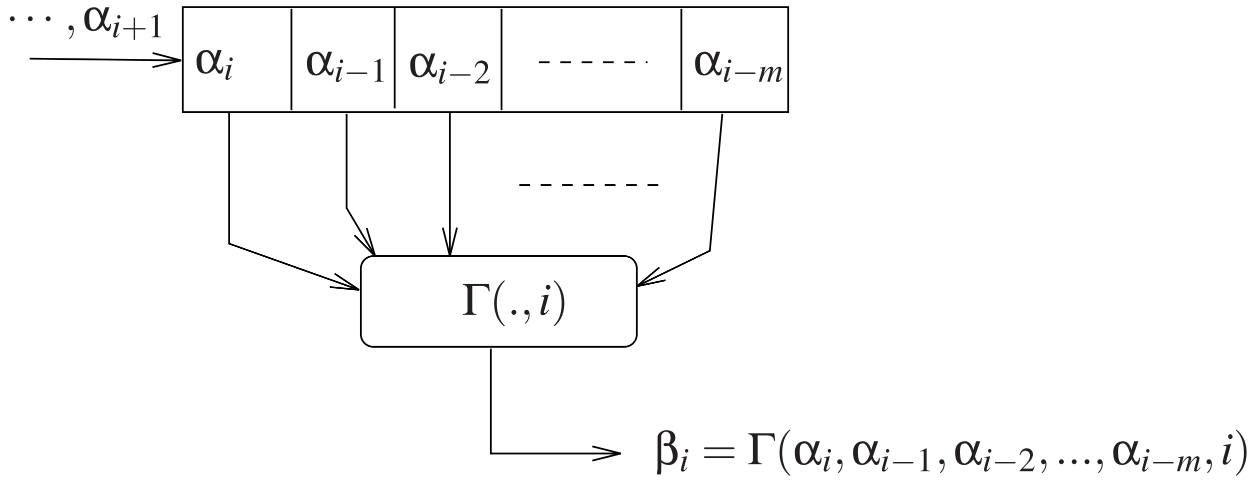


Figure 13: Time-varying convolutional encoder from equation (14)

For time-invariant functions,

$$\beta_i = \alpha_i G_0 \oplus \alpha_{i-1} G_1 \oplus \ldots \oplus \alpha_{i-m} G_m, \tag{15}$$

where the G_js are $L \times N$ matrices whose elements are over $GF(q)$.

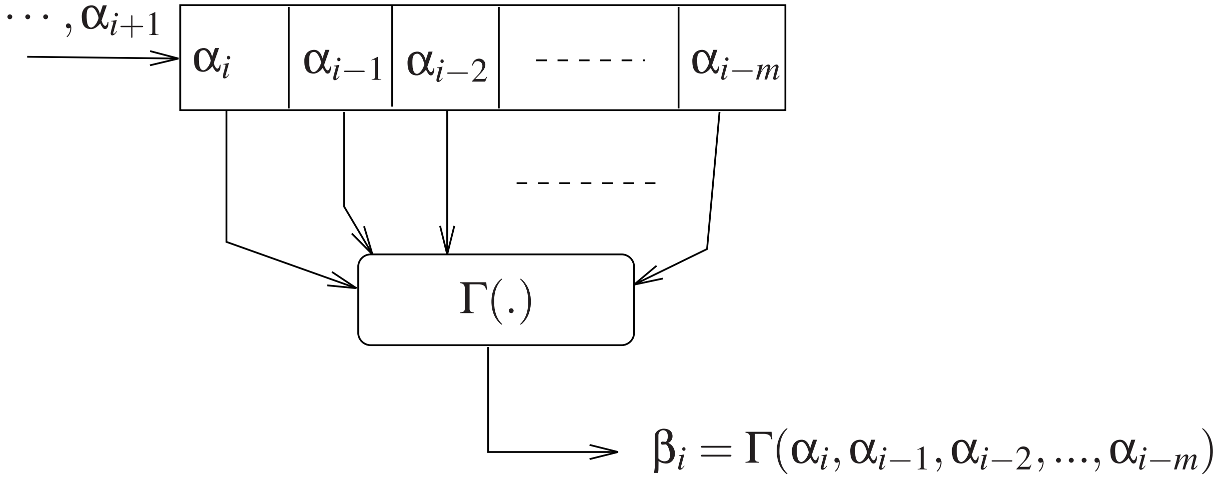


Figure 14: Time-invariant convolutional encoder from equation (15)

In order to have a matrix representation for the generator matrix of the convolutional code as it happens with the block codes.

Let $\alpha_{[u,v]} = (\alpha_u, \alpha_{u+1}, \dots, \alpha_v)$ and $\alpha_{[u,v)} = (\alpha_u, \alpha_{u+1}, \dots, \alpha_{v-1})$ be sequences. Let $\beta_{[u,v]}$ and $\beta_{[u,v)}$ be sequences similarly defined as $\alpha_{[u,v]}$ and $\alpha_{[u,v)}$. From (14),

$$\beta_{[0,D+M)} = \alpha_{[0,D)} \begin{bmatrix} G_0(0) & G_1(1) & \cdots & G_m(m) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & G_0(1) & \cdots & G_{m-1}(m) & G_m(m+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & G_0(m) & G_1(m+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & G_0(D-1) & \cdots & G_M(D+M-1) \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}.$$

Thus, G is called (semi-infinite) **generator matrix** of the (N, R, D, M, m) time-varying convolutional code.

For a time-invariant convolutional code, G (semi-infinite) **generator matrix** is given by

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0} & G_0 & G_1 & \cdots & G_{m-1} & G_m & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{0} & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & \mathbf{0} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots \\ \mathbf{0} & \cdots & \mathbf{0} & G_0 & G_1 & \cdots & G_m & \mathbf{0} & \cdots \\ \ddots & \ddots & \ddots & \ddots & \cdots & \ddots & \ddots & \ddots & \cdots \end{bmatrix}.$$

11.1 Encoding process of the convolutional codes

Fig. 15, shows a convolutional encoder with memory $m = 2$ and rate $r = 1/2$.

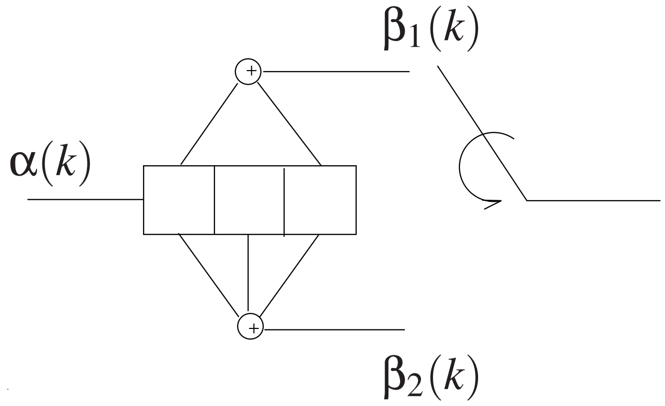


Figure 15: Binary convolutional encoder, $m = 2$ and $r = 1/2$

Let $\alpha = \alpha_{[1,\infty)}$ be an information sequence, $\alpha = (\alpha_1, \alpha_2, ...)$ where each $\alpha_i \in GF(2)$. Note the encoder has a characteristic of a **linear system without feedback** a **non-recursive convolutional code**.

The corresponding encoded sequences β_1 and β_2, are, respectively, given by

$$\beta_1 = \{\beta_1(0), \beta_1(1), \beta_1(2), ...\}, \quad \text{and} \quad \beta_2 = \{\beta_2(0), \beta_2(1), \beta_2(2), ...\}.$$

These sequences are generated by the convolution between the information sequence α and the **generating functions** $\mathbf{g}_1$ and $\mathbf{g}_2$, respectively.

The **generating functions** are obtained by the application of the following information sequences,

$$\alpha = (1,0,0); \qquad \alpha = (0,1,0); \qquad \alpha = (0,0,1),$$

at the encoder input.

Taking as an example the encoder in consideration, the generating functions $\mathbf{g}_1$ and $\mathbf{g}_2$ are given by

$$\mathbf{g}_1 = (1,0,1) \qquad \text{and} \qquad \mathbf{g}_2 = (1,1,1),$$

where "1" means there exists a connection between the $mod\ 2$ adders and the corresponding cells, and “0” there exists no connection.

Since $\mathbf{g}_1$ and $\mathbf{g}_2$ are generating functions, the encoding process of convolutional codes may be established as the convolution of the information sequence α with the generating functions $\mathbf{g}_1$ and $\mathbf{g}_2$, more precisely,

$$\beta_1 = \alpha * \mathbf{g}_1 \qquad \text{and} \qquad \beta_2 = \alpha * \mathbf{g}_2,$$

where * denotes convolution operation, implying that the encoded sequence is given by

$$\beta = \{\alpha_1(0)\alpha_2(0), \alpha_1(1)\alpha_2(1), \alpha_1(2)\alpha_2(2), \ldots\}. \tag{16}$$

The term **convolutional code** comes from the discrete convolution of the input symbols at the encoder with the generating functions. Consider the encoder shown in Fig. 15 at time $t = k$. From equation (16)

$$\begin{cases} \beta_1(k) = \alpha(k) \oplus \alpha(k-2) \\ \\ \beta_2(k) = \alpha(k) \oplus \alpha(k-1) \oplus \alpha(k-2) \end{cases}.$$

Furthermore, if we consider $\mathbf{g}_1$ and $\mathbf{g}_2$ as being represented by

$$\mathbf{g}_1 = (g_{01}, g_{11}, g_{21}) \qquad \text{and} \qquad \mathbf{g}_2 = (g_{02}, g_{12}, g_{22}),$$

then, the generator matrix G of this convolutional code is given by

$$G = \begin{bmatrix} g_{01}, g_{02} & g_{11}, g_{12} & g_{21}, g_{22} & \cdot & \cdot & \cdot \\ \cdot & g_{01}, g_{02} & g_{11}, g_{12} & g_{21}, g_{22} & \cdot & \cdot \\ \cdot & \cdot & g_{01}, g_{02} & g_{11}, g_{12} & g_{21}, g_{22} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \overline{g_0} & \overline{g_1} & \overline{g_2} & \cdot & \cdot & \cdot \\ \cdot & \overline{g_0} & \overline{g_1} & \overline{g_2} & \cdot & \cdot \\ \cdot & \cdot & \overline{g_0} & \overline{g_1} & \overline{g_2} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix},$$

where $\overline{g_0} = (1,1)$; $\overline{g_1} = (0,1)$; and $\overline{g_2} = (1,1)$.

The encoded sequence is written as

$$\beta = \alpha * G.$$

Up to this point, we saw that the generation and the encoding processes of the time-invariant convolutional codes, that is, the generating functions $\mathbf{g}_1$ and $\mathbf{g}_2$, and the submatrices $\overline{g_0}$, $\overline{g_1}$, and $\overline{g_2}$, respectively, are the same all the time.

On the other hand, the generator matrix G of a time-varying convolutional code may be specified by

$$G = \begin{bmatrix} \overline{g_0(1)} & \overline{g_1(2)} & \overline{g_2(3)} & . & . & . \\ . & \overline{g_0(2)} & \overline{g_1(3)} & \overline{g_2(4)} & . & . \\ . & . & \overline{g_0(3)} & \overline{g_1(4)} & \overline{g_2(5)} & . \\ . & . & . & . & . & . \end{bmatrix},$$

where for the example in consideration

$$\overline{g_j(i)} = (g_{j1}(i), g_{j2}(i)),$$

are bi-dimensional binary vectors with $g_{jn}(i)$ denoting the time-varying connections. Note that $\overline{g_j(i)}$ may be seen as 1×2 submatrices. In this way, a generalization for rates $r = k/n$ is quite simple, for now $\overline{g_j(i)}$ are $k \times n$ submatrices with $1 \leq i \leq m+1$, and the subsequent displacements for $u \leq j \leq v$.

11.2 Convolutional codes as finite state machines (Mealy machines)

Definition 3. *A machine M is represented by a 5-tuple $(\alpha, \beta, \mathcal{S}, f, g)$ where α denotes the input set; β denotes the output set; $\mathcal{S}$ denotes the set of states; $f : \alpha \times \mathcal{S} \to \mathcal{S}$ denotes the next state function; and $g : \alpha \times \mathcal{S} \to \beta$ denotes the output function.*

The encoder being a **finite state machine** has a **state diagram** associated with the encoding process.

$$
\begin{aligned}
s_k &= f(\alpha(k-1), \alpha(k-2)) && \rightarrow \quad \text{state} \\
\beta(k) &= g(s_k, \alpha(k)) && \rightarrow \quad \text{output}
\end{aligned}
$$

respectively, where $\beta(k) = (\beta_1(k), \beta_2(k))$.

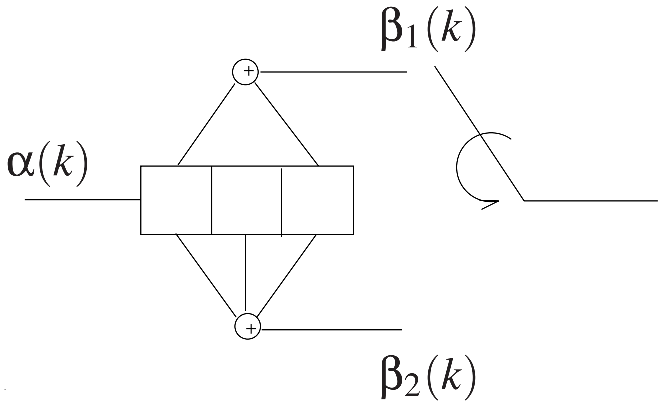


Figure 16: Binary convolutional encoder, $m = 2$ and $r = 1/2$

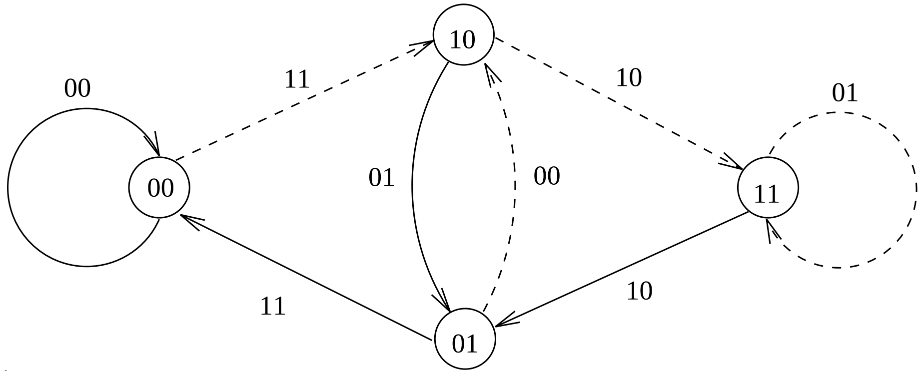


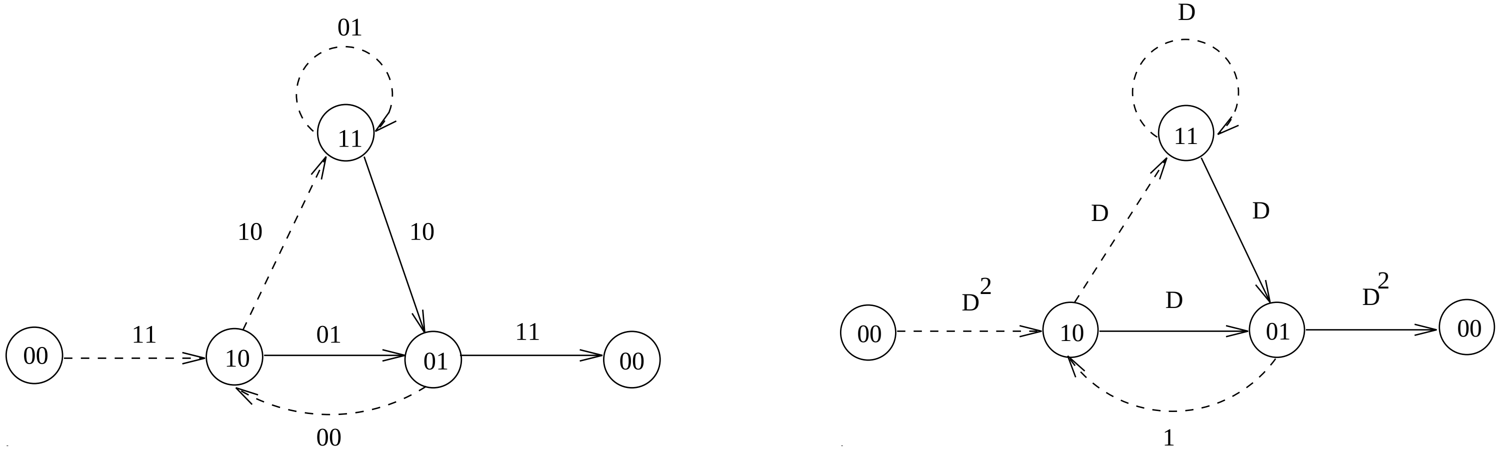
Figure 17: State diagram of the convolutional encoder with $m = 2$ and $r = 1/2$

11.3 Enumeration of the weights of the codewords

Note that the convolutional code form a group code, that is, the codewords of the convolutional code are closed under the addition over $GF(q)$; each codeword has an additive inverse; the all zero codeword is the identity element of the addition operation. These are the conditions that a set of elements with a given operation has to satisfy to be an algebraic group. This statement also holds for any linear code.

For each transition in the state diagram, Fig. 17, there is associated a transfer function of the transition given by D^w, where w denotes the Hamming weight associated with the branch codeword, and $D = \sqrt{4p(1-p)}$ denotes the Bhatacharyya function for a BSC channel, see Fig. 18(b).

Every path starting and ending in the zero state represents a nonzero codeword of a convolutional code. The transfer function resulting for each one of these paths is obtained by the product of the corresponding transfer functions of the transitions.



(a) Binary branch codeword (b) Bhatacharyya function

Figure 18: Split-state diagrams of the CE with $m = 2$ and $r = 1/2$

The **enumerator polynomial** specifying the number of codewords with Hamming weight w is obtained once the split-state diagram may be interpreted as a **discrete-time linear system**.

The **state and output equations** associated with this discrete-time linear system are given by

$$\begin{aligned} E(i+1) &= A(i)E(i) + B(i) \\ T(i) &= C(i)E(i) \end{aligned} \tag{17}$$

- $E(i)$ denotes the **state matrix**, which for the example in consideration, is a 3×1 matrix. The elements of this matrix are the intermediate states at time $t = i$;
- $A(i)$ denotes the **transition matrix**, which for the example in consideration, is a 3×3 matrix. This matrix contains the elements corresponding to the transition between the intermediate states;
- $C(i)$ denotes the **output matrix**, which for the example in consideration, is a 3×1 matrix. This matrix specifies the transitions from the intermediate states to the zero state;
- $B(i)$ denotes the **initial condition matrix**, which for the example in consideration, is a 3×1 matrix. This matrix specifies the transitions from the state zero to the intermediate states.

From Fig. 18(b), we have

$$E(i) = \left[\begin{array}{ccc} \xi_{i1} & \xi_{i2} & \xi_{i3} \end{array}\right] \quad ; \quad B^T(i) = \left[\begin{array}{ccc} D^2 & 0 & 0 \end{array}\right] \quad ; \quad C(i) = \left[\begin{array}{ccc} 0 & D^2 & 0 \end{array}\right]$$

$$A(i) = \left[\begin{array}{ccc} 0 & 1 & 0 \\ D & 0 & D \\ D & 0 & D \end{array}\right] \tag{18}$$

Substituting equations (18) in equations (17), and solving the system of equations, we arrive at

$$T(D) = \frac{D^5}{1-2D} = D^5 + 2D^6 + 4D^7 + \ldots \tag{19}$$

- There exists a path with Hamming weight 5,
- Two paths with Hamming weight 6,
- Four paths with Hamming weight 7, and so on.

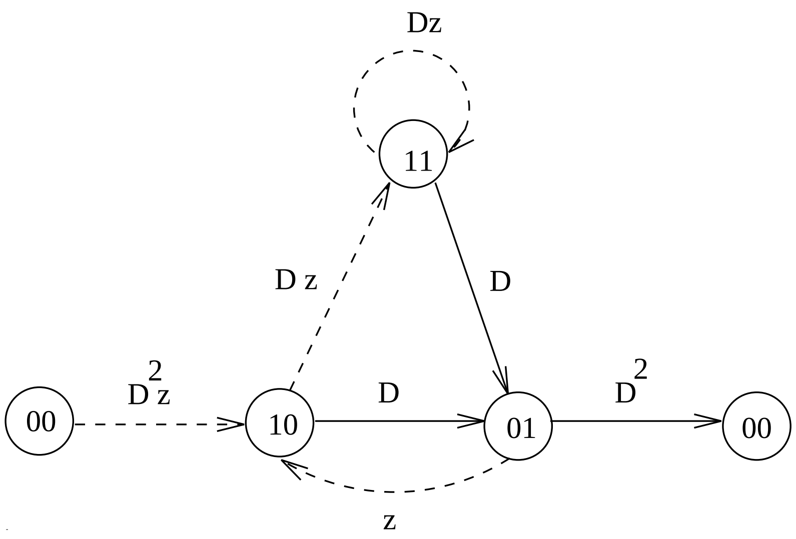


Figure 19: Split-state diagram with z included

if we are interested in counting the number of digits "1" in a given information sequence, the way to do this is to introduce a variable z in all the transitions in the split-state diagram that have been originated by the digit "1". Fig. 19, illustrates this procedure. Note that only the matrices $A(i)$ and $B(i)$ will have to be modified where some elements will have z

appended. These matrices are given by

$$B^T = \begin{bmatrix} D^2 z & 0 & 0 \end{bmatrix} \qquad \text{and} \qquad A(i) = \begin{bmatrix} 0 & z & 0 \\ D & 0 & D \\ Dz & 0 & Dz \end{bmatrix}$$

Substituting these in the previous equations, we arrive at

$$T(D,z) = \frac{D^5 z}{1-2Dz} = D^5 z + 2D^6 z^2 + 4D^7 z^3 + \ldots \tag{20}$$

- There exists a path with Hamming weight 5 whose corresponding information sequence contains only one digit "1",
- There are two paths with Hamming weight 6 whose corresponding information sequences contain two digits "1",
- There are four paths with Hamming weight 7 whose corresponding information sequences contain three digits "1", and so on.

12 Distance Properties of Convolutional Codes

The performance of the convolutional code is directly related with the decoding algorithm and with the distance properties. The distances used with convolutional codes are the following:

- **free distance**, denoted by d_{free}, or by d_∞;
- **column distance**, denoted by d_i;
- **minimum distance**, denoted by $d_{\min}$.

It is from the **free distance** that the error-correction is derived.

12.1 Free distance

The free distance d_{free} of a convolutional code is defined as the least Hamming distance between any two encoded sequences originated from two distinct information sequences, that is,

$$d_{free} \doteq \min\{d(\beta_1, \beta_2) : \alpha_1 \neq \alpha_2\},$$

where β_1 and β_2 are the encoded sequences corresponding to the information sequences α_1 and α_2, respectively. Hence, d_{free} is the least distance between any β_1 and β_2. Hence,

$$d_{free} \doteq \min\{w(\beta_1 \oplus \beta_2) : \alpha_1 \neq \alpha_2\} = \min\{w(\beta) : \alpha \neq \mathbf{0}\} = \min\{w(\alpha G) : \alpha \neq \mathbf{0}\},$$

where β is the encoded sequence corresponding to the information sequence α.

Consequently, d_{free} corresponds to the minimum Hamming weight between all the possible paths in the trellis that diverge from the zero state and come back to the zero state after some transitions in the trellis.

12.2 Column distance

With respect to the column distance, d_i, consider an encoded sequence being truncated after i transitions in the trellis. This sequence is denoted by $\beta_{[0,i]}$, or by

$$\beta_{[0,i]} = (\beta_0, \beta_1, ..., \beta_i).$$

Let $\alpha_{[0,i]}$ be the corresponding truncated information sequence, that is,

$$\alpha_{[0,i]} = (\alpha_0, \alpha_1, ..., \alpha_i).$$

The column distance of order i, denoted by d_i, is defined as

$$d_i \doteq \min\left\{d(\beta^1_{[0,i]}, \beta^2_{[0,i]}) : \alpha^1_{[0]} \neq \alpha^2_{[0]}\right\}$$

$$= \min\left\{w(\beta_{[0,i]}) : \alpha_{[0]} \neq \mathbf{0}\right\}.$$

Thus, d_i is the minimum Hamming weight of the encoded sequence of length $(i+1)$ such that the first block of the information sequence is different from zero, that is, $\alpha_{[0]} \neq \mathbf{0}$.

To better characterize the determination of the column distance of order i by the truncated encoded sequence $\beta_{[0,i]}$ is that we explicit $\beta_{[0,i]}$, the generator matrix $G_{[0,i]}$, and the truncated information sequence $\alpha_{[0,i]}$, that is,

$$\beta_{[0,i]} = \alpha_{[0,i]} G_{[0,i]},$$

where $G_{[0,i]}$ is a $b(i+1) \times n(i+1)$ submatrix of the matrix G and that it is given by

$$G_{[0,i]} = \begin{bmatrix} G_0 & G_1 & G_2 & \ldots\ldots & G_i \\ . & G_0 & G_1 & \ldots\ldots & G_{i-1} \\ . & . & G_0 & \ldots\ldots & G_{i-2} \\ . & . & . & . & . \\ . & . & . & . & G_0 \end{bmatrix}, \qquad \text{for} \quad i \leq m$$

or

$$G_{[0,i]} = \begin{bmatrix} G_0 & G_1 & \ldots\ldots & G_m & . \\ . & G_0 & \ldots\ldots & G_{m-1} & G_m \\ . & . & . & . & . \\ . & . & G_0 & G_1 & G_2 \\ . & . & . & G_0 & G_1 \\ . & . & . & . & G_0 \end{bmatrix}, \qquad \text{for} \quad i > m$$

then,

$$d_i = \min\left\{w(\alpha_{[0,i]} G_{[0,i]}) : \alpha_{[0]} \neq \mathbf{0}\right\}, \tag{21}$$

depends only on the $n(i+1)$ columns of G. This is the reason why d_i is called **column distance**. As can be seen, as long as the value of i increases the value of d_i does not decrease.

From the **column distance**:

- When $i = m$, d_m is called the **minimum distance** of the convolutional code, denoted by $d_{\min}$, of the encoded sequence relative to the first constraint length.
- When $i \to \infty$, $\lim_{i\to\infty} d_i$ is the minimum Hamming weight of the encoded sequence of any length whose first block of the information sequence is different from zero. Hence,

$$d_{free} \doteq d_\infty = \lim_{i\to\infty} d_i,$$

  for the noncatastrophic codes. Therefore, d_i eventually will reach d_{free} and from this point on will not increase anymore.
- A characteristic of catastrophic convolutional codes is **a finite number of errors introduced by the channel results in an infinite number of decoding errors**. Therefore, these codes are not appropriate for channel coding applications. However, they find their place when unequal error protection is the main objective.

Chapter 3

Classes of Time-Invariant Convolutional Codes[a]

[a]Most of the content of this chapter may be found in the papers by Forney, Lee, Yasuda, Hagenauer, Daut, Palazzo, etc, and books Viterbi/Omura, Shu Lin/Costello, Dholakia, Johannesson/Zigangirov, Benedeto/Biglieri, Gallager, etc.

13 Specific Convolutional Codes

The aim is to show a similarity between the problem of finding good **convolutional codes** (CC) with that of **finding flow in networks**.

- The first step is to find **flow in networks**. Here, we propose to determine the branch values (branch Hamming weights) of a **network of flows** (split state diagram) by use of the **max-flow min-cut theorem of Ford and Fulkerson**.
- The second step is to solve a **knapsack problem** via the **Branch and Bound algorithm**.
- The third step is to obtain the generator matrices G_i, $0 \leq i \leq m$, with m the encoder memory, of the CC by use of the **modular representation of block codes**.

This association allows us to relate the parameters of each problem as follows:

- The split state diagram representation of CC will be associated with a network of flows;

- The states are associated with the nodes;
- The Hamming distance of each transition in the split state diagram is associated with the branch flow;
- The property of conservation of flows holds, that is, the sum of the branch Hamming distances going to a given intermediate state equals the sum of the branch Hamming distances leaving from the given state;
- The least value of the sum of the branch Hamming distances belonging to the minimum cut set is associated with the maximum flow;
- The sum of the branch Hamming distances leaving from (going to) the zero state in the split state diagram equals the maximum flow.

13.1 The link between coding and combinatorics

Let G be a directed graph (split state diagram) $G = [N, \tilde{A}]$ consisting of a finite collection N, $N = \{1, 2, ..., \widetilde{n}\}$ together with a set $\tilde{A}$ of non necessarily distinct ordered pairs (i, j), that is, $\tilde{A} = \{(i, j) : i$ and $j \in N\}$. Elements of N and $\tilde{A}$ are called **states** and **branches**, respectively.

Let us associate with each branch (i, j) of a directed graph (split state diagram) a non negative integer c_{ij}, the **capacity of** (i, j), to be thought of as representing the **maximum branch Hamming distance of the branch codeword** between the adjacent nodes i and j.

The concept of **cut** is useful for modeling the carrying capacity of a network. An (i, j) **cut** is a set of arcs whose removal disconnects node i from node j. A **minimal cut** is one in which replacement of any of its members reconnects the graph. In a **weighted graph**, each **cut** has a **capacity**. The **capacity of a cut** is the sum of the weights of the arcs in the cut. A **cut** with the **minimum capacity** is called a **minimum cut**.

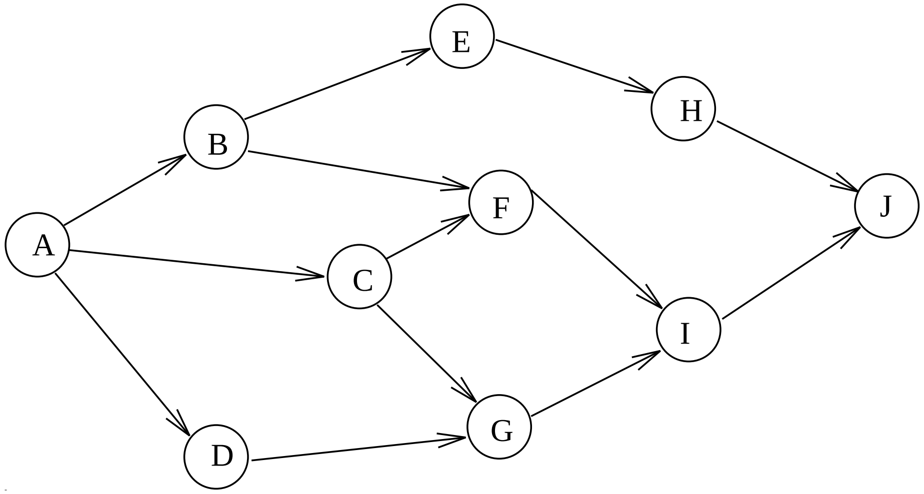


Figure 20: Directed graph

Consider a flow from node A to node J. This flow is **feasible** if

- the source has no inwards arcs;
- the sink has no outwards arcs;
- no arc has more flow than its capacity;
- the inflow and outflow balance at all nodes except the source and the sink;
- the outflow at the source must equal the inflow at the sink.

If there are multiple sources and sinks, the single source-sink pair graphs may be linearly superimposed.

Theorem 1. *(Max-Flow Min-Cut Theorem) The maximum flow between any two arbitrary nodes in a graph is equal to the capacity of the minimum cut separating those two nodes.*

Corollary 1. *A flow* Φ *is maximum if and only if there is no flow augmenting path with respect to* Φ.

The linear programming nature of the network flow problem is:

Problem 1.

$$\max\{v\}$$

subject to

$$\sum_i w_{ij} - \sum_k w_{jk} = \begin{cases} -v & \text{if} \quad j = 1 \\ 0 & \text{if} \quad j \neq 1 \text{ and } \widetilde{n} \\ v & \text{if} \quad j = \widetilde{n} \end{cases} \tag{22}$$

$$0 \leq w_{ij} \leq c_{ij} \tag{23}$$

with v being its flow value.

Note that (22) and (23) **are exactly the conditions that have to be met by a flow to be feasible**, that is, **flow is conserved at every node except the source and the sink**, and each branch flow w_{ij} is bounded from above by c_{ij}, the branch capacity. From the **max-flow min-cut theorem, the maximum flow value $v = \Phi$, from the source to the sink in a network, is equal to the capacity of a minimum cut separating the source and the sink**.

It has also been observed that the class of nonsystematic CCs (specific or not) satisfies both the maximum flow and the conservation of flow properties, and so achieving the largest free distance.

13.1.1 Review of CCs and a claim

Our claim is that the split state diagram is the proper link between the problem of finding good convolutional encoders and that of finding flow in network.

Let us consider an (n,k,m) convolutional encoder over $GF(q)$. Let G be the semi-infinite generator matrix of this CC

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \ldots & G_m & . & . \\ . & G_0 & G_1 & \ldots & G_{m-1} & G_m & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & G_0 & G_1 & \ldots & G_m \end{bmatrix}. \tag{24}$$

The free distance is defined as $d_{free} = \min_{\forall \mathbf{u}_{[0,\infty)}, \mathbf{u}_0 \neq \mathbf{0}} \left\{ w(\mathbf{v}_{[0,\infty)}) \right\}$. From the definition of d_{free}, and from the interpretation that the first row of (24) can be viewed as a $k \times nm$ generator matrix of a block code, we conclude that d_{free} is upper bounded by the best Hamming distance of an (nm,k) block code.

Now consider the steady state has been reached in the trellis diagram for this convolutional encoder. Assume that at time $t = t' - m - 1$, and also at time $t = t'$ the content of the cells of the shift-register(s) are zero, that is, the encoder is at state zero.

The transitions between states in the time interval $[t' - m - 1, t' - m]$ and $[t' - 1, t']$ are those transitions from the zero state to the intermediate states, and from the intermediate states back to the zero state, respectively, and in the time interval $[t' - m - 1, t' - 1]$ the transitions are only between intermediate states. This is the **split state diagram** representation of the convolutional encoder.

The transitions between the zero state and the intermediate states in subsequent time intervals have branch codewords associated with G_0. Similarly, the transitions between the intermediate states and the zero state in subsequent time intervals have branch codewords associated with G_m.

If we select G_0 and G_m as $k \times n$ generator matrices of block codes with the best Hamming distance for the parameters (n,k), that is, $d(n,k)$, and such that the nonzero codewords have Hamming weight equal to or as close as possible to $d(n,k)$, certainly $rank(G_0) = rank(G_1) = k$ and the total Hamming weight of the branch codewords for each one of these block codes is $n(q-1)q^{k-1}$.

It is quite clear that to achieve this total Hamming weight no column of G_0 and of G_m can be all-zero, for otherwise it would lead to $k \times (n-1)$ generator matrices each with total Hamming weight $(n-1)(q-1)q^{k-1}$.

On the other hand, no row can be all-zero for otherwise it would imply that the submatrices are $(k-1) \times n$. However, we can relax the condition $rank(G_m) = k$ to $rank(G_m) < k$, but keep the constraint that the total Hamming weight be $n(q-1)q^{k-1}$. This is equivalent to satisfying the condition of maximum flow in a network.

Now, to satisfy the conservation of flow (middle equality in (22)) a sufficient condition is that the remaining submatrices generate block codes with total Hamming weight $n(q-1)q^{k-1}$.

This implies that $rank(G_i) \leq k$, $1 \leq i \leq m-1$. When $rank(G_i) = k$ for $1 \leq i \leq m-1$, we call this code a ranked CC. Otherwise, a unranked CC. Note that (23) is also satisfied since no branch codeword can have Hamming weight greater than n.

Conjecture 1. *A time-invariant nonsystematic convolutional encoder with rate $r = k/n$ and memory m is **optimum** if and only if: 1) $M_i(A') = M^j(A') = D^{\Phi}$ for all $k \neq 1$ and for all $i \neq j$ and vice-versa, with Φ the maximum flow; 2) the maximum flow augmenting path is achieved.*

We conjecture that among the possible w_{ij} flow sets yielding the maximum flow Φ, there exists at least one w_{ij} flow set such that the maximum capacity flow augmenting path is achieved. Therefore, if there exists a nonsystematic convolutional encoder satisfying the conjecture then the feasible flow conditions (22) and (23) are met. Hence, the problem of finding good nonsystematic CCs is characterized by a maximal flow in networks.

The next theorem establishes the maximum flow that good TICEs satisfy.

Theorem 2. *For time-invariant nonsystematic CCs over $GF(q)$ with rate $r = k/n$, and memory m, the maximum flow is given by*

$$\Phi = n(q-1)q^{k-1}$$

Corollary 2. *For time-invariant nonsystematic binary CCs with $k = 1$, the branch codewords going to and leaving from any state complement each other.*

The problem we are faced with is:

Problem 2. *Let be given the rate $r = k/n$, k and n integers with $\gcd(k,n) \geq 1$, and the memory m of a nonsystematic convolutional encoder over $GF(q)$. Knowing that the flow value is $\Phi = n(q-1)q^{k-1}$, determine the w_{ij} flow set such that the maximum capacity flow augmenting path is achieved.*

Since the problem of finding the maximum capacity flow augmenting path is similar to the problem of finding a shortest path (Lawler), the solution of Problem 2 comes from the application of both the max-flow min-cut and the shortest path algorithms

Max-Flow Min-Cut Algorithm (Ford and Fulkerson)

Input: A directed network with source 1 and a sink $\widetilde{n}$.

Output: A maximum $(1,\widetilde{n})$ flow w_{ij}.

1. Set $w_{ij} = 0$ for all i, j;
2. Find a flow augmenting path C from 1 to $\widetilde{n}$. If none exists, go to 6;
3. Compute and record the slack of each forward arc of C and record the flow of each backward arc of C;
4. Compute η, the minimum of the numbers recorded in 3;
5. Redefine w_{ij} by adding η to the flow on all forward arcs of C and subtracting η from the flow on all backward arcs of C. Return to 1;
6. Stop with the flow w_{ij}.

Maximum Capacity Flow Augmenting Path/Shortest Path Algorithm (Dijkstra)

Input: A directed network D and nodes x and y from D.

Output: A shortest path from x to y or the message that there is no path from x to y.

1. Initially place node x in the class W. Let $B = \Phi$, and $d(x,x) = 0$;
2. For each node u in W and each node v not in W, let $\alpha(u,v) = d(x,u) + w(v,u)$. Find u in W and v not in W such that $\alpha(u,v)$ is minimized;
3. If the minimum in 2 is infinite, stop and give the message that there is no path from x to y;
4. If the minimum in 2 is not infinite, then place v in W (v is the next node chosen), place arc (u,v) in B, and set $d(x,v) = \alpha(u,v)$;
5. If y is not in W, return to 2. If y is in W, stop. A shortest path from x to y can be found by using the unique path of arcs of B which goes from x to y. This can be found by working backward from y.

Example 4. *Let us consider the case where we want to find a good binary convolutional encoder with memory $m = 2$ and rate $r = k/n = 1/15$. The corresponding split state diagram for this convolutional encoder is shown in Fig. 23*

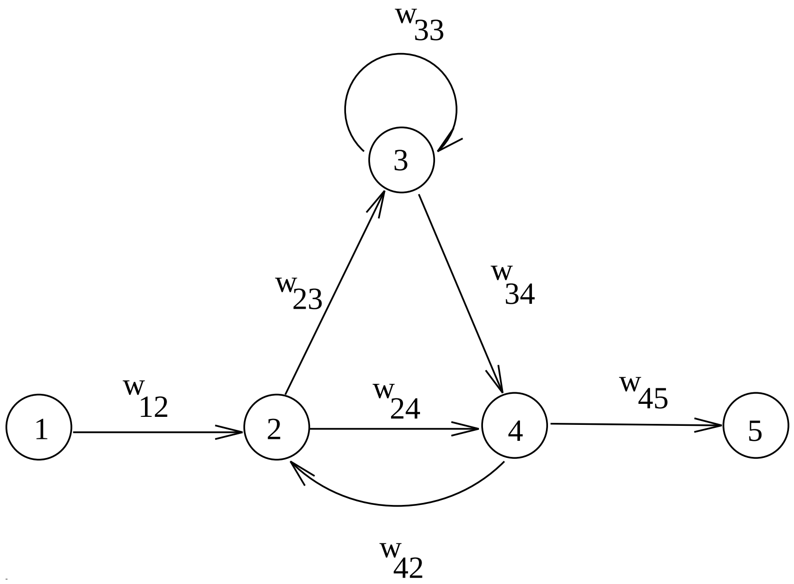


Figure 21: Convolutional encoder with $m = 2$ and $r = 1/15$

From the statement of the problem, we have that each branch has value $c_{ij} = 15$ as its maximum capacity and that the maximum flow $\Phi = n(q-1)q^{k-1} = 15$ from Theorem 2. Now, from Fig. 23 we see that the (1,2) and (4,5) cut among others separate the source (node 1) from the sink (node 5). Moreover, from the max-flow min-cut theorem we know *that the capacity of the minimum cut set has to be equal to* $\Phi = 15$. *Therefore, from (22) and the previous reasoning, we conclude that* $w_{12} = w_{45} = 15$.

Now, following the max-flow and shortest path algorithms to this problem, we have:

1. *Set* $w_{24} = 15$ *and choose the (1,2,4,5) path. Since the conservation of flow property holds, we have that the remaining flows are zero but* $w_{33} = 15$*and that the weight of this path equals 45 (note that* $w_{23} + w_{34} + w_{42} = 0$, *implying a catastrophic code). However, the (1,2,3,4,5) path has weight 30. Therefore, there exists no flow augmenting path from the source to the sink, however, the maximum capacity flow augmenting path was not achieved;*

2. *Consider* $w_{24} = 14$. *Again, since conservation of flow holds, we have that* $w_{23} = w_{34} = 1$, $w_{33} = 14$ *and* $w_{42} = 0$. *As a consequence, the (1,2,4,5) and (1,2,3,4,5) paths have weight 44 and 32, respectively. Once again, there exists no flow augmenting paths however the maximum capacity flow augmenting path was not achieved either;*

3. *Following this procedure, we end up with* $w_{24} = 10$, $w_{23} = 5$, $w_{33} = 10$, $w_{34} = 5$ *and* $w_{42} = 0$. *Note that there exists no flow augmenting path and that this set of flow values results in the maximum capacity flow augmenting path of 40.*

In Fig. 22, it is shown the split state diagram for the problem in consideration with the corresponding branch flows as solution of the network flow approach. The convolutional encoder associated with the split state diagram of Fig. 22 can be easily determined. The number of taps connecting the 3 cells of the shift-register to the modulo 2 adders are respectively, 15, 10, and 15, that is, the first and the third cells have to be connected to all mod 2 adders whereas the second has to be connected to 10 out of 15 mod 2 adders.

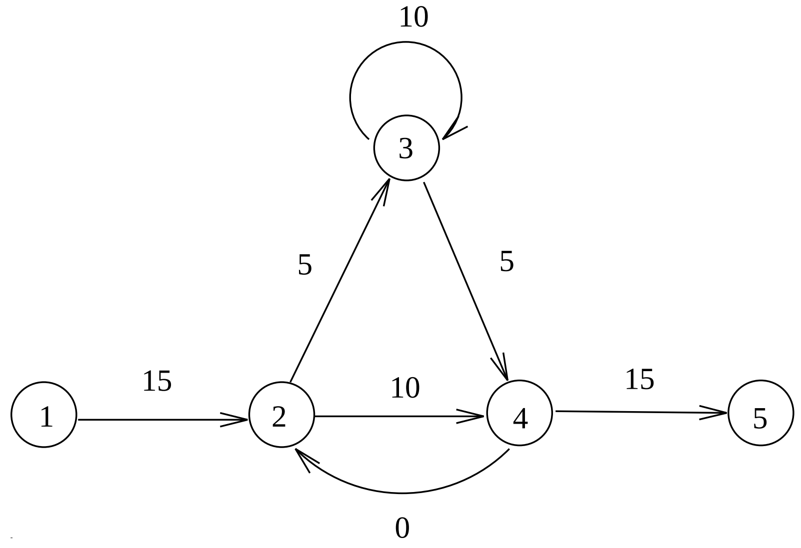


Figure 22: Split state diagram.

In order to speed up the process, the minimum distance of the code is valuable in the exclusion of "potential candidates" from the class of good codes. A Plotkin bound on the minimum distance of any CC over $GF(q)$ with memory m and rate $r = k/n$, $\gcd(k,n) = 1$ is given by

$$d_{\min} \leq \min_{p \geq 1} \left\{ (q-1) \frac{q^{p-1}}{q^p - 1} \frac{n}{k} (p + km) \right\} = \Psi(p^*), \tag{25}$$

where p^* is the optimum solution for a given k, n, and m.

Example 5. *Find a good convolutional encoder over $GF(2)$ with $m = 2$ and $r = 1/85$. The maximum flow is $\Phi = 85$. From the previous theorem, $w_{12} = w_{45} = 85$. This implies that the first and the third cells of the shift-register are connected to all 85 mod 2 adders. The minimum distance is upper bounded by $d_{\min} \leq 226$. If equality holds, then the second cell has to be connected to any of the 56 mod 2 adders out of 85. Anyone of the possible set of tap connections lead to equivalent codes, in the sense of having the same d_{free}. The remaining Hamming weights can be found by either the conservation of flow property, or from the infinite class derivation. Thus, the set w_{ij} of flows is:*

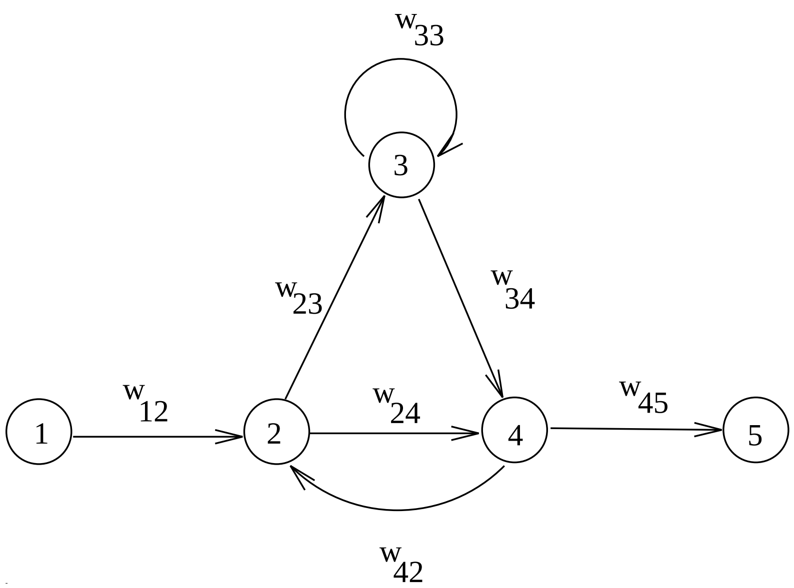


Figure 23: Convolutional encoder with $m = 2$ and $r = 1/85$

$$w_{12} = w_{45} = 85, \quad w_{42} = 0, \quad w_{23} = w_{34} = 29, \quad w_{24} = w_{33} = 56.$$

13.2 Selection Criterion

Although maximum flow and conservation of flow are important properties, they only provide the search in a set whose cardinality is smaller than that of the original problem. The trivial least upper bound on the free distance is another parameter that must be used in the process, which will further reduce the search.

Let Δ be a split state diagram of a convolutional encoder over $GF(q)$ with rate $r = k/n$ and memory m. Let $\underline{\mathbf{d}}$ be the ordered set of all $(q^k - 1)$ accumulated branch Hamming distances along the $(m+1)$ branches that leave from node 1 and go to node $\tilde{n}$, that is, $\underline{\mathbf{d}} = \left\{d_1, d_2, \ldots, d_{q^k-1}\right\}$. For $m = 1$, each $d_i = w_{1i} + w_{i\tilde{n}}$, where w_{1i} and $w_{i\tilde{n}}$ are the Hamming weights leaving from node 1 to the intermediate node i, and from node i to node $\tilde{n}$, respectively. In general, we have that $d_{free} \geq \min\{\underline{\mathbf{d}}\}$.

The optimality criterion to be adopted when comparing ordered sets is of the lexicographical type, that is, given two ordered sets $\mathbf{x}$ and $\mathbf{y}$

$$\mathbf{x} = \{x_1, x_2, \ldots, x_M\} \quad \text{and} \quad \mathbf{y} = \{y_1, y_2, \ldots, y_M\}$$

$$\mathbf{x} > \mathbf{y} \quad \text{if} \quad \begin{cases} x_i = y_i & \text{, for } 1 \le i \le k \\ x_{k+1} > y_{k+1} & \end{cases}$$

We know that $\Phi = n(q-1)q^{k-1}$ is equal to the sum of all branch Hamming distances that leave from or go into a node. Since there are $(m+1)$ branches the maximum possible value attainable by the sum of all d_i along the $(m+1)$ branches is $(m+1)\Phi$. Hence, this is equivalent to a linear programming problem established as follows: let k, n, and m be given

Problem 3 (Knapsack Problem)**.** *Determine the lexicographically optimum λ-tuple $(a_1, a_2, \ldots, a_\lambda)$ of*

$$\sum_{i=1}^{\lambda} a_i d_i \le (m+1)\Phi, \tag{26}$$

subject to

$$\sum_{i=1}^{\lambda} a_i = q^k - 1,$$

with $d_1 = \Psi(p^*)$, $d_2 = \Psi(p^*) + 1$, and so on, and $\lambda = q^k - 1$, where each a_i represents the number of times the value d_i appears.

Note that (26) has at most $q^{k(m+1)}$ possible set of solutions $(a_1, a_2, ..., a_\lambda)$ obtained from the branch-and-bound technique. To each solution set $(a_1, a_2, ..., a_\lambda)$ there corresponds a set $\underline{\mathbf{d}} = \{d_1, d_2, ..., d_\lambda\}$ of Hamming distances. Now, the optimality criterion, lexicographic comparisons, will be employed in the selection of the optimum solution set $\underline{\mathbf{d}}$. Once the optimum $\underline{\mathbf{d}}$ is known, the next step is to solve the following

Problem 4 (Modular Representation)**.** *Knowing the set* $\underline{\mathbf{d}}$ *of Hamming distances, determine the* $(m+1)$ *submatrices* G_i *with elements over* $GF(q)$ *such that the linear combinations of the rows of the* G_i*'s match the solution of Problem 3.*

Note that if one knows the generator matrix G of a block code then the Hamming distance of the codewords can easily be found. However, what we know is a set of specified Hamming distances $\underline{\mathbf{d}}$ for which we want to find a possible set of submatrices G_i that generates this set of distances. The solution of this problem comes from the application of the modular representation of block codes.

Example 6. *As an example of the selection criterion being proposed, let us consider a convolutional encoder with* $m = 1$, $r = 2/11$ *over* $GF(2)$*. The maximum flow is* $\Phi = 112^{2-1} = 22$*. The upper bound on the minimum distance is* $d_{\min} \leq 14$*. Thus, the knapsack problem to be solved is*

$$a_1d_1 + a_2d_2 + a_3d_3 \leq 44,$$

where $d_1 = 14$, $d_2 = 15$, *and* $d_3 = 16$. *So,*

$$14a_1 + 15a_2 + 16a_3 \leq 44. \tag{27}$$

The solution sets of (27) are $(1,2,0)$ *and* $(2,0,1)$. *Consequently,* $(1,2,0)$ *leads to* $\underline{\mathbf{d}} = \{14,15,15\}$ *and* $(2,0,1)$ *leads to* $\underline{\mathbf{d}} = \{14,14,16\}$. *Therefore,* $\underline{\mathbf{d}} = \{14,15,15\}$ *is lexicographically optimum.*

From Problem 4, which is inherently combinatorial, consists of finding the submatrices G_0 *and* G_1 *such that the three paths leaving from node* 1 *and going to node ñ after two branches, have accumulated Hamming weights* $14, 15,$ *and* 15. *However, each* $d_i = w_{1i} + w_{i\tilde{n}}$. *Hence, from the modular representation of block codes a possible solution to this combinatorial problem is*

$$w_{11} = 7, \quad w_{12} = 8, \quad \textit{and} \quad w_{13} = 7,$$

$$w_{1\tilde{n}} = 7, \quad w_{2\tilde{n}} = 7, \quad \textit{and} \quad w_{3\tilde{n}} = 8,$$

and the corresponding submatrices are

$$G_0 = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix},$$

$$G_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Note that the solution of Problem 3 provided the number of codewords with Hamming distances d_1, d_2, and d_3. On the other hand, the solution of Problem 4 provided the submatrices that generate the codewords with the specified Hamming distances from Problem 3. However, if this was not the case, then we would have to decrease each d_i by 1 *and solve Problems 3 and 4 again. This procedure ends up only when the solution of Problem 4 matches the solution of Problem 3. Palazzo, has extended this approach to unit-memory CCs.*

13.3 Search algorithm

Example 7. *Let us consider the case of* $m = 3$ *and rate* $r = 1/5$ *as a complete example of this algorithm. The maximum flow is* $\Phi = 5$*, and the upper bound to* $d_{\min}$ *is 16. In Fig. 24, it is shown the tree associated with this encoder. Since* $\Phi = 5$*, the transition from the root node to node "1" equals 5, and also from node 4 to node "0". Now, the judicious choice is like this, let us assume* $d_{\min} = 16$*. Therefore, the transitions* $1 \rightarrow 2$ *and* $2 \rightarrow 4$ *should add to 6. Let us choose* $w_{12} = w_{24} = 3$ *. From this selection results the following branch flows as shown in Fig. 24.*

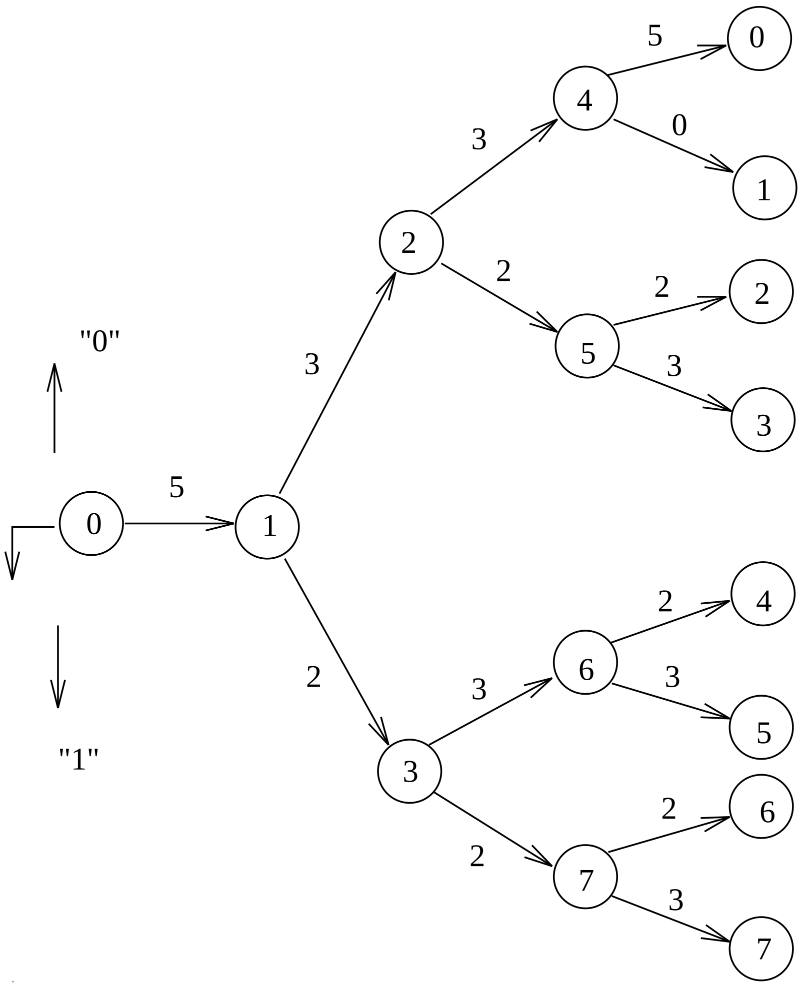


Figure 24: Tree associated with the convolutional encoder with $r = 1/5$ and $m = 3$.

Now comes the algorithm: by setting $j = 0$ *the accumulated Hamming weight is* $\alpha_4 = 16$*. The value of* $\alpha_5 = 16$ *and so step 2 of the algorithm is satisfied. Now, the value of* $\alpha_6 = 18$ *in step 3. From step 4 we have that* $\alpha_6 > \alpha_5$*. Setting* $j = 1$ *and going to step 3 we find* $\alpha_7 = 22$*. From step 4, we have* $\alpha_7 > \alpha_6$*. Now, keeping repeating this procedure until a desired value of* α *is achieved. The encoder associated with this resulting tree has tap connections between the 4 cells and the 5 mod 2 adders represented by*

$$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{array}$$

which corresponds in octal representation to $13:15:17:17:11$.

14 Unit-Memory and Partial Unit-Memory Codes

Let the binary k_0-tuple $\mathbf{u}_t$ denote the sub-block of input information digits at time $t = 0, 1, ...$ in an (n_0, k_0) TICC, and let the binary n_0-tuple $\mathbf{v}_t$ denote the encoded sub-block at time t. Then the encoding equation is:

$$\mathbf{v}_t = \mathbf{u}_t G_0 \oplus \mathbf{u}_{t-1} G_1 \oplus .... \oplus \mathbf{u}_{t-M} G_M, \tag{28}$$

where each G_i is a $k_0 \times n_0$ binary matrix, referred to as the generator submatrix, M is the encoder memory, $\oplus$ is the addition $mod\ 2$, and by convention, $\mathbf{u}_t = \mathbf{0}$ for $t < 0$.

Lee showed that the $(n = Mn_0, k = Mk_0)$ CC with memory $M' = 1$ defined by

$$G'_0 = \begin{bmatrix} G_0 & G_1 & .... & G_{M-1} \\ 0 & G_0 & .... & G_{M-2} \\ . & . & .... & . \\ 0 & 0 & .... & G_0 \end{bmatrix}, \qquad G'_1 = \begin{bmatrix} G_M & 0 & .... & 0 \\ G_{M-1} & G_M & .... & 0 \\ . & . & .... & . \\ G_1 & G_2 & .... & G_M \end{bmatrix}, \tag{29}$$

is equivalent to the code in (28) in the sense that the same semi-infinite binary encoded sequence is associated with the same semi-infinite information sequence. Lee called the codes of (29) (complete) unit-memory codes. Note from (29) that these codes essentially exchange memory by input information bits.

The other parameter that has to be specified is the free distance, d_{free}, that is,

$$d_{free} = \min\left\{W(\mathbf{v}_{[0,\infty]})\right\},$$

where the minimum is taken over all $\mathbf{u}_{[0,\infty)}$ such that $\mathbf{u}_0 \neq \mathbf{0}$, and where $W(.)$ denotes the Hamming weight.

The partial unit-memory codes as defined by Lauer differ essentially from the unit-memory codes defined by Lee with respect to the state-complexity. While Lee's codes have state-complexity Mk_0, Lauer's codes have state-complexity $\alpha = rank(G_1')$. Note that if G_1' has rank α then $(Mk_0 - \alpha)$ rows of G_1' are all-zero.

14.1 Design of UMC codes

It is known that unit-memory codes (UMC) may have a better error correcting capability than the usual multi-memory CCs with the same rate and number of memory elements.

We introduce a UMC design method based on mathematical optimization models with the purpose of providing a systematic way of finding these codes. It is shown that the problem can be decomposed into two subproblems which can be formulated as standard optimization problems with integer variables.

1. Find the sets of generator matrix columns that form distinct $(2n,k)$ BCs with minimum distance greater than or equal to d^*_{free} (**column types selection problem**);
2. Use these sets of columns to find the pairwise combination of the columns in G_0 and G_1 that forms a UMC with $d_{free} = d^*_{free}$ (**column types matching problem**);

14.2 The Column Types Selection Problem

The objective function to be minimized should add a penalty for each occurrence of a codeword with weight constraints (32) , and when all those constraints are satisfied it should count the number of codewords with minimum weight. If we set d_{free} as a parameter, then one function that can be used in the objective function is

$$\nu(x) = \begin{cases} \left|d_{free}+1-x\right|^r, & \text{if } x \leq d_{free} \\ \\ 0, & \text{otherwise} \end{cases} \tag{30}$$

The exponent $r > 1$ is usually 2 or 3, but a greater value may be required to ensure that constraints (32) are satisfied.

Hence, we can formulate the column types selection problem as the following nonlinear integer optimization problem

Problem 5.

$$Minimize \left\{ \pi(\mathbf{c}) = \sum_{i=1}^{N} \nu(w_i) \right\} \tag{31}$$

such that

$$\mathbf{Hc} = \mathbf{w} \tag{32}$$

$$\mathbf{1}^t \mathbf{c} = 2n \tag{33}$$

$$c_i \in \{0, 1, 2, 3, ..., 2n\} \quad for \quad i = 1, 2, ..., N \tag{34}$$

Equation (32) represents a change of variables and constraints (33) ensures that exactly $2n$ column types are selected. Note that excluding constraints (34) this optimization problem has a convex feasible solution space and its objective function is convex $\cup$.

Algorithm I - Column Type Selection Problem

1. Choose an initial solution $\mathbf{c}$, satisfying constraints (33), and (26). A random selection can provide fairly good initial solutions;
2. Find $i \notin I$ such that $c_i \geq 1$ and $j \notin O$ that minimize the objective function of Problem (P1): $\pi(\mathbf{c} + \delta(i, j))$;
3. Update the solution vector $\mathbf{c} \leftarrow \mathbf{c} + \delta(i, j)$ and the tabu lists I and O. Save $\mathbf{c}$ if it is better than the best solution already found;
4. If a "good solution" $\mathbf{c}$ is found satisfying $\mathbf{Hc} = \mathbf{w} \geq d^*_{free}\mathbf{1}$, then stop Algorithm I. Otherwise, return to step 2.

In the last step of Algorithm I, we have used the rather vague term "good solution" to allow considering the number of codewords with minimum weight. Algorithm I can be interpreted as a progressive enhancement of a BC via column types substitutions. It is expected to give better results than , for example, substitution of rows or single elements of the generator matrix because it is based on the properties of Problem (P1).

14.3 The column types matching problem

From the solution provided by Algorithm I we can construct the set $\{t_1, t_2, ..., t_{2n}\}$, where each t_i corresponds to a selected column type. The variables of the matching problem can be represented by a $2n \times 2n$ matrix A such that

$$A_{pq} = \begin{cases} 1, & \text{if column types } t_p \text{ and } t_q \text{ form a pair with } t_p \text{ in } G_0 \\ & \text{and } t_q \text{ in } G_1 \ (t_p + t_q.D \text{ is a column of } G(D) \\ & \\ 0, & \text{otherwise} \end{cases}$$

To calculate the Hamming weight of the codewords from the matching matrix A let us define $\mathbf{w}_{ij}$ as the weight of the output byte associated with the transition from state i to state j. The individual contribution of matched columns t_p and t_q, to the output byte weight $\mathbf{w}'_{(ij)(pq)} \in GF(2)$, for $p, q = 1, 2, ..., 2n$, can be calculated from

$$\mathbf{w}'_{(ij)(pq)} = \left[t_q^t \otimes b_k(i)\right] \oplus \left[t_p^t \otimes b_k(j)\right],$$

where α^t is the transpose of α.

Assuming that we have a feasible solution A, the next step is to employ the Viterbi algorithm to calculate the minimum distances in the trellis over S stages. The result is the truncated distance profile $\{p_2(A), p_3(A), ..., p_S(A)\}$. It is also possible to find the number of codewords of length L with weight equal to $p_L(A)$. These numbers define the sequence $\{\eta_2(A), \eta_3(A), ..., \eta_S(A)\}$. In order to find the best solution possible, a "reference" distance profile defined by $\{r_2, r_3, ..., r_S\}$, with the expected optimum values of $p_L(A)$ is also needed.

The objective function should be a type of distance profile evaluation function that adds a penalty whenever $p_L(A) < r_L$. A generalization of the function given by (30) can be defined as

$$\Gamma(x,y) = \begin{cases} |y+1-x|^r, & \text{if} \quad x \leq y \\ \\ 0, & \text{otherwise} \end{cases} \tag{35}$$

The inclusion of this penalty function into the formulation of the optimization problem leads to

Problem 6.

$$Minimize\ \xi(A) = \sum_{m=2}^{S} \alpha_m \eta_m(A) \Gamma(p_m(A), r_m) \tag{36}$$

such that

$$\sum_{j=1}^{2n} A_{ij} + \sum_{j=1}^{2n} A_{ji} = 1, \quad for \quad i = 1, 2, ..., 2n \tag{37}$$

$$A_{ij} \in \{0, 1\}, \quad for \quad i, j = 1, 2, ..., 2n \tag{38}$$

Constraints (37) state that each column type can not be simultaneously in G_0 and in G_1 and should be matched to only one column type. The parameters α_m in the objective function are used primarily as penalty "weights" for each component of the distance profile with the purpose of enforcing the maximization of the free distance and, secondarily, to maximize the distance profile growth rate.

Algorithm II - Column Types Matching Problem

1. Build up matrices G_0 and G_1 with the column types provided by Algorithm I from a random matching;

2. For each column $i = 1, 2, ..., 2n$, find the column $j \neq i$ such that if interchanged with column i, will minimize the objective function (**??**), that is, $\xi(A)$;

3. If the objective function decreases, then interchange columns i and j and go back to step 2, otherwise go to step 4;

4. If a "good" solution is found (at least $d_{free} = d^*_{free}$) then stop, else go to step 1 to restart the search from another initial solution.

15 Punctured Convolutional Codes

If the constraint is no bandwidth expansion a high-rate code has to be used. However, finding good high-rate codes is a combinatorial optimization problem which is classified as an $NP-complete$ problem.

In order to overcome this difficulty, a technique called **puncturing** is employed. This technique is based on the fact that by properly deleting certain digits of a low-rate code a desired high-rate code is obtained in order to meet the bandwidth requirements.

Consider a CC with memory m. We use the polynomial matrix representation for this code, that is, $G(D)$. The element (j,i) of $G(D)$ is represented by

$$G_j^i(D) = g_{0,j}^i + g_{1,j}^i D + g_{2,j}^i D^2 + \cdots + g_{m,j}^i D^m,$$

where $G_j^i(D)$ denotes the polynomial element associated with the j-th row and i-th column. Hence, relating the i-th output sequence with the j-th input sequence.

As an example, consider a CC with rate $r = 2/3$ and $m = 1$. Its generator matrix in polynomial form is given by

$$
\begin{array}{c}
\longrightarrow output \\
G(D) = \begin{bmatrix} 1+D & 1+D & 1 \\ D & 0 & 1+D \end{bmatrix}, \\
\uparrow input
\end{array}
$$

the encoder representation is as shown in Fig. 25. Note that the degree of each polynomial is at most equal to 1. This is due to $m = 1$.

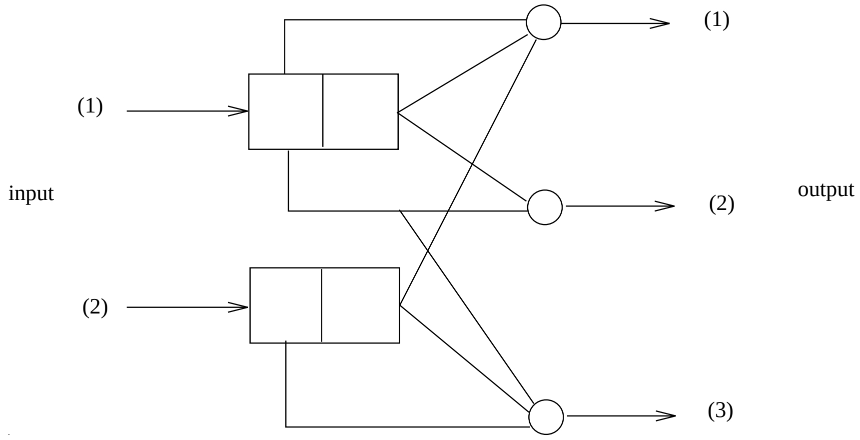


Figure 25: Encoder representation, $r = 2/3$, $m = 1$.

The corresponding trellis is as shown in Fig.26.

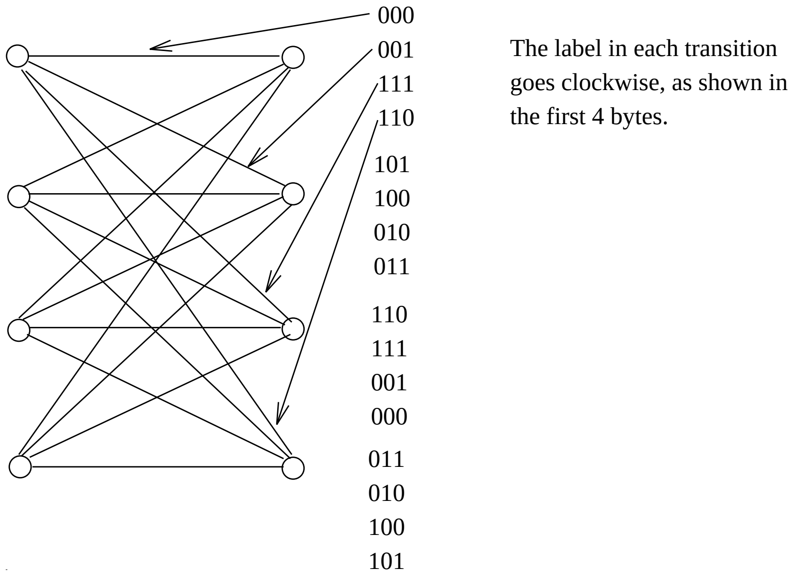


Figure 26: Trellis section of the previous encoder, $r = 2/3$, $m = 1$.

Note that at each state the Viterbi algorithm realizes a 4-*ary* comparison.

Consider now a convolutional encoder with rate $r = 1/2$ and $m = 2$ with generator matrix, in polynomial form, given by

$$G(D) = \left[\begin{array}{cc} 1 + D + D^2 & 1 + D^2 \end{array} \right].$$

The corresponding trellis is as shown in Fig. 27.

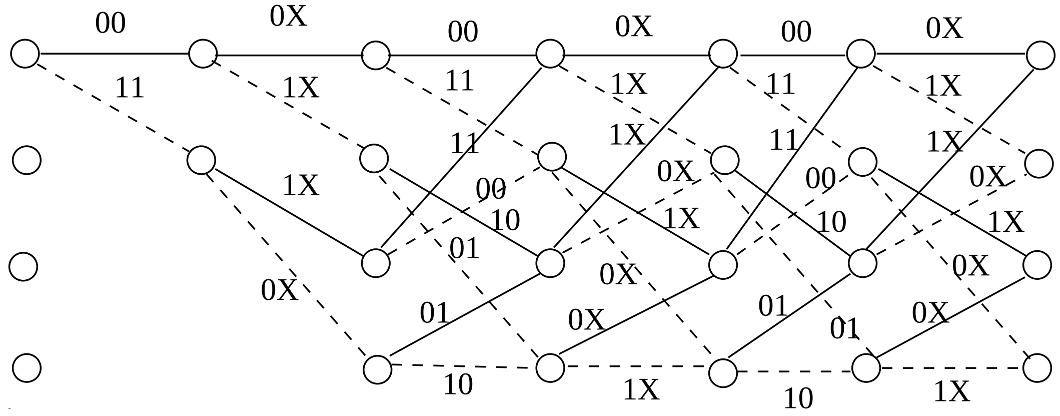


Figure 27: Trellis structure of the CE with $r = 1/2$ and $m = 1$.

If for every output sequence consisting of four coded digits, the $4\text{-}th$ digit is deleted, the resulting code will reproduce the 3 encoded digits as shown in Fig.26. Note that for every 2 input information bits we have a 3 encoded output bits, the code rate is $r = 2/3$. The same result is met if the 2nd encoded bit with respect to the 2nd transition is periodically deleted. Notice that after 2 transitions in the trellis we end up with the same branch codewords as those shown in Fig. 26.

This very same result is achieved if we employ a convolutional encoder with rate $r = 1/3$ and $m = 2$ with generator matrix, in polynomial form, given by

$$G(D) = \left[\begin{array}{ccc} 1+D+D^2 & 1+D^2 & 1+D+D^2 \end{array} \right],$$

where the first two encoded bits are deleted in a transition, and the third encoded digit is deleted in the next transition in the trellis. This procedure leads to the trellis of the convolutional encoder with rate $r = 2/3$ and $m = 1$.

Example 8. *Let us consider the case where we want to find a punctured unit-memory code with rate $r = 5/6$. For that, let us assume the mother code has rate $1/2$ and $M = 5$. From the action of puncturing, with $h = M = 5$, we end up with a unit-memory code with rate $r = 5/10$. Now, instead of employing a 5×20 generator matrix $[G_0' : G_1']$ for this unit-memory code and after that to search for the appropriate columns to be deleted, we simply bypass this step by employing a 5×12 generator matrix $[G_0' : G_1']$ of a block code. This procedure is incorporated in the combinatorial optimization search algorithm.*

From the previous arguments, we are ready to establish an upper bound on the free distance of punctured unit-memory CCs. To the best of our knowledge, this upper bound was not given previously.

Let d_{fp} denote the free distance of a punctured unit-memory code. Then

$$d_{fp} \le d(\{2M(n_0-1)+2i\},\, Mk_0) = d(N,\, K),$$

where $d(.,.)$ is the best minimum distance of an (N, K) linear block code, and $1 \le i \le M$.

As a consequence of the previous reasonings, we have

Proposition 1. *For a given state-complexity and a given rate, the maximum value of the free distance is achieved by the punctured unit-memory codes within the subset of punctured CCs.*

15.1 A new criterion for puncturing CCs

We present this rule by examples in order to avoid cumbersome notations.

Example 9. *Let us consider the best CC with parameters* $(n_0, k_0, M) = (2, 1, 2)$ *and* $d_{free} = 5$ *as the mother code. Its semi-infinite generator matrix is*

$$G = \begin{bmatrix} 11 & 01 & 11 & . & \dots & . \\ . & 11 & 01 & 11 & \dots & . \\ . & . & 11 & 01 & 11 & \dots \\ . & . & . & . & . & . \end{bmatrix}.$$

Let us derive the best punctured unit-memory CC with $r = 2/3$ *from this mother code. Since* $h = M = 2$*, the resulting code is a* $(4, 2)$ *unit-memory code with submatrices* G'_0 *and* G'_1 *given by*

$$G'_0 = \begin{bmatrix} 11 & 01 \\ 00 & 11 \end{bmatrix}, \qquad G'_1 = \begin{bmatrix} 11 & 00 \\ 01 & 11 \end{bmatrix}.$$

Let $\mathbf{u} = (u_1, u_2)$ *be the information byte to be encoded. Let* $\mathbf{v} = (v_1, v_2, v_3, v_4)$ *be the corresponding encoded byte. The split state diagram of this* $(4,2)$ *encoder with its transfer function weights is shown in Fig. 28.*

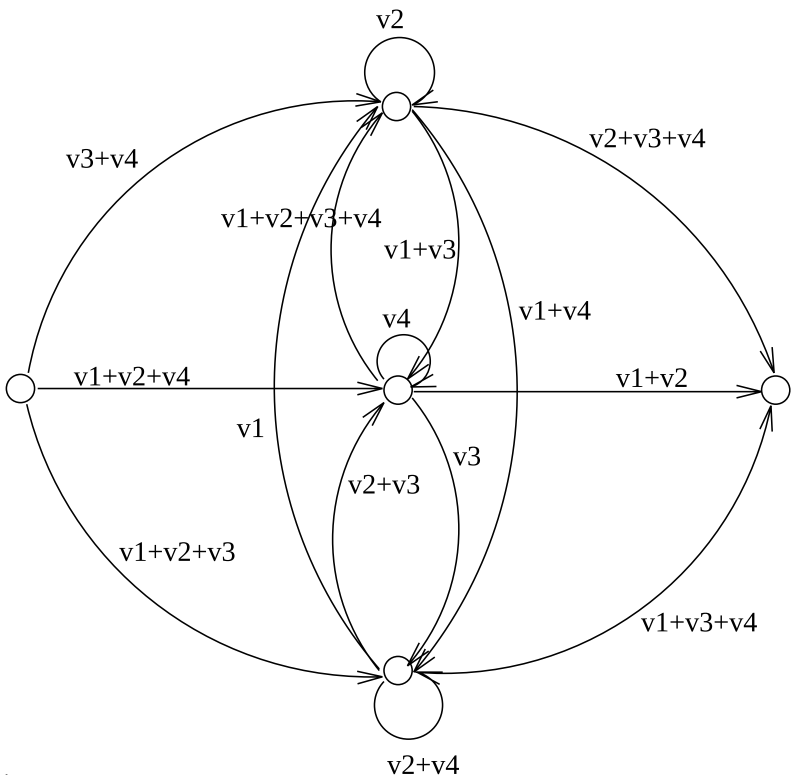


Figure 28: Split state diagram of the $(4,2)$ *encoder.*

Using the standard procedures for determining the transfer function of a convolutional encoder, we have

$$T(D, v_1, v_2, v_3, v_4) = D^{v_2+2v_3+2v_4} + D^{2v_1+2v_2+v_4} + D^{2v_1+v_2+2v_3+v_4} + \ldots. \quad (39)$$

Now the optimality criterion to be employed is the minimum encoded bit error probability, P_{benc}. It is a simple matter to show that an upper bound on P_{benc} is given by

$$P_{benc} \leq (1/n)D\frac{\partial}{\partial D}T(D, v_1, v_2, v_3, v_4)|_{D,v_i},$$

where D is the Battacharyya function.

Since we are interested in determining the upper bound on P_{benc} for each v_i, i.e., when the v_i is the deleted bit, we have

$$P_{benc}(v_i) \leq (D/n)\frac{\partial}{\partial D}T(D, v_1, v_2, v_3, v_4)|_{D,v_i=0,v_j=1,\, all\ j,j\neq i} \quad (40)$$

Substituting (39) in (40), and after some algebraic manipulations, subject to $v_i = 0$ (deletion) and $v_j = 1$ for all j, $j \neq i$, and $i = 1, 2, 3, 4$ we have the following accumulated values for two and three transitions in the trellis

After Two Transitions		After Three Transitions	
1)	$v_2 + 2v_3 + 2v_4$	4)	$2v_2 + 2v_3 + 2v_4$
2)	$2v_1 + 2v_2 + v_4$	5)	$2v_1 + v_2 + 2v_3 + v_4$
3)	$2v_1 + v_2 + 2v_3 + v_4$	6)	$2v_1 + 2v_3 + 3v_4$
		7)	$2v_1 + 3v_2 + 2v_3 + 3v_4$
		8)	$2v_1 + 2v_2 + 2v_4$
		9)	$2v_1 + v_2 + 2v_3 + 2v_4$
		10)	$2v_1 + 2v_2 + 2v_3 + v_4$
		11)	$2v_1 + 3v_2 + 2v_3$
		12)	$2v_1 + 2v_2 + 2v_3 + 2v_4$

now, the accumulated weights are as shown below

	1)	2)	3)	4)	5)	6)	7)	8)	9)	10)	11)	12)
$v_1 = 0, v_2 = v_3 = v_4 = 1$	5	3	4	6	4	5	8	4	5	5	5	6
$v_2 = 0, v_1 = v_3 = v_4 = 1$	4	3	5	4	5	7	7	4	6	5	4	6
$v_3 = 0, v_1 = v_2 = v_4 = 1$	3	5	4	4	4	5	8	6	5	5	5	6
$v_4 = 0, v_1 = v_2 = v_3 = 1$	3	4	5	4	5	4	7	4	5	6	7	6

Therefore, we end up with the following upper bounds on the encoded bit error probabilities

$$P_{benc}(v_1) \leq (3/4)D^3 + 3(4/4)D^4 + 5(5/4)D^5 + 2(6/4)D^6 + ...,$$

$$P_{benc}(v_2) \leq (3/4)D^3 + 4(4/4)D^4 + 3(5/4)D^5 + 2(6/4)D^6 + ...,$$

$$P_{benc}(v_3) \leq (3/4)D^3 + 3(4/4)D^4 + 5(5/4)D^5 + 2(6/4)D^6 + ...,$$

$$P_{benc}(v_4) \leq (3/4)D^3 + 4(4/4)D^4 + 3(5/4)D^5 + 2(6/4)D^6 + ... \quad (41)$$

where each term $a(b/c)D^b$ denotes the number of times b have appeared, b denotes the Hamming weight, and c denotes the blocklength.

Since we aim at finding the best upper bound on P_{benc} for each v_i, this implies that the deletion of anyone of the encoded bits v_1 or v_2 or v_3 or v_4, has to lead to the largest possible value of the exponent of D, however, with the lowest possible coefficient.

Therefore, from (41) we conclude that deleting v_1 or v_3 yields the least upper bound. Notice that the deletion of v_3 is the one already shown by Yasuda and Hole.

Example 10. *Let us consider the best CC with parameters $(n_0, k_0, M) = (2, 1, 3)$ and $d_{free} = 6$ as the mother code. Its semi-infinite generator matrix is*

$$G = \begin{bmatrix} 11 & 11 & 01 & 11 & . & . & . \\ . & 11 & 11 & 01 & 11 & . & . \\ . & . & 11 & 11 & 01 & 11 & . \\ . & . & . & . & . & . & . \end{bmatrix}.$$

Let us derive the best punctured unit-memory CC with $r = 3/4$ from this mother code. Since $h = M = 3$, the resulting code is a $(6, 3)$ unit-memory code with submatrices G'_0 and G'_1 given by

$$G_0' = \begin{bmatrix} 11 & 11 & 01 \\ 00 & 11 & 11 \\ 00 & 00 & 11 \end{bmatrix}, \qquad G_1' = \begin{bmatrix} 11 & 00 & 00 \\ 01 & 11 & 00 \\ 11 & 01 & 11 \end{bmatrix}.$$

Let $\mathbf{u} = (u_1, u_2, u_3)$ *be the information byte to be encoded. Let* $\mathbf{v} = (v_1, v_2, v_3, v_4, v_5, v_6)$ *be the corresponding encoded byte.*

From the split state diagram representation of this $(6,3)$ *encoder and the use of the transfer function techniques, we have*

$$\begin{aligned} T(D, v_1, v_2, v_3, v_4, v_5, v_6) = \; & D^{v_1+v_2+v_4+2v_5+2v_6} + D^{v_2+2v_3+2v_4+v_5+v_6} + D^{v_1+2v_3+v_4+v_5+v_6} \\ & + D^{2v_1+2v_2+v_3+v_4+v_6} + D^{v_1+v_2+v_3+2v_4+2v_5+v_6} \\ & + D^{2v_1+v_2+v_3+v_4+v_5} + D^{v_1+2v_2+v_3+v_5+2v_6} + \cdots. \end{aligned}$$

Similar to the procedure followed in the previous example we have that, after some algebraic manipulations, the least upper bound on the encoded bit error probabilities for each 2-tuple (v_i, v_j) subject to $v_i = 0$ and $v_j = 0$, $j \neq i$ deletions and $v_k = 1$ for the remaining values that k can take on the set $\{1,2,3,4,5,6\}$, is $P_{benc} \leq \alpha D^4 + \cdots$. This result comes from the deletion of columns 1 and 6, (v_1, v_6), or column 4 and 5, (v_4, v_5). This latter result is the same as shown by Yasuda and Hole.

TABLE I

Parameters of the New Codes

k	n	rate	l_b	u_b	d_{free}	w_0	GV_b	Gain	$N(d_{free})$
3	4	.75	4	4	4	.40	.17	1.76	10
4	5	.80	4	4	4	.28	.16	2.04	9
5	6	.83	4	4	4	.29	.15	2.22	10
6	7	.86	4	5	4	.25	.14	2.34	6
7	8	.88	4	6	4	.19	.14	2.43	2
8	9	.89	4	6	4	.13	.13	2.50	4

Table 1: Parameters of the new codes

TABLE III

Comparative Values of d_{free} and $N(d_{free})$

			New Codes		*Hole*		*Yasuda et al*		*Cain et al*	
k	*n*	u_b	d_{free}	$N(d_{free})$	d_{free}	$N(d_{free})$	d_{free}	$N(d_{free})$	d_{free}	$N(d_{free})$
3	4	4	4	10	–	–	4	124	4	124
4	5	4	4	9	4	312	3	11	–	–
5	6	4	4	10	4	100	4	100	–	–
6	7	5	4	6	4	122	3	5	–	–
7	8	6	4	2	4	472	4	258	–	–
8	9	6	4	4	–	–	4	124	–	–

Table 2: Comparative values

Chapter 4

Periodic Time-Varying Convolutional Codes[a]

17. Objectives
18. Generator Matrix of PTVCC
19. PTVC Encoder
20. Modified Transfer Function for TVCC
21. Distance Properties of PTVCC
22. Good PTVCC

[a] Most of the content of this chapter may be found in the papers by Moser, Palazzo, Mills/Costello, etc.

16 Objectives

- **Determination of periodically time-varying convolutional codes (PTVCC)** from specific convolutional codes.
- **Derivation of the enumerator function of PTVCC** by use of the modified transfer function technique.
- **Comparison** of the (optimum) time-varying convolutional codes with the (optimum) time-invariant convolutional codes.

17 Generator Matrix of PTVCC

The set of (N,R,D,M,m)-μ-ary trellis codes for a DMC channel is sufficiently general for a characterization of the PTVCC.

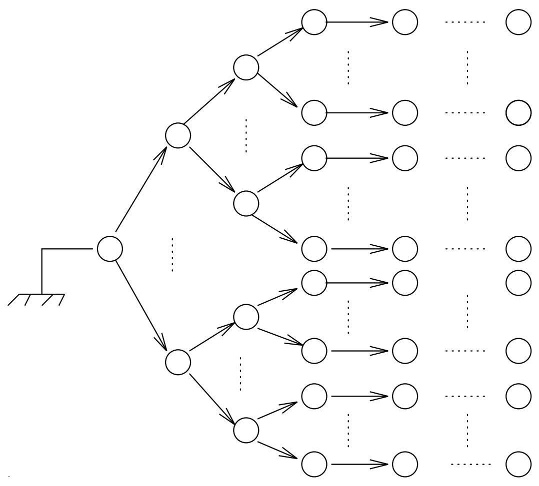


Figure 29: $A=(D,M)$-μ-ary tree

Let $\{\alpha_j\}_{j=0}^{D-1}$ be a sequence of information symbols to be encoded. Each α_j is associated with a trellis branch starting in the root node and ending in a node at depth D. Let $\{\beta_j\}_{i=0}^{D+M-1}$ be the corresponding encoded sequence where $\alpha_j = \mathbf{0}$, for $D \leq i \leq D+M$.

The mathematical representation of the encoder is given by

$$\beta_i = \Gamma(\alpha_i, \alpha_{i-1}, \alpha_{i-2}, ..., \alpha_{i-m}, i).$$

When $\Gamma(.,i)$ varies with time the encoder is said to be **time-varying**.

We consider $\Gamma(.,i)$ as a **linear function**. Furthermore, we consider the **information digits** and the **encoded digits** have an associated algebraic structure.

Suppose the input alphabet of the DMC channel belongs to $GF(q)$, where q is a prime or a prime power. Let $\mu = q^L$ be such that α_i is an L-tuple over $GF(q)$.

The rate of this code is given by $R = (\log \mu)/N = (L/N) \log q$, bits/channel use.

The (N,R,D,M,m) TVCC with $\beta_i = \Gamma(\alpha_i, \alpha_{i-1}, \alpha_{i-2}, ..., \alpha_{i-m}, i)$ is such that $\alpha_i \in GF(q)^L$ and $\beta_i \in GF(q)^N$, and $\Gamma(.,i)$ are linear functions given by

$$\Gamma(.,i): \; GF(q)^{(m+1).L} \quad \rightarrow \quad GF(q)^N.$$

Such functions may be represented mathematically by

$$\beta_i = \alpha_i G_0(i) \oplus \alpha_{i-1} G_1(i) \oplus \ldots \oplus \alpha_{i-m} G_m(i), \tag{42}$$

where the $G_j(i)$s are $L \times N$ matrices whose elements are over $GF(q)$, and $\oplus$ denotes the addition operation in $GF(q)$. Note that equation (42) may be represented as shown in Fig. 30.

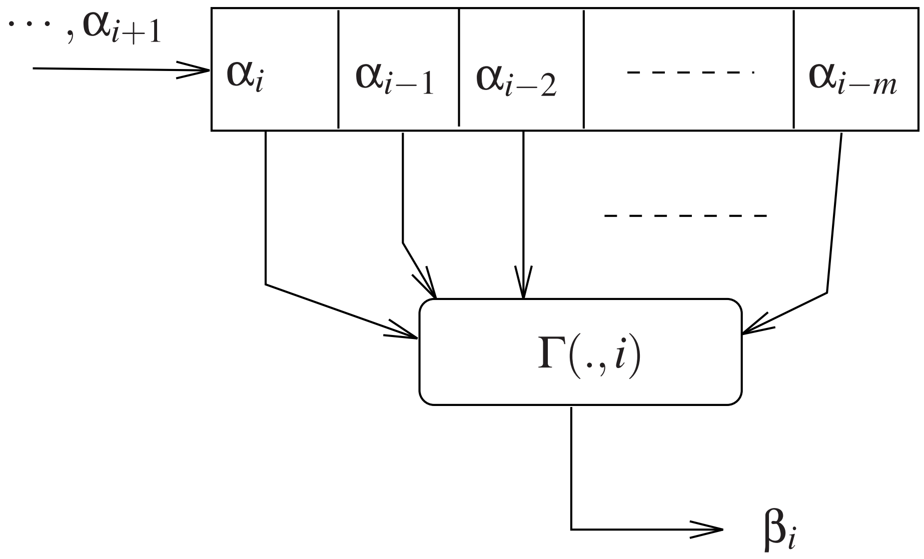


Figure 30: Convolutional encoder of (42).

To have a matrix representation for the generator matrix of the TVCC, let $\alpha_{[u,v]}$ and $\alpha_{[u,v)}$ be sequences such that $\alpha_{[u,v]} = (\alpha_u, \alpha_{u+1}, ..., \alpha_v)$ and $\alpha_{[u,v)} = (\alpha_u, \alpha_{u+1}, ..., \alpha_{v-1})$. Let $\beta_{[u,v]}$ and $\beta_{[u,v)}$ be sequences similarly defined as $\alpha_{[u,v]}$ and $\alpha_{[u,v)}$. Hence,

$$\beta_{[0,D+M)} = \alpha_{[0,D)} \begin{bmatrix} G_0(0) & G_1(1) & \ldots\ldots & G_m(m) & . & . & . \\ . & G_0(1) & \ldots\ldots & G_{m-1}(m) & G_m(m+1) & . & . \\ . & . & . & . & . & . & . \\ . & . & . & G_0(m) & G_1(m+1) & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & G_0(D-1) & . & G_M(D+M-1) \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}.$$

Thus, G is called **generator matrix** of the (N,R,D,M,m) time-varying convolutional code.

18 PTVCC Encoder

The objective of this section is to present the encoding process of time-varying convolutional codes under the point of view of dynamical linear systems. We consider the realization of the PTV convolutional encoder as the composition of two time-invariant convolutional codes each with memory $m = 2$ and rate $r = 1/2$ as shown in Fig. 31.

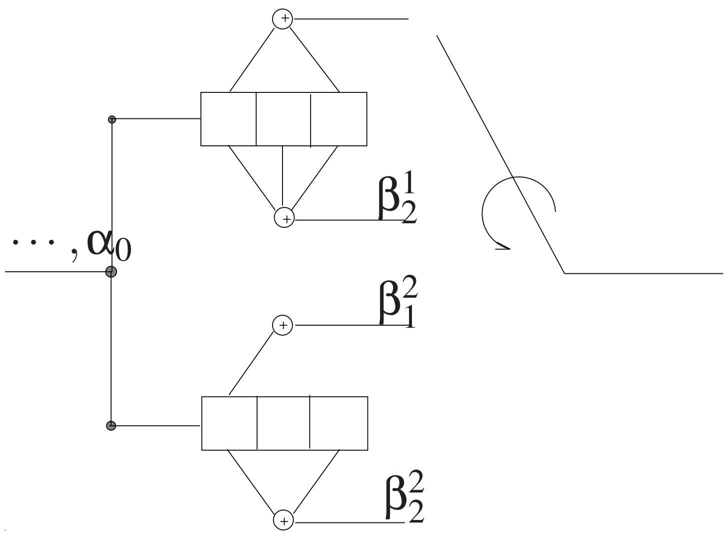


Figure 31: Binary CE_1 (top) and CE_2 (down) with $m = 2$ and $r = 1/2$.

Let $\alpha = \alpha_{[0,\infty)}$ be an information sequence, that is, $\alpha = (\alpha_0, \alpha_1, \alpha_2, ...)$ where each $\alpha_i \in GF(2)$. The corresponding encoded sequences β^1 and β^2, are given, respectively, by

$$\beta^1 = (\beta_1^1(0), \beta_2^1(0), \beta_1^1(1), \beta_2^1(1), \beta_1^1(2), \beta_2^1(2)...),$$

and

$$\beta^2 = (\beta_1^2(0), \beta_2^2(0), \beta_1^2(1), \beta_2^2(1), \beta_1^2(2), \beta_2^2(2)...).$$

These sequences are generated by the convolution between the information sequence α and the **generating functions** $\mathbf{g}_{1i}$ and $\mathbf{g}_{2i}$, where $i = 1, 2$, respectively. The **generating functions** $\mathbf{g}_{1i}$ and $\mathbf{g}_{2i}$ are obtained by the application of the following information sequences to the input of each one of the encoders,

$$\alpha = (1, 0, 0); \quad \alpha = (0, 1, 0); \quad \alpha = (0, 0, 1).$$

For the encoders in consideration, the generating functions $\mathbf{g}_{1i}$ and $\mathbf{g}_{2i}$ are given, respectively, by

$$\mathbf{g}_{11} = (1,0,1) \qquad ; \qquad \mathbf{g}_{12} = (1,1,1),$$

and

$$\mathbf{g}_{21} = (1,0,0) \qquad ; \qquad \mathbf{g}_{22} = (1,0,1),$$

where "1" means there exists a connection between the mod 2 sum and the corresponding cells of the shift-register, and "0" otherwise.

Since $\mathbf{g}_{1i}$ and $\mathbf{g}_{2i}$ are generating functions, we have to specify how the multiplexer will collect the output sequences of the mod 2 sum of each encoder.

This procedure is called **encoding pattern**. Once the **encoding pattern** is known, the corresponding encoded sequence is

$$\beta^1 = \alpha * \mathbf{g}_{1i} \quad \text{and} \quad \beta^2 = \alpha * \mathbf{g}_{2i},$$

where $*$ denotes convolution. Hence, the encoded sequence is given by

$$\beta^1 = (\beta_{11}(0)\beta_{12}(0), \beta_{11}(1)\beta_{12}(1), \beta_{11}(2)\beta_{12}(2), \ldots),$$

and

$$\beta^2 = (\beta_{21}(0)\beta_{22}(0), \beta_{21}(1)\beta_{22}(1), \beta_{21}(2)\beta_{22}(2), \ldots),$$

where $\beta_{ij}(k)$ denotes the encoded output with respect to the convolutional encoder CE_i, $i = 1,2$, where the index j indicates if it is relative to the generating function g_{1j} or g_{2j}, $j = 1,2$, at time $t = k$.

Considering the encoders shown in Fig. 31, at time $t = k$, β^1 and β^2, and that the encoding pattern is periodic with period 2, then the encoding process may start with any one of the encoders, and so we have

$$\begin{cases} \beta_{11}(k) = \alpha(k) \oplus \alpha(k-2) \\ \\ \beta_{12}(k) = \alpha(k) \oplus \alpha(k-1) \oplus \alpha(k-2) \end{cases}$$

and

$$\begin{cases} \beta_{21}(k) = \alpha(k) \\ \\ \beta_{22}(k) = \alpha(k) \oplus \alpha(k-2) \end{cases},$$

respectively.

The generator matrix G^1 may be placed in the semi-infinite form if the encoding pattern starts with the convolutional encoder CE_1 followed by the convolutional encoder CE_2, and so on,

$$G^1 = \begin{bmatrix} \overline{g}_{10} & \overline{g}_{21} & \overline{g}_{12} & \cdot & \cdot & \cdot \\ \cdot & \overline{g}_{20} & \overline{g}_{11} & \overline{g}_{22} & \cdot & \cdot \\ \cdot & \cdot & \overline{g}_{10} & \overline{g}_{21} & \overline{g}_{12} & \cdot \\ \cdot & \cdot & \cdot & \overline{g}_{20} & \overline{g}_{11} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \overline{g}_{10} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix},$$

where $\overline{g}_{10} = (1,1)$; $\overline{g}_{11} = (0,1)$; $\overline{g}_{12} = (1,1)$ e $\overline{g}_{20} = (1,1)$; $\overline{g}_{21} = (0,0)$; $\overline{g}_{22} = (0,1)$.

In case the encoding pattern starts with the convolutional encoder CE_2 followed by the convolutional encoder CE_1, the generator matrix G^2 is given by

$$G^2 = \begin{bmatrix} \overline{g}_{20} & \overline{g}_{11} & \overline{g}_{22} & \cdot & \cdot & \cdot \\ \cdot & \overline{g}_{10} & \overline{g}_{21} & \overline{g}_{12} & \cdot & \cdot \\ \cdot & \cdot & \overline{g}_{20} & \overline{g}_{11} & \overline{g}_{22} & \cdot \\ \cdot & \cdot & \cdot & \overline{g}_{10} & \overline{g}_{21} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \overline{g}_{20} & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix},$$

where $\overline{g}_{20} = (1,1); \overline{g}_{21} = (0,0); \overline{g}_{22} = (0,1)$ and $\overline{g}_{10} = (1,1); \overline{g}_{11} = (0,1); \overline{g}_{12} = (1,1)$.

The encoded sequence is then written as

$$\beta = \alpha * G^i, \qquad i = 1,2$$

Note that $\overline{g}_{ij}$ may be seen as 1×2 submatrices. A generalization for rates $r = k/n$ is quite simple, for now $\overline{g}_{ij}$ are $k \times n$ submatrices with $1 \leq i \leq m+1$ and with subsequent shifts, for $u \leq j \leq v$.

19 Modified Transfer Function for TVCC

Under the assumption of **sequence independence** the **superstate representation reduces to the usual state representation**.

Consider two discrete-time linear systems L_1 and L_2 whose state and output equations are, respectively, given by:

$$\text{System } L_1 \quad \Longleftrightarrow \quad \begin{cases} \varepsilon_1(k+1) = A_1(k).\varepsilon_1(k) + B_1(k).u(k) \\ \\ T_1(k) = C_1(k).\varepsilon_1(k) \end{cases}$$

$$\text{System } L_2 \quad \Longleftrightarrow \quad \begin{cases} \varepsilon_2(k+1) = A_2(k).\varepsilon_2(k) + B_2(k).u(k) \\ \\ T_2(k) = C_2(k).\varepsilon_2(k) \end{cases}$$

where

- $u(k)$ is the **input**;
- $\varepsilon_i(k)$ is the $(M-1)\times 1$ **state matrix** (the intermediate states);
- $B_i(k)$ is the $(M-1)\times 1$ **initial condition matrix** (transitions between the zero state to the intermediate states);
- $C_i(k)$ is the $1\times(M-1)$ **output condition matrix** (transitions between the intermediate states to the zero state);
- $A_i(k)$ is the $(M-1)\times(M-1)$ **transition matrix** (transitions between intermediate states only);
- $T_i(k)$ is the **output**, at time $t = k$, $i = 1,2$.

The discrete-time evolution of these systems is as follows:

- System L_1 (convolutional encoder) is **on** in the time interval $i \leq k < i+1$;
- System L_2 (convolutional encoder) is **on** in the time interval $i+1 \leq k < i+2$, for any i an integer.

It can be shown the combination of these two linear systems results in a linear system.

The state and output equations of the resulting linear system can be described mathematically by the following reasoning:

- The states ε_1 (ε_2) belonging to system L_1 (L_2), at time $t = k+1$ can be reached from the states ε_2 (ε_1) belonging to system L_2 (L_1) at time $t = k$, by applying the transition matrix A_1 (A_2), corresponding to system L_1 (L_2), or from the initial condition matrix B_1 (B_2);
- the output equation T_1 (T_2) of system L_1 (L_2) is obtained by multiplying the output condition matrix C_1 (C_2) by the state matrix ε_2 (ε_1) of system L_2 (L_1).

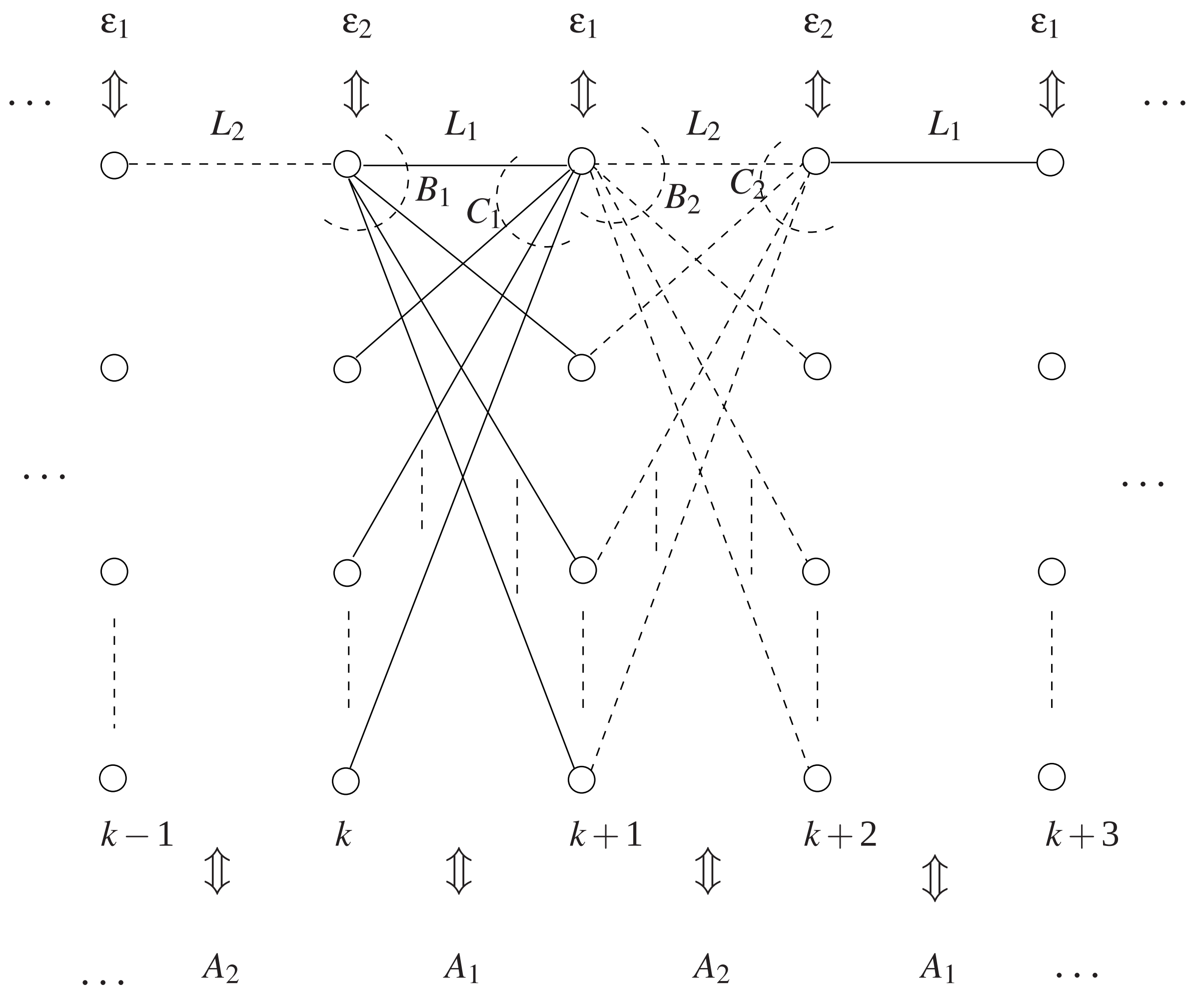


Figure 32: Trellis showing the state and output equations of a time-varying linear system

Mathematically,

$$\text{System } L_1 \Longleftrightarrow \begin{cases} \varepsilon_1(k+1) = A_1(k).\varepsilon_2(k) + B_1(k) \\ \\ T_1(k) = C_1(k).\varepsilon_2(k) \end{cases}$$

$$\text{System } L_2 \Longleftrightarrow \begin{cases} \varepsilon_2(k+1) = A_2(k).\varepsilon_1(k) + B_2(k) \\ \\ T_2(k) = C_2(k).\varepsilon_1(k) \end{cases}$$

The total output equation is

$$T(k) = T_1(k) + T_2(k)$$

Solving the equations for $\varepsilon_1(k)$ and $\varepsilon_2(k)$, we have

$$\varepsilon_1 = (I - A_1.A_2)^{-1}(A_1.B_2 + B_1)$$

$$\varepsilon_2 = A_2.(I - A_1.A_2)^{-1}(A_1.B_2 + B_1) + B_2$$

Substituting these solutions in $T_2(k)$ and $T_1(k)$, the total output $T(k)$ is completely determined, and is given by

$$T = C_1.A_2.(I - A_1.A_2)^{-1}(A_1.B_2 + B_1) + C_1.B_2 + C_2.(I - A_1.A_2)^{-1}(A_1.B_2 + B_1) \quad (43)$$

Let the systems L_1 and L_2 be characterized by the convolutional encoders $(3,2,1)$ with memory allocation vectors $\mathbf{M} = (1,1)$ and with corresponding submatrices

$$\mathbf{G}_0^{(1)} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} ; \mathbf{G}_1^{(1)} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$\mathbf{G}_0^{(2)} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} ; \mathbf{G}_1^{(2)} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Consider the following distortion table

$$\begin{array}{cccccc} d(u_k, \widetilde{u}_k) & 00 & 01 & 10 & 11 & \leftarrow \quad \widetilde{u}_k \\ 00 & 0 & \beta & \alpha & \alpha+\beta & \\ 01 & \beta & 0 & \alpha+\beta & \alpha & \\ 10 & \alpha & \alpha+\beta & 0 & \beta & \\ 11 & \alpha+\beta & \alpha & \beta & 0 & \\ \uparrow & & & & & \\ u_k & & & & & \end{array}$$

The split state diagram representations of the convolutional encoders are shown in Fig. 40.

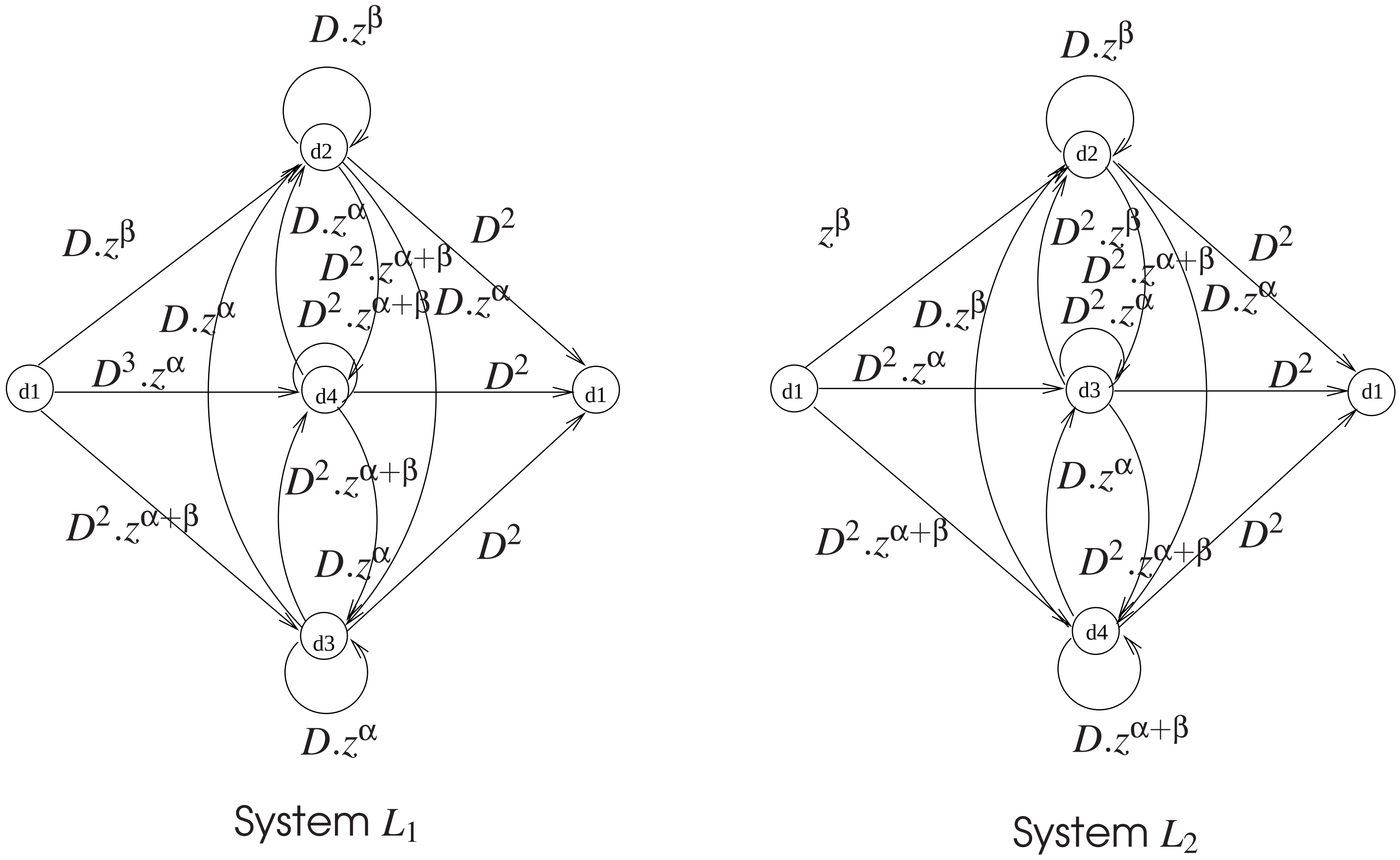


Figure 33: Split state diagram of the convolutional encoders.

From the split state diagrams, the corresponding (A, B, C) matrices are

$$A_1 = \begin{bmatrix} D.z^{\beta} & D^3.z^{\beta} & D.z^{\beta} \\ D.z^{\alpha} & D.z^{\alpha} & D.z^{\alpha} \\ D^2.z^{\alpha+\beta} & D^2.z^{\alpha+\beta} & D^2.z^{\alpha+\beta} \end{bmatrix} ; B_1 = \begin{bmatrix} D.z^{\beta} \\ D^3.z^{\alpha} \\ D^2.z^{\alpha+\beta} \end{bmatrix} ; C_1 = \begin{bmatrix} D^2 & D^2 & D^2 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} D.z^{\beta} & D^2.z^{\beta} & D^3.z^{\beta} \\ D.z^{\alpha} & D^2.z^{\alpha} & D.z^{\alpha} \\ D.z^{\alpha+\beta} & D^2.z^{\alpha+\beta} & D.z^{\alpha+\beta} \end{bmatrix} ; B_2 = \begin{bmatrix} z^{\beta} \\ D^2.z^{\alpha} \\ D^2.z^{\alpha+\beta} \end{bmatrix} ; C_2 = \begin{bmatrix} D & D^2 & D^3 \end{bmatrix}$$

Plugging these matrices in (50), we end up with

$$T = 2.D^2z^{\beta} + D^2z^{2\beta} + D^3z^{\alpha+\beta} + D^4z^{\alpha} + 2.D^4z^{\alpha+\beta} + D^4z^{2\beta} + D^4z^{3\beta} + 2.D^4z^{\alpha+2\beta} + D^4z^{\alpha+3\beta} + \ldots \tag{44}$$

Taking the derivative of (51) with respect to z and evaluating at $z = 1$, yields

$$\frac{\partial}{\partial z}T = \alpha.D^3 + 6\alpha D^4 + 4.\beta.D^2 + \beta.D^3 + 14.\beta.D^4 + \ldots\ldots \tag{45}$$

Note that the total average distortion is determined when (52) is evaluated with the values $\alpha = 1/2$ and $\beta = 1/2$. The total average distortion is

$$\widehat{d} = \frac{1}{2}\widehat{d_1} + \frac{1}{2}\widehat{d_2} \leq 2.D^2 + D^3 + 10.D^4 + \ldots\ldots$$

Now consider the time-invariant case by assuming system L_1 only. By substituting $A_2 = A_1$, $B_2 = B_1$, and $C_2 = C_1$ in (50), the total transfer function T is

$$T = 2.C_1.(I - A_1)^{-1}.B_1 = 2.T_1 \tag{46}$$

Substituting the (A_1, B_1, C_1) matrices in (53), we have

$$T_1 = D^3 z^{\beta} + D^4 z^{2\beta} + 2D^4 z^{\alpha+\beta} + D^4 z^{2\alpha+2\beta} + \ldots\ldots$$

Taking the derivative of T_1 with respect to z and evaluating at $z = 1$, gives us

$$\frac{\partial}{\partial z} T_1 = \beta . D^3 + 4.\alpha . D^4 + 6.\beta . D^4 + ... \tag{47}$$

Setting $\alpha = 1/2$ and $\beta = 1/2$ and evaluating (54), gives the total average distortion

$$\widehat{d} = \frac{1}{2}\widehat{d_1} + \frac{1}{2}\widehat{d_2} \leq \frac{1}{2}D^3 + 5.D^4 + ...$$

The modified transfer functions derived in this section allow existing codes to be evaluated for unequal error protection by providing a method to calculate the effective free distance vector.

A good LUEP encoder will have at least one effective free distance greater than the free distance of the optimal code of the same rate and state complexity.

A less than optimal free distance is an acceptable tradeoff when at least one effective free distance exceeds the optimal value.

20 Distance Properties of PTVCC

The performance of the PTVCCs, as the performance of the TICCs, is directly related to the decoding algorithm and to the distances properties. Thus, the most important distances used with PTVCCs are the following:

- **free distance**, denoted by d_{free} or by d_∞;
- **column distance**, denoted by d_i;
- **minimum distance**, denoted by $d_{\min}$.

Among these distances, the most important one is the free distance, for it is this distance that provides the error correction or error detection associated with the PTVCCs.

Next, we consider and define these distances and present some examples.

20.1 Free distance

The free distance d_{free} of a PTVCC is defined as the least Hamming distance between any two encoded sequences derived from two distinct information sequences, that is,

$$d_{free} \doteq \min\left\{d(\beta^1,\beta^2) : \alpha_1 \neq \alpha_2\right\},$$

where β^1 and β^2 are the encoded sequences corresponding to the information sequences α_1 and α_2, respectively. In this way, d_{free} is the least distance between any encoded sequences β^1 and β^2. Since convolutional codes are linear, the following properties hold,

$$\begin{aligned} d_{free} &\doteq \min\left\{w(\beta^1 \oplus \beta^2) : \alpha_1 \neq \alpha_2\right\}, \\ &= \min\left\{w(\beta) : \alpha \neq \mathbf{0}\right\}, \\ &= \min\left\{w(\alpha G) : \alpha \neq \mathbf{0}\right\}, \end{aligned}$$

where β is the encoded sequence corresponding to the information sequence α. Therefore, d_{free} corresponds to the minimum Hamming weight between all possible paths in the trellis that diverge from the state zero and come back to the state zero after some transitions in the trellis.

20.2 Column distance

With respect to the column distance, d_i, consider the encoded sequence being truncated after i transitions in the trellis. This sequence is denoted by $\beta^j_{[0,i]}$, $j = 1, 2$, or by,

$$\beta^j_{[0,i]} = (\beta^j_0, \beta^j_1, ..., \beta^j_i).$$

Let $\alpha_{[0,i]}$ be the corresponding truncated information sequence, that is,

$$\alpha_{[0,i]} = (\alpha_0, \alpha_1, ..., \alpha_i).$$

The column distance of order i, denoted by d_i, is defined as

$$
\begin{aligned}
d_i &\doteq \min\left\{ d(\beta^1_{[0,i]}, \beta^2_{[0,i]}) : \alpha^1_{[0]} \neq \alpha^2_{[0]} \right\}, \\
&= \min\left\{ w(\beta_{[0,i]}) : \alpha_{[0]} \neq \mathbf{0} \right\}.
\end{aligned}
$$

Thus, d_i is the minimum Hamming weight of the encoded sequence of length $(i+1)$ such that the first block of the information sequence is different from zero, that is, $\alpha_{[0]} \neq \mathbf{0}$.

In order to better characterize the determination of the column distance of order i by the truncated encoded sequence $\beta^j_{[0,i]}$ is that we write $\beta^j_{[0,i]}$ as a function of the generator matrix $G^j_{[0,i]}$ and of the truncated information sequence $\alpha_{[0,i]}$, that is,

$$
\beta^j_{[0,i]} = \alpha_{[0,i]} G^j_{[0,i]},
$$

where $G^j_{[0,i]}$ is a submatrix $b(i+1) \times n(i+1)$ of the matrix G^j, and it is given by

$$G^j_{[0,i]} = \begin{bmatrix} G^j_0 & G^j_1 & G^j_2 & \ldots\ldots & G^j_i \\ . & G^j_0 & G^j_1 & \ldots\ldots & G^j_{i-1} \\ . & . & G^j_0 & \ldots\ldots & G^j_{i-2} \\ . & . & . & . & . \\ . & . & . & . & G^j_0 \end{bmatrix}, \qquad \text{for} \quad i \le m$$

or

$$G^j_{[0,i]} = \begin{bmatrix} G^j_0 & G^j_1 & \ldots\ldots & G^j_m & . \\ . & G^j_0 & \ldots\ldots & G^j_{m-1} & G^j_m \\ . & . & . & . & . \\ . & . & G^j_0 & G^j_1 & G^j_2 \\ . & . & . & G^j_0 & G^j_1 \\ . & . & . & . & G^j_0 \end{bmatrix}, \qquad \text{for} \quad i > m$$

then,

$$d_i = \min\left\{ w(\alpha_{[0,i]} G^j_{[0,i]}) : \alpha_{[0]} \neq \mathbf{0} \quad \text{and} \quad j = 1,2 \right\}, \tag{48}$$

depends only on the $n(i+1)$ columns of G^j. This is the reason d_i is called **column distance**. As can be seen, as long as the value of i increases, the value of d_i does not decrease.

It is from the column distance that we may derive two cases of interest,

- When $i = m$, d_m is called **minimum distance**, $d_{\min}$, is the minimum Hamming weight of the encoded sequence with respect to the first constraint length.
- When $i \to \infty$, $\lim_{i\to\infty} d_i$ is the minimum Hamming weight of the encoded sequence of any length whose first block of the information sequence is different from zero.
- Finally, $d_{free} \doteq d_\infty = \lim_{i\to\infty} d_i$, for the noncatastrophic codes. Therefore, d_i eventually will achieve the value of d_{free} and from it the distance value will not increase any more.

21 Good PTVCC

The only known PTVCC satisfying Costello's conjecture, that is, the **lower bound on the free distance of nonsystematic TVCCs is greater than the upper bound on the free distance of TICCs** is shown next.

We define the goodness of a PTVCC with period T with respect to a TICC if the former:

1. has greater d_∞; or
2. has lower values of N_∞ (the average number of paths with distance d_∞ per time instant) and I_∞ (the average number of information bits in error per time instant along the paths with distance d_∞).

The parameters of the optimum noncatastrophic PTVCC with period $T = 2$, memory $m = 1$, rate $r = 2/3$ are shown in the next table.

	d_∞	N_∞	I_∞
TICC	3	1	1
PTVCC	4	$21/2$	$57/2$

Based on the optimality criterion, the generator matrices and the corresponding encoder of the optimum noncatastrophic PTVCC with period $T = 2$, memory $m = 1$, rate $r = 2/3$ are

$$G_0^1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad ; \quad G_1^1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

and

$$G_0^2 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad ; \quad G_1^2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

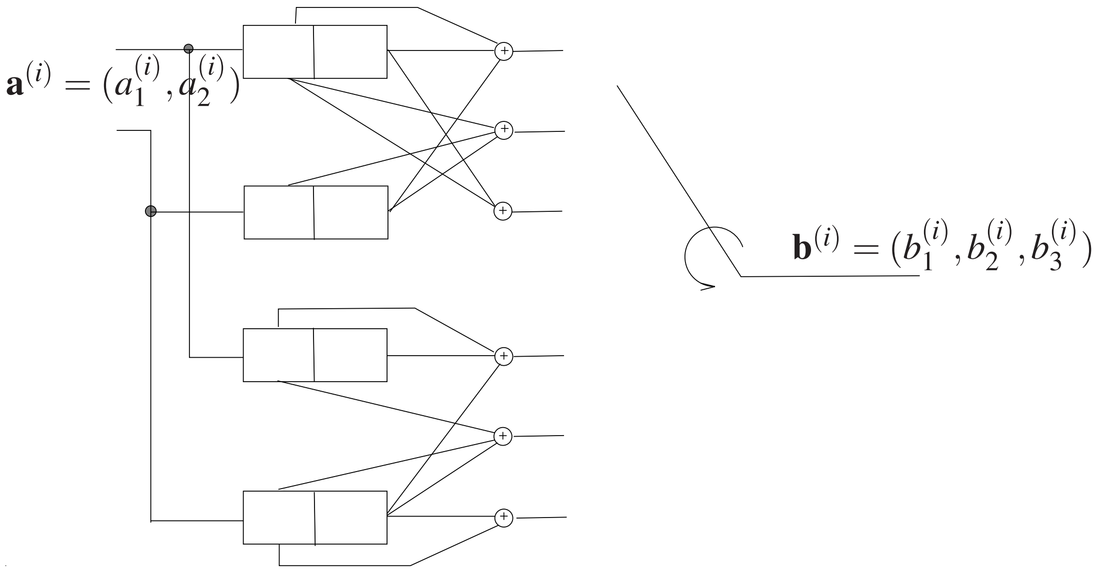


Figure 34: PTV convolutional encoder with period $T = 2$

Next, we present in Tables 3, 4, 5 and 6 the optimum PTVCC with period T with the corresponding values of d_∞, N_∞ and I_∞.

In general, we have that for any $n > 1$, n an integer, the PTVCC with period $T > 2$ with rate $r = 1/(3n+1)$ achieve free distances $d_\infty = 8n+2$, $N_\infty = I_\infty = (T-1)/T$, and generating functions consisting of $(T-1)5^{n+1}\,7^{2n}$ and one $5^n\,7^{2n+1}$ TI codes, where $5^2\,7^2 = 5577$. For $T = 1$, $d_\infty = 8n+2$, $N_\infty = I_\infty = 1$ and the generating functions are given by $5^{n+1}\,7^{2n}$.

Tabela I

T	r	d_{free}	N_∞	I_∞	**Octal representation**
1	1/4	10	1	1	$5^2 7^2$
2	1/4	10	1/2	1/2	$5^2 7^2$; 57^3
3	1/4	10	2/3	2/3	$5^2 7^2$; $5^2 7^3$; 57^3
4	1/4	10	3/4	3/4	$5^2 7^2$; $5^2 7^2$; $5^2 7^2$; 57^3
5	1/4	10	4/5	4/5	$5^2 7^2$; $5^2 7^2$; $5^2 7^2$; $5^2 7^2$; 57^3

Table 3: PTV convolutional encoders with memory $m = 2$

Tabela II					
T	r	d_{free}	N_∞	I_∞	**Octal representation**
1	1/7	18	1	1	$5^3 7^4$
2	1/7	18	1/2	1/2	$5^3 7^4$; $5^2 7^5$
3	1/7	18	2/3	2/3	$5^3 7^4$; $5^3 7^4$; $5^2 7^5$
4	1/7	18	3/4	3/4	$5^3 7^4$; $5^3 7^4$; $5^3 7^4$; $5^2 7^5$
5	1/7	18	4/5	4/5	$5^3 7^4$; $5^3 7^4$; $5^3 7^4$; $5^3 7^4$; $5^2 7^5$

Table 4: PTV convolutional encoders with memory $m = 2$

Tabela III					
T	r	d_{free}	N_∞	I_∞	**Octal representation**
1	1/10	26	1	1	$5^4 7^6$
2	1/10	26	1/2	1/2	$5^4 7^6$; $5^3 7^7$
3	1/10	26	2/3	2/3	$5^4 7^6$; $5^4 7^6$; $5^3 7^7$
4	1/10	26	3/4	3/4	$5^4 7^6$; $5^4 7^6$; $5^4 7^6$; $5^3 7^7$
5	1/10	26	4/5	4/5	$5^4 7^6$; $5^4 7^6$; $5^4 7^6$; $5^4 7^6$; $5^3 7^7$

Table 5: PTV convolutional encoders with memory $m = 2$

Tabela IV					
T	r	d_{free}	N_∞	I_∞	**Octal representation**
1	1/2	7	2	4	23 35
2	1/2	7	3/2	3	23 35; 25 37
3	1/2	7	4/3	8/3	23 35; 25 37; 27 35

Table 6: PTV convolutional encoders with memory $m = 4$

Chapter 5

Unequal Error Protection Convolutional Codes[a]

23. Objectives
24. More on Convolutional Codes
25. Effective Free Distance Vector
26. Classes of LUEP Convolutional Codes
27. State Diagram Analysis
28. Tables of LUEP Codes

[a]Most of the content of this chapter may be found in the papers by Palazzo, Mills/Costello, etc.

22 Objectives

In conventional channel coding applications, **it is assumed** the **code provides essentially equal error protection** ($\overline{P}_b$) to each bit.

In some applications, **some bit positions** in the information stream **are more important than others** like in PCM, in packet switched networks, in mobile communications, in embedded coding schemes for speech and high-definition television (HDTV), and so on.

Encoders which provide more than one level of error protection to the **input information sequence** are called **unequal error protection encoders (UEP)**.

These new classes of codes contain both TICE and PTVCE.

23 More on Convolutional Codes

A **code** consists of a set of allowable **output sequences** $\mathbf{v}$ and an **encoder** $\mathcal{E}(.)$, where $\mathcal{E}(.)$ **specifies the mapping** between the **input sequence** $\mathbf{u}$ and the output sequence $\mathbf{v} = \mathcal{E}(\mathbf{u})$.

Fig. 35 shows a rate $r = k/n$ feedforward convolutional encoder.

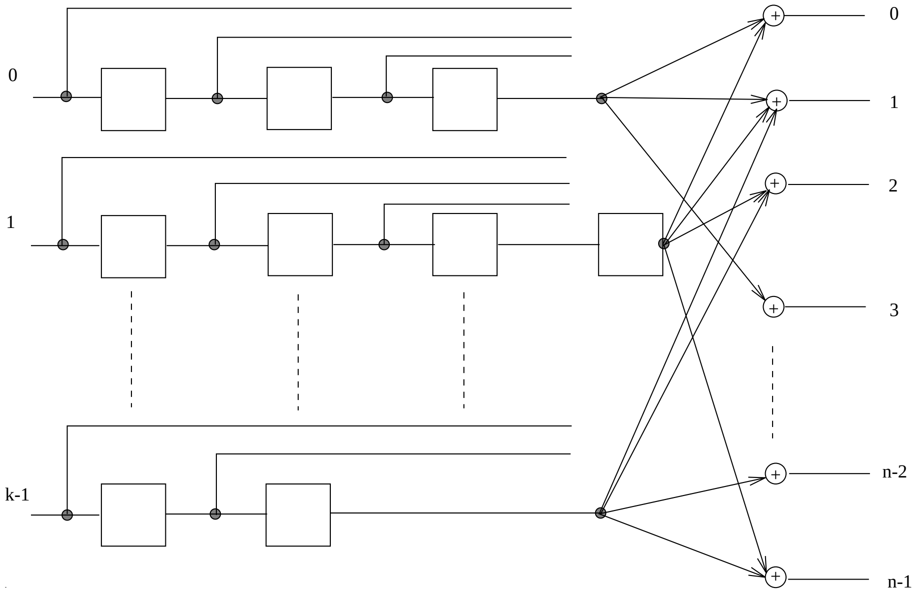


Figure 35: Convolutional encoder with $r = k/n$.

An (n,k,m) binary convolutional encoder can be represented by the encoding equation

$$\mathbf{v}_t = \mathbf{u}_t\mathbf{G}_0 \oplus \mathbf{u}_{t-1}\mathbf{G}_1 \oplus \cdots \oplus \mathbf{u}_{t-m}\mathbf{G}_m,$$

where the **information vector** at time t, $\mathbf{u}_t$, is a k-bit vector, the **code vector** at time t, $\mathbf{v}_t$, is an n-bit vector, and the **encoding matrices** $\mathbf{G}_i, i = 0, 1, \cdots, m$, are $(k \times n)$ binary matrices.

The more succinct notation

$$\mathbf{v}_{[t_0,t_1]} = \mathbf{u}_{[t_0,t_1]}\mathbf{G},$$

is used to represent the sequence of code vectors from time t_0 to time t_1 generated by the encoding matrices when the input is the vector $\mathbf{u}_{[t_0,t_1]}$.

The **memory allocation vector** $\mathbf{M} = (m_0, m_1, \cdots, m_{k-1})$ indicates the length of the shift-register connected to each input line. For example, the first input line has m_0 memory units.

The **largest value of** m_i is denoted by the **memory order**, $m = \max_i\{m_i\}$. When $\mathbf{M} = (m, m, \cdots, m)$, we say the encoder is a **balanced convolutional encoder**.

The **state complexity**, or **total memory**, of a convolutional encoder is defined to be **the number of memory units,** $K = \sum_{i=1}^{k} m_i$.

We define the overall encoding matrix, $\mathbf{G}$, as the concatenation of the matrices $\mathbf{G}_i$, $0 \le i \le m$, that is,

$$\mathbf{G} = [\mathbf{G}_0 | \mathbf{G}_1 | \cdots | \mathbf{G}_m].$$

The free distance, d_{free}, of a convolutional code, $\mathcal{C}$, is

$$d_{free} = \min_{\mathbf{v}_{[0,t]} \neq \mathbf{v}'_{[0,t]}} \left\{ d_H(\mathbf{v}_{[0,t]}, \mathbf{v}'_{[0,t]}) : \mathbf{v}_{[0,t]}, \mathbf{v}'_{[0,t]} \in \mathcal{C}, \text{ and } t \ge 1 \right\}.$$

Example 11. *Consider the rate $r = 2/3$ convolutional encoder with $\mathbf{M} = (1,2), K = 3$, and encoding matrices*

$$\mathbf{G}_0 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}; \mathbf{G}_1 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}; \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

This encoder is shown in Fig. 36. The free distance is 4.

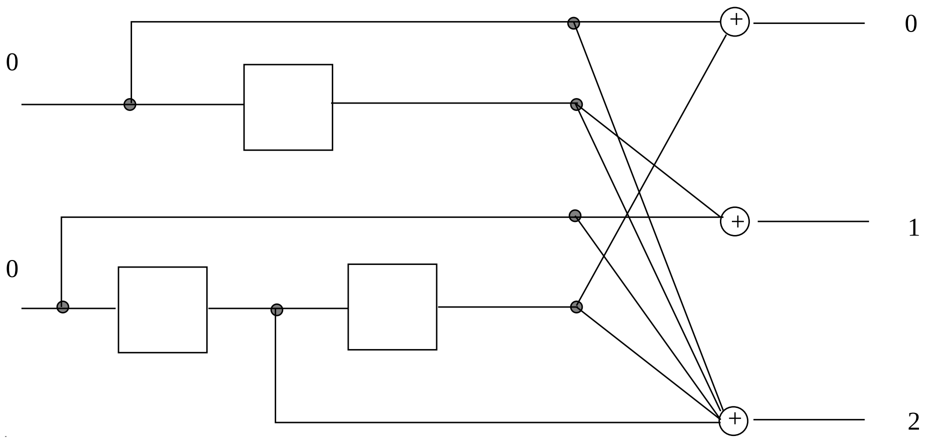


Figure 36: Convolutional encoder with $r = 2/3, M = (1,2)$ and $K = 3$.

An (n,k,m) periodically time-varying convolutional encoder with period P can be represented by the encoding equation

$$\mathbf{v}_t = \mathbf{u}_t \mathbf{G}_0^{(t)_P} \oplus \mathbf{u}_{t-1} \mathbf{G}_1^{(t)_P} \oplus \ldots\ldots \oplus \mathbf{u}_{t-m} \mathbf{G}_m^{(t)_P},$$

where $(t)_P = 0, 1, \ldots, (P-1)$ denotes the value of t modulo P, and the encoding matrices $\mathbf{G}_i^{(t)_P}$, $i = 0, 1, \ldots, m$ are $(k \times n)$ binary matrices which describe the input-output relationship at time t.

The encoder matrices describing the connections at time t are collectively called $\mathcal{E}_{(t)_P}$. The rate of the encoder is $r = k/n$. It is assumed that the first non-zero input occurs at time 0.

A periodically time-varying convolutional encoder with period $P = 1$ is identical to a time-invariant convolutional encoder.

Example 12. *Consider the $r = 2/3$ and $\mathbf{M} = (1, 1)$ periodically time-varying convolutional encoder with period 2 and encoding matrices*

$$\mathbf{G}_0^{(0)} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}; \mathbf{G}_1^{(0)} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{G}_0^{(1)} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}; \mathbf{G}_1^{(1)} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

The encoder produces the code sequence

$$\mathbf{v} = (010\;010\;100\;101\;000\;111\;110\;\cdots)$$

given the input sequence

$$\mathbf{u} = (10\;11\;01\;01\;11\;10\;00\;\cdots).$$

A unit-memory code (UMC) is a binary convolutional code with memory $m = 1$ and multiple input lines, $i.e.$, $k > 1$.

The encoding equation is $\mathbf{v}_t = \mathbf{u}_t \mathbf{G}_0 \oplus \mathbf{u}_{t-1}\mathbf{G}_1$.

An (n_0, k_0, m) convolutional encoder with $(k_0 \times n_0)$ encoding matrices $\mathbf{g}_0, \mathbf{g}_1, \cdots, \mathbf{g}_m$, is equivalent to the $(n = mn_0, k = mk_0)$ UMC which has the two $(mk_0 \times mn_0)$ binary encoding matrices

$$\mathbf{G}_0 = \begin{bmatrix} \mathbf{g}_0 & \mathbf{g}_1 & \cdots & \mathbf{g}_{m-1} \\ \mathbf{0} & \mathbf{g}_0 & \cdots & \mathbf{g}_{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{g}_0 \end{bmatrix} \quad \text{and} \quad \mathbf{G}_1 = \begin{bmatrix} \mathbf{g}_m & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{g}_{m-1} & \mathbf{g}_m & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_m \end{bmatrix}. \tag{49}$$

The **two encoders are equivalent** in the sense that **they generate identical outputs** when **operating on identical input sequences**.

24 Effective Free Distance Vector

It is convenient to define an effective free distance vector, $\mathbf{d_{eff}}$, as an alternative to the free distance as a primary performance parameter for the linear unequal error protection codes.

The **performance criterion** based on the **effective free distance vector** is equivalent to the **separation vector concept of** (n,k) **linear codes** introduced by Dunning and Robbins.

Definition 4. *For an (n,k,m), $k \neq 1$, time-invariant convolutional code, and an (n,k,m) periodically time-varying convolutional code over $GF(2)$, the effective free distance vector is defined as the k-dimensional vector*

$$d_{eff}(C) = (d_{eff}^{(1)}, d_{eff}^{(2)}, \ldots\ldots, d_{eff}^{(k)}),$$

where $d_{eff}^{(j)}$, the j-th effective free distance, is given by

$$d_{eff}^{(j)} = \min\ degree \left\{ \frac{\partial}{\partial z} T(c_1, c_2, \ldots\ldots, c_k, z, D) \right\}_{z=1, c_j=1, c_i=0, any\ i \neq j, D},$$

with $T(.)$ being either the modified transfer function for time-invariant or time-varying convolutional codes, z is a decoding error, D is the Bhatacharyya function and the c_i's are distortion functions such that $c_j = 0$ if the j-th transmitted and decoded information digits are equal, otherwise $c_j = 1$, with $1 \leq j \leq k$.

From this definition follows the error-correcting capability of a convolutional code when used in a binary-input symmetric-output channel.

Theorem 3. *An* (n,k,m), $k \neq 1$, *time-invariant convolutional code, and an* (n,k,m) *periodically time-varying convolutional code over* $GF(2)$ *with generator matrix* $\mathbf{G}$ *and Viterbi or Sequential decoding algorithm, guarantees that the* j*-th input information digit is decoded correctly whenever the error pattern has Hamming distance less than or equal to* $\left\lfloor (d_{eff}^{(j)} - 1)/2 \right\rfloor$, *where* $\lfloor \cdot \rfloor$ *is the floor-function.*

It is evident from the previous definition that the free distance of the code equals

$$d_{free} = \min\left\{d_{eff}^{(1)}, d_{eff}^{(2)}, \ldots\ldots, d_{eff}^{(k)}\right\}$$

Thus, $d_{eff}^{(i)} \geq d_{free}$.

If an effective free distance vector $d_{eff}(\mathcal{C})$ corresponding to a convolutional code with generator matrix $\mathbf{G}$ is such that its components are not mutually equal, then we call this code a **linear unequal error protection convolutional code**.

Note that for a given memory allocation vector $\mathbf{M} = (m_1, m_2, ..., m_k)$, and a given set of associated submatrices $\mathbf{G}_i$, each permutation of the k components of $\mathbf{M}$ and the corresponding rows of each $\mathbf{G}_i$ lead to effective free distance vectors $d_{eff}(\mathcal{C})$ whose components take values on the set $\left\{d_{eff}^{(1)}, d_{eff}^{(2)}, ..., d_{eff}^{(k)}\right\}$.

Without loss of generality, let $\widetilde{d}_{eff}(\mathcal{C}) = \left\{\widetilde{d}_{eff}^{(1)}, \widetilde{d}_{eff}^{(2)}, \cdots, \widetilde{d}_{eff}^{(k)}\right\}$ denote the ordered set of the effective free distance vector $d_{eff}(\mathcal{C})$ of an (n, k, m) convolutional code.

Hence, the optimality and equivalence conditions of LUEP convolutional codes are established as follows:

Definition 5. *An (n,k,m) UEP convolutional code with state complexity K and ordered effective free distance vector $\widetilde{d}_{eff}(\mathcal{C})$ is* ***optimum*** *if* ***there exists no other*** (n,k,m) ***LUEP convolutional code*** *with state complexity K and* ***ordered effective free distance vector*** $\widetilde{d}_{eff}(\mathcal{C}_1)$ *which is* ***lexicographically better than*** $\widetilde{d}_{eff}(\mathcal{C})$.

Definition 6. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two (n,k,m) UEP convolutional codes each with state complexity K and ordered effective free distance vectors $\widetilde{d}_{eff}(\mathcal{C}_1)$ and $\widetilde{d}_{eff}(\mathcal{C}_2)$, respectively. They are said to be* ***partially equivalents****, or* ***equivalents of order*** *j, with $1 \leq j < k$, if the j first components of the corresponding ordered effective free distance vectors are equal.*

Definition 7. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two (n,k,m) UEP convolutional codes each with state complexity K and ordered effective free distance vectors $\widetilde{d}_{eff}(\mathcal{C}_1)$ and $\widetilde{d}_{eff}(\mathcal{C}_2)$, respectively. They are said to be* ***completely equivalents****, or* ***equivalents of order*** *$j = k$, or simply* ***equivalents*** *if the ordered sets of the effective free distance vectors are equal, that is, $\widetilde{d}_{eff}(\mathcal{C}_1) = \widetilde{d}_{eff}(\mathcal{C}_2)$.*

Example 13. *Consider the* $(3,2)$ *convolutional encoder with* $\mathbf{M} = (1,1)$, *and submatrices*

$$\mathbf{G}_0 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad ; \qquad \mathbf{G}_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

The encoder diagram and the associated trellis are shown in Figs. 37, and 38, respectively.

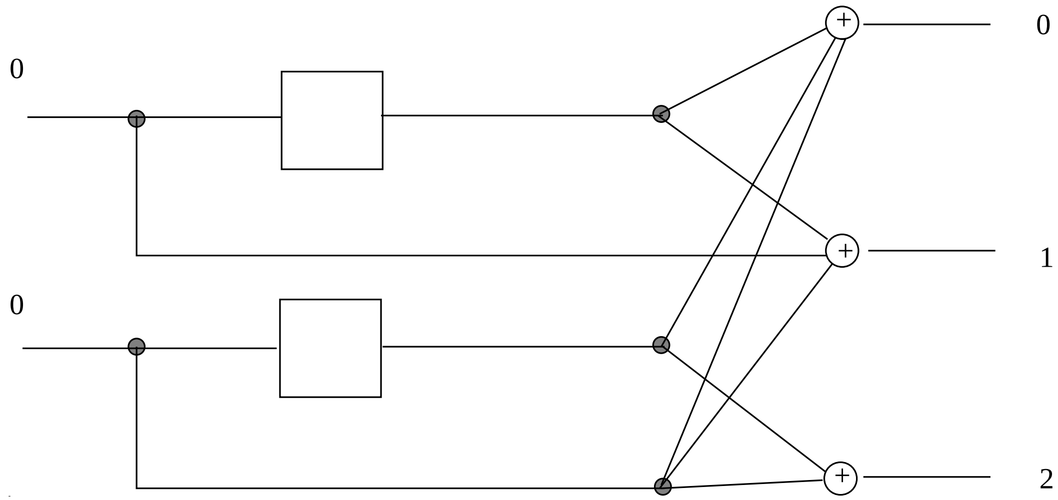


Figure 37: Convolutional encoder with $r = 2/3, M = (1,1)$.

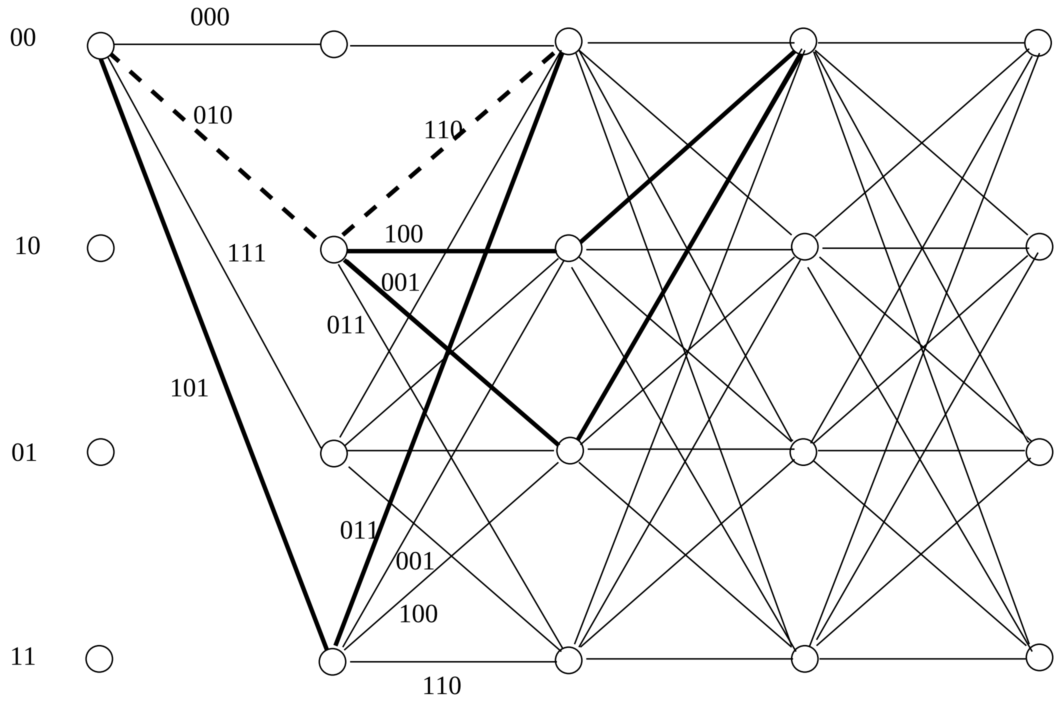


Figure 38: Associated trellis for the (3,2) and M=(1,1) convolutional encoder.

Examining the possible paths through the trellis reveals that the first effective free distance, $d_{eff}^{(1)}$, is 3, and the second effective free distance, $d_{eff}^{(2)}$, is 4, or equivalently, $d_{eff} = (3,4)$. The non-zero path through the trellis with weight 3 is shown by dotted lines. The non-zero paths with weight 4 are shown with dark lines. All other paths have weight greater than 4. It can be seen that the weight 3 path is created by an input vector *sequence that is non-zero only in the first input bit position,* $(01,00)$. *When the input sequences are non-zero in the second position, the minimum weight of any path is* 4.

It should be noted that the individual bit error probability and the effective free distance vector depend on the encoder realization of a code. An example that demonstrates this dependence follows.

Example 14. *The bounds on the individual bit error probabilities for the encoder in Fig. 37 are*

$$\widehat{d_1} \leq D^3 + 7D^4 + \cdots,$$

and

$$\widehat{d_2} \leq D^3 + 10D^4 + \cdots.$$

If the feedback form of the previous code is used, then the bounds are $\widehat{d_1} \leq 3D^3 + 9D^4 + \cdots$, *and* $\widehat{d_2} \leq 2D^3 + 6D^4 + \cdots$. *The effective free distance vectors of the two encoders are both* $d_{eff} = (3,3)$, *but the bounds on the individual bit error probabilities are slightly different for the two encoders.*

25 Classes of LUEP Convolutional Codes

We start this section by restricting the class of convolutional codes that can be used as LUEP codes.

Proposition 2. *An $(n,1,m)$* ***time-invariant*** *binary convolutional code* **can not** *provide* ***unequal error protection****.*

As a consequence of Proposition 1, we have

Proposition 3. *An $(n' = mn, k' = m, 1)$* ***time-invariant binary unit-memory*** *or an $(n' = n(m+i), k' = m+i, 1)$****, i an integer and $i \geq 1$, partial unit-memory convolutional code generated from an*** *$(n,1,m)$* ***time-invariant*** *binary convolutional code* **can not** *provide* ***unequal error protection****.*

On the other hand, **specific time-invariant convolutional codes**, that is, k**-parallel shift-registers with** m**-memory cells each as well as systematic convolutional codes with rate** $r = k/n$ **with** $k \geq 2$, **offer** a natural **unequal error protection** for each one of the k information digits.

This fact is due to the **nonsymmetric** characteristic of the tap connections between the cells of the shift-registers and the mod 2 adders.

Omura was the **first to notice the unequal error protection** provided by a $(3,2,2)$ convolutional code.

In the class of PTVCE codes, one can find good (n,k,m) LUEP convolutional codes by imposing the following conditions:

1. the **combination has the same rate and constraint-length** (state complexity);
2. the **combination has different rates** such that **the parameter maximum flow Φ is the same**, where $\Phi = n.2^{(k-1)}$; state diagram;
3. the **combination has different maximum flows however with the branch codewords having the same blocklength**.

In general, the **codes obtained from condition** 3 have a **smaller number of paths with distance** $\widetilde{d}_{eff}^{(j)} + i$, with i an integer and $i \geq 0$. Note that under Definition 6 these codes are equivalents.

26 State Diagram Analysis

State diagram analysis has long been used to determine the transfer functions of low complexity (n,k,m) convolutional encoders. **Under the assumption of sequence independence** the **superstate representation reduces to the usual state representation**.

Consider two independent discrete-time linear systems L_1 and L_2 whose state and output equations are, respectively, given by:

$$\text{System } L_1 \quad \Longleftrightarrow \quad \begin{cases} \varepsilon_1(k+1) = A_1(k).\varepsilon_1(k) + B_1(k).u(k) \\ \\ T_1(k) = C_1(k).\varepsilon_1(k) \end{cases}$$

$$\text{System } L_2 \quad \Longleftrightarrow \quad \begin{cases} \varepsilon_2(k+1) = A_2(k).\varepsilon_2(k) + B_2(k).u(k) \\ \\ T_2(k) = C_2(k).\varepsilon_2(k) \end{cases}$$

The evolution of these systems is as follows:

- System L_1 (convolutional encoder) is **on** in the time interval $i \leq k < i+1$
- System L_2 (convolutional encoder) is **on** in the time interval $i+1 \leq k < i+2$, for any i, i an integer.

It can be shown that the combination of these two convolutional encoders results in a convolutional encoder.

The state and output equations of this linear system can be described by the following reasoning:

- The **states** ε_1 (ε_2) belonging to system L_1 (L_2), at time $t = k+1$ can be reached from the **states** ε_2 (ε_1) belonging to system L_2 (L_1) at time $t = k$, by applying the **transition matrix** A_1 (A_2), corresponding to system L_1 (L_2), or from the **initial condition matrix** B_1 (B_2);
- The **output equation** T_1 (T_2) of system L_1 (L_2) is obtained by multiplying the **output condition matrix** C_1 (C_2) by the **state matrix** ε_2 (ε_1) of system L_2 (L_1).

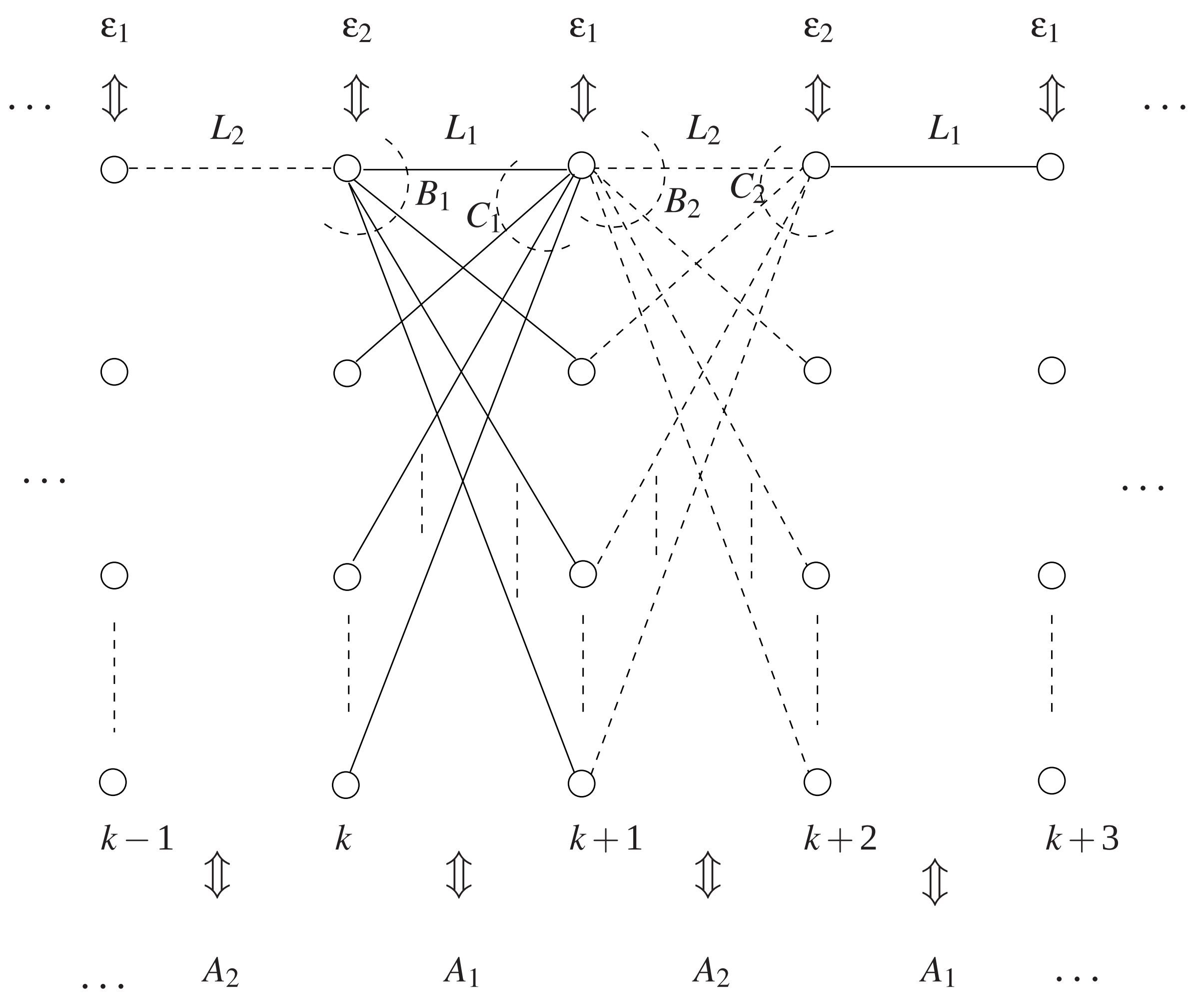


Figure 39: Trellis showing the state and output equations of a time-varying linear system

Mathematically,

$$\text{System } L_1 \Longleftrightarrow \begin{cases} \varepsilon_1(k+1) = A_1(k).\varepsilon_2(k) + B_1(k) \\ \\ T_1(k) = C_1(k).\varepsilon_2(k) \end{cases}$$

$$\text{System } L_2 \Longleftrightarrow \begin{cases} \varepsilon_2(k+1) = A_2(k).\varepsilon_1(k) + B_2(k) \\ \\ T_2(k) = C_2(k).\varepsilon_1(k) \end{cases}$$

The total output equation is

$$T(k) = T_1(k) + T_2(k)$$

Solving the equations for $\varepsilon_1(k)$ and $\varepsilon_2(k)$, we have

$$\varepsilon_1 = (I - A_1.A_2)^{-1}(A_1.B_2 + B_1)$$

$$\varepsilon_2 = A_2.(I - A_1.A_2)^{-1}(A_1.B_2 + B_1) + B_2$$

Substituting these solutions in $T_2(k)$ and $T_1(k)$, the total output $T(k)$ is completely determined, and is given by

$$T = C_1.A_2.(I - A_1.A_2)^{-1}(A_1.B_2 + B_1) + C_1.B_2 + C_2.(I - A_1.A_2)^{-1}(A_1.B_2 + B_1) \quad (50)$$

Example 15. *Let the systems L_1 and L_2 be characterized by the convolutional encoders $(3,2,1)$ with memory allocation vectors $\mathbf{M} = (1,1)$ and with corresponding submatrices*

$$\mathbf{G}_0^{(1)} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} ; \mathbf{G}_1^{(1)} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$\mathbf{G}_0^{(2)} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} ; \mathbf{G}_1^{(2)} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Consider the following distortion table

$$
\begin{array}{cccccc}
d(u_k,\widetilde{u}_k) & 00 & 01 & 10 & 11 & \leftarrow \quad \widetilde{u}_k \\
00 & 0 & \beta & \alpha & \alpha+\beta & \\
01 & \beta & 0 & \alpha+\beta & \alpha & \\
10 & \alpha & \alpha+\beta & 0 & \beta & \\
11 & \alpha+\beta & \alpha & \beta & 0 & \\
\uparrow & & & & & \\
u_k & & & & &
\end{array}
$$

The split state diagram representations of the convolutional encoders are shown in Fig. 40.

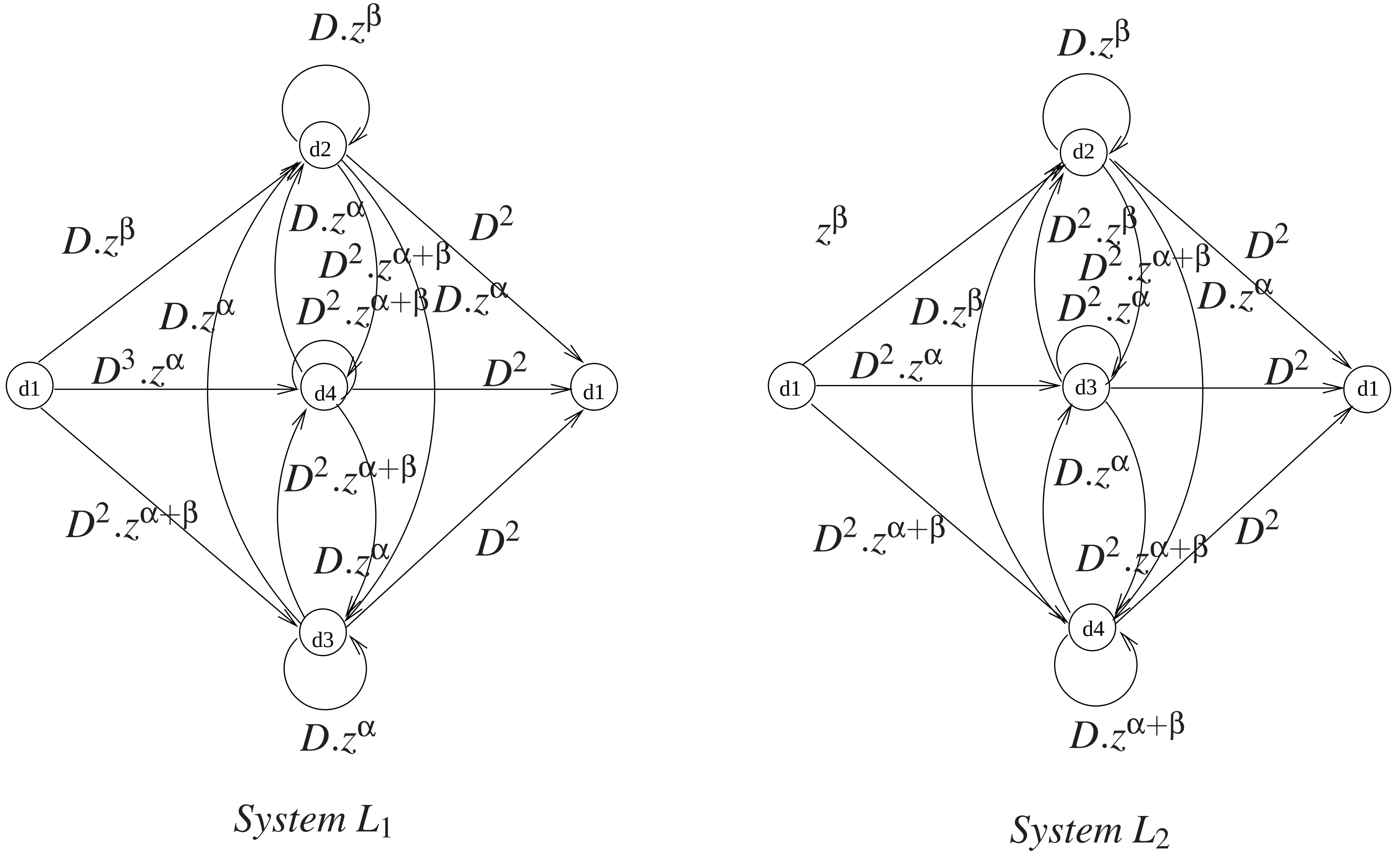


Figure 40: Split state diagram of the convolutional encoders.

From the split state diagrams, the corresponding (A,B,C) *matrices are*

$$A_1 = \begin{bmatrix} D.z^{\beta} & D^3.z^{\beta} & D.z^{\beta} \\ D.z^{\alpha} & D.z^{\alpha} & D.z^{\alpha} \\ D^2.z^{\alpha+\beta} & D^2.z^{\alpha+\beta} & D^2.z^{\alpha+\beta} \end{bmatrix} ; B_1 = \begin{bmatrix} D.z^{\beta} \\ D^3.z^{\alpha} \\ D^2.z^{\alpha+\beta} \end{bmatrix} ; C_1 = \begin{bmatrix} D^2 & D^2 & D^2 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} D.z^{\beta} & D^2.z^{\beta} & D^3.z^{\beta} \\ D.z^{\alpha} & D^2.z^{\alpha} & D.z^{\alpha} \\ D.z^{\alpha+\beta} & D^2.z^{\alpha+\beta} & D.z^{\alpha+\beta} \end{bmatrix} ; B_2 = \begin{bmatrix} z^{\beta} \\ D^2.z^{\alpha} \\ D^2.z^{\alpha+\beta} \end{bmatrix} ; C_2 = \begin{bmatrix} D & D^2 & D^3 \end{bmatrix}$$

Plugging these matrices in (50), we end up with

$$T = 2.D^2z^{\beta} + D^2z^{2\beta} + D^3z^{\alpha+\beta} + D^4z^{\alpha} + 2.D^4z^{\alpha+\beta} + D^4z^{2\beta} + D^4z^{3\beta} + 2.D^4z^{\alpha+2\beta} + D^4z^{\alpha+3\beta} + \cdots \tag{51}$$

Taking the derivative of (51) with respect to z and evaluating at $z = 1$, yields

$$\frac{\partial}{\partial z}T = \alpha.D^3 + 6\alpha D^4 + 4.\beta.D^2 + \beta.D^3 + 14.\beta.D^4 + \cdots \tag{52}$$

The average distortion with respect to the first position of the input information digit is found when $\alpha = 1$ *and* $\beta = 0$ *in (52). This leads to*

$$\widehat{d_1} \leq \frac{\partial}{\partial z} T \Big|_{z=1,\ \alpha=1,\ \beta=0} = D^3 + 6D^4 + \cdots$$

Similarly, the average distortion with respect to the second position of the input information digit is found when $\alpha = 0$ *and* $\beta = 1$ *in (52). Therefore,*

$$\widehat{d_2} \leq \frac{\partial}{\partial z} T \Big|_{z=1,\ \alpha=0,\ \beta=1} = 4.D^2 + D^3 + 14.D^4 + \cdots$$

Therefore, $d_{eff} = (3,2)$*, or* $\widetilde{d}_{eff} = (2,3)$*. Note that the total average distortion is determined when (52) is evaluated with the values* $\alpha = 1/2$ *and* $\beta = 1/2$*. The total average distortion is*

$$\widehat{d} = \frac{1}{2}\widehat{d_1} + \frac{1}{2}\widehat{d_2} \leq 2.D^2 + D^3 + 10.D^4 + \cdots$$

Now consider the time-invariant case by assuming system L_1 only. By substituting $A_2 = A_1$, $B_2 = B_1$, and $C_2 = C_1$ in (50), the total transfer function T is

$$T = 2.C_1.(I - A_1)^{-1}.B_1 = 2.T_1 \tag{53}$$

Substituting the (A_1, B_1, C_1) matrices in (53), we have

$$T_1 = D^3 z^{\beta} + D^4 z^{2\beta} + 2D^4 z^{\alpha+\beta} + D^4 z^{2\alpha+2\beta} + \ldots\ldots$$

Taking the derivative of T_1 with respect to z and evaluating at $z = 1$, gives us

$$\frac{\partial}{\partial z} T_1 = \beta.D^3 + 4.\alpha.D^4 + 6.\beta.D^4 + \cdots \tag{54}$$

The average distortion with respect to the first position of the input information digit is determined by setting $\alpha = 1$ *and* $\beta = 0$ *in (54), and so*

$$\widehat{d_1} \leq \frac{\partial}{\partial z} T \Big|_{z=1,\ \alpha=1,\ \beta=0} = 4.D^4 + \cdots$$

The average distortion with respect to the second position of the input information digit is found when $\alpha = 0$ *and* $\beta = 1$ *in (54), and so*

$$\widehat{d_2} \leq \frac{\partial}{\partial z} T \Big|_{z=1,\ \alpha=0,\ \beta=1} = D^3 + 6.D^4 + \cdots$$

Therefore, $d_{eff} = (4,3)$*, or* $\widetilde{d}_{eff} = (3,4)$*. We remind the reader that this is the same result as that of Example 2. Setting* $\alpha = 1/2$ *and* $\beta = 1/2$ *and evaluating (54), gives the total average distortion*

$$\widehat{d} = \frac{1}{2}\widehat{d_1} + \frac{1}{2}\widehat{d_2} \leq \frac{1}{2}D^3 + 5.D^4 + \cdots$$

27 Tables

Memory Distribution M=(1,1)				
Rate	**Encoding Matrices**		**Eff. Free Dist.**	**Free Dist.**
k/n	G_0	G_1	$\tilde{d}_{eff}$	d_{free}
$2/3$	$3,6$	$2,5$	$(3,4)$	3
$2/5$	$31,26$	$25,33$	$(6,7)$	6
$2/8$	$370,037$	$174,237$	$(10,11)$	10
$2/11$	$3163,1755$	$1077,3760$	$(14,15)$	14
$2/14$	$37700,03477$	$37760,01777$	$(18,19)$	18
$2/17$	$003777,377700$	$201777,177760$	$(22,23)$	22
$2/20$	$3777600,0017777$	$0777740,3007777$	$(26,27)$	26

Table 7: Time-Invariant LUEP Encoders - In general, for rates $r = 2/(3n+5)$; $\tilde{d}_{eff}^{(1)} = (4n+6)$ and $\tilde{d}_{eff}^{(2)} = (4n+7)$ where n is an integer and $n \geq 0$. Notation: the numbers under the columns G_0 and G_1 are in octal representation; the numbers prior to the comma represent the first row, and the numbers after the comma represent the second row of each submatrix. For instance 370,037 means that the first row of G_0 is 11 111 000 and the second row is 00 011 111.

Memory Distribution M=(1,1)				
Rate	**Encoding Matrices**		**Eff. Free Dist.**	**Free Dist.**
k/n	G_0	G_1	$\tilde{d}_{eff}$	d_{free}
$3/4$	$03, 05, 16$	$11, 12, 14$	$(4, 4, 5)$	4
$3/6$	$70, 36, 13$	$34, 07, 62$	$(6, 6, 7)$	6
$3/8$	$360, 074, 227$	$146, 073, 303$	$(8, 9, 9)$	8
$3/11$	$3434, 0770, 2133$	$2525, 2752, 1361$	$(12, 12, 13)$	12
$4/5$	$16, 22, 02, 13$	$03, 06, 14, 30$	$(3, 3, 4, 5)$	3
$4/7$	$170, 036, 115, 055$	$140, 070, 056, 041$	$(6, 6, 6, 7)$	6
$4/9$	$740, 146, 264, 453$	$170, 147, 716, 444$	$(8, 8, 9, 9)$	8

Table 8: Time-Invariant LUEP Encoders. The notation used in this table follows the same notation of Table 7.

Memory Elements m = 2				
Period 2				
Rate	**Encoding Matrices**		**Eff. Free Dist.**	**Free Dist.**
k/n	$Code1$	$Code2$	$\tilde{d}_{eff}$	d_{free}
$1/4$	$5^2 7^2$	$5 7^3$	$(10,11)$	10
$1/7$	$5^3 7^4$	$5^2 7^5$	$(18,19)$	18
$1/10$	$5^4 7^6$	$5^3 7^7$	$(26,27)$	26
$1/13$	$5^5 7^8$	$5^4 7^9$	$(34,35)$	34
$1/16$	$5^6 7^{10}$	$5^5 7^{11}$	$(42,43)$	42
$1/19$	$5^7 7^{12}$	$5^6 7^{13}$	$(50,51)$	50
$1/22$	$5^8 7^{14}$	$5^7 7^{15}$	$(58,59)$	58

Table 9: Time-Varying LUEP Encoders - In general, for rates $r = 1/(3n+4)$; $\tilde{d}_{eff}^{(1)} = (8n+10)$ and $\tilde{d}_{eff}^{(2)} = (8n+11)$, where n is an integer and $n \geq 0$. We employ the following representation $5^2 7^5 = 5577777$.

Memory Elements m = 2					
Period 3					
Rate	**Encoding Matrices**			**Eff. Free Dist.**	**Free Dist.**
k/n	$Code1$	$Code2$	$Code3$	$\tilde{d}_{eff}$	d_{free}
$1/4$	57^3	$5^2 7^2$	$5^2 7^2$	$(10,10,11)$	10
$1/7$	$5^3 7^4$	$5^2 7^5$	$5^3 7^4$	$(18,18,19)$	18
$1/10$	$5^4 7^6$	$5^4 7^6$	$5^3 7^7$	$(26,26,27)$	26
$1/13$	$5^5 7^8$	$5^5 7^8$	$5^4 7^9$	$(34,34,35)$	34
$1/16$	$5^6 7^{10}$	$5^6 7^{10}$	$5^5 7^{11}$	$(42,42,43)$	42
$1/19$	$5^7 7^{12}$	$5^7 7^{12}$	$5^6 7^{13}$	$(50,50,51)$	50
$1/22$	$5^8 7^{14}$	$5^8 7^{14}$	$5^7 7^{15}$	$(58,58,59)$	58

Table 10: Time-Varying LUEP Encoders - In general, for rates $r = 1/(3n+4)$; $\widetilde{d}^{(1)}_{eff} = \widetilde{d}^{(2)}_{eff} = (8n+10)$ and $\widetilde{d}^{(3)}_{eff} = (8n+11)$, where n is an integer and $n \geq 0$. We employ the following representation $5^3 7^4 = 5557777$.