Convolutional Codes

Communication and Coding Laboratory

Dept. of Electrical Engineering,
National Chung Hsing University

- **Chapter 7: Convolutional Codes**
  1. Preview of convolutional codes
  2. Shift register representation
  3. Scalar generator matrix in the time domain
  4. Impulse response of MIMO LTI system
  5. Polynomial generator matrix in the frequency domain
  6. State diagram, tree, and trellis
  7. Construction of minimal trellis
  8. The algebraic theory of convolutional codes
  9. Free distance and path enumerator
  10. Termination, truncation, tailbiting, and puncturing
  11. Optimal decoding: Viterbi and BCJR decoding
  12. Suboptimal decoding: sequential and threshold decoding

Reference

1. **Lin, Error control coding: chapter 11, 12, and 13**
2. **Johannesson, Fundamentals of convolutional coding**
3. **Lee, Convolutional coding**
4. **Dholakis, Introduction to convolutional codes**
5. Adamek, Foundation of coding: chapter 14
6. Wicker, Error control systems for digital communication: chapter 11 and 12
7. Reed, Error Control doding for data networks: chapter 8
8. Blahut, Algebraic codes for data transmission: chapter 9

Preview of Convolutional Codes

- Block codes and convolutional codes are two major class of codes for error correction.
- From a viewpoint, convolutional codes differ form block codes in that the encoder contains memory.
- For convolutional codes, the encoder outputs at any given time unit depend not only on the inputs at that time unit but also on some number of previous inputs:

$$v_t = f(u_{t-m}, \cdots, u_{t-1}, u_t),$$

where $v_t \in F_2^n$ and $u_t \in F_2^k$.
- A rate $R = \frac{k}{n}$ convolutional encoder with memory order m can be realized as a k-input, n-output linear sequential circuit with input memory m.

- Convolutional codes were first introduced by Elias in 1955.
- The information and codewords of convolutional codes are of infinite length, and therefore they are mostly referred to as information and code sequence.
- In practice, we have to truncate the convolutional codes by zero-biting, tailbiting, or puncturing.
- There are several methods to describe a convolutional codes.
  1. Sequential circuit: shift register representation.
  2. MIMO LTI system: impulse response encoder
  3. Algebraic description: scalar G matrix in time domain
  4. Algebraic description: polynomial G matrix in Z domain,
  5. Combinatorial description: state diagram and trellis

- We emphasize differences among the terms: code, generator matrix, and encoder.
  1. Code: the set of all code sequences that can be created with a linear mapping.
  2. Generator matrix: a rule for mapping information to code sequences.
  3. Encoder: the realization of a generator matrix as a digital LTI system.
- For example, one convolutional code can be generated by several different generator matrices and each generator matrix can be realized by different encoder, e.g., controllable and observable encoders.

Summary

- $\cdots u(i-1)u(i)\cdots \longrightarrow$ Encoding $\longrightarrow \cdots c(i-1)c(i)\cdots$
  $u(i) = (u_1(i), \ldots, u_k(i))$, $c(i) = (c_1(i), \ldots, c_n(i))$
- $u(D) = \sum_i u(i)D^i \longrightarrow$ G(D) $\longrightarrow c(D) = \sum_i c(i)D^i$
  $c(D) = u(D)G(D)$
- There are two types of codes in general
  - Block codes: $G(D) = G \Longrightarrow c(i) = u(i)G$
  - Convolutional codes: $G(D) = G_0 + G_1 D + \cdots + G_m D^m$

$$\Longrightarrow c(i) = u(i)G_0 + u(i-1)G_1 + \cdots u(i-m)G_m$$

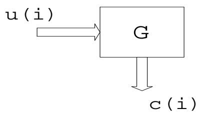


Figure 1: Encoder of block codes

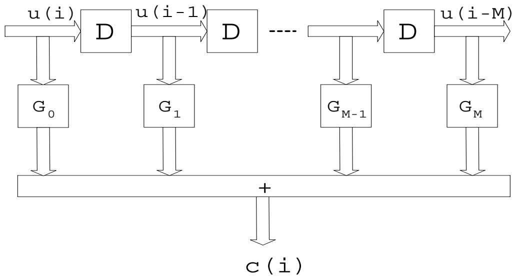


Figure 2: Encoder of convolutional codes

Shift register representation

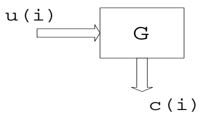


Figure 3: Encoder of block codes

- Use combinatorial logic to implement block codes.
- A information block $u(i)$ of length k at time i is mapped to a codeword $c(i)$ of length n at time i by a $k \times n$ generator matrix for each i, i.e., no memory.

$$c(i) = u(i) \cdot G$$

- We denote this linear block code by $C[n, k]$, usually, n and k are large.

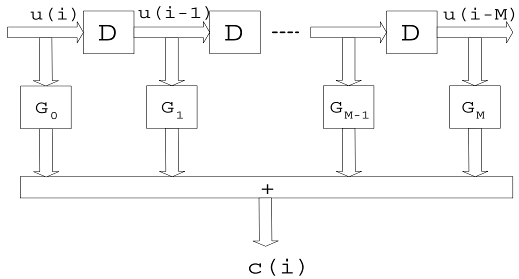


Figure 4: Encoder of convolutional codes

- Use sequential logic to implement convolutional codes.
- A information sequence u of infinite length is mapped to a codeword v of infinite length. In practice, we will output the convolutional codes by termination, truncation, or tailbiting.

- Assume we use feed forward encoder with memory m, then the codeword $c(i)$ of length n at time i is dependent on the current input $u(i)$ and previous m inputs, $u(i-1), \cdots, u(i-m)$.
- We need $m+1$ matrices $G_0, G_1, \cdots, G_m$ of size $k \times n$:

$$c(i) = u(i)G_0 + u(i-1)G_1 + u(i-2)G_2 + \cdots + u(i-m)G_m$$

- We denote this (linear) convolutional code by $C[n, k, m]$, usually, n and k are small.

Relation between block and convolutional codes

- A Convolutional code maps information blocks of length k to code blocks of length n. This linear mapping contains memory, because the code block depends on m previous information blocks.
- In this sense, block codes are a special case of convolutional codes, i.e., convolutional codes without memory.

- In practical application, convolutional codes have code sequences of finite length. When looking at the finite generator matrix of the created code in time domain, we find that it has a special structure.
- Because the generator matrix of a block code with corresponding dimension generally dose not have a special structure, convolutional codes with finite length can be considered as a special case of block codes.
- The trellis structure of convolutional codes is time-invariant, but the trellis structure of block codes is usually time-varying.

Scalar generator matrix in the time domain

G matrix of block codes

$$[u(0), u(1), u(2), \cdots] \begin{bmatrix} G & & & \\ & G & & \\ & & G & \\ & & & \ddots \end{bmatrix}$$
$$= [c(0), c(1), c(2), \cdots]$$

G matrix of convolutional codes

$$[u(0), u(1), u(2), \cdots] \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & & & \cdots \\ & G_0 & G_1 & \cdots & G_{m-1} & G_m & & \cdots \\ & & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & \cdots \\ & & & \cdots & \vdots & \vdots & \vdots & \cdots \\ & & & & G_1 & G_2 & G_3 & \cdots \\ & & & & G_0 & G_1 & G_2 & \cdots \\ & & & & & G_0 & G_1 & \cdots \\ & & & & & & G_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$
$$= [c(0), c(1), c(2), \cdots\cdots, c(m), c(m+1), \cdots]$$

- Here, we assume that the initial values in the m memories are all set to zeros.
- For example, the scalar matrix shows that

$$\begin{aligned} c(0) &= u(0)G_0 + u(-1)G_1 + u(-2)G_2 + \cdots + u(-m)G_m \\ &= u(0)G_0 \end{aligned}$$

  since $u(-1) = u(-2) = \cdots u(-m) = 0$.
- Similarly

$$\begin{aligned} c(1) &= u(1)G_0 + u(0)G_1 + u(-1)G_2 + \cdots + u(-m+1)G_m \\ &= u(1)G_0 + u(0)G_1. \end{aligned}$$

- In general, we have

$$\begin{aligned} c(i) &= u(i)G_0 + u(i-1)G_1 + u(i-2)G_2 + \cdots + u(i-m)G_m \\ &= u(i) \otimes G_i. \end{aligned}$$

Impulse response of MIMO LTI systems

- From the scalar G matrix representation, we have

$$\begin{aligned} c(i) &= u(i)G_0 + u(i-1)G_1 + u(i-2)G_2 + \cdots + u(i-m)G_m \\ &= u(i) \otimes G_i \end{aligned}$$

- This is the form of discrete time convolutional sum, i.e., the output $c(i)$ is the convolutional sum of input sequence $u(i)$ and the finite impulse response (FIR) $(G_0, \cdots, G_m)$.
- In the undergraduate course of signal and system, we deal with SISO.
- Here, we have MIMO LTI systems with k inputs and n outputs and thus we need $k \times n$ impulse responses

$$g_i^{(j)} = g_i^{(j)}(l), \quad 0 \le l \le m, \ \ 1 \le i \le k, \ \text{and } 1 \le j \le n,$$

which can be obtained from $m+1$ matrices $\{G_0, \cdots, G_M\}$.

- Correspondingly, we can associate each $g_i^{(j)}$ with its Fourier transform $G_i^{(j)}(D)$ and form a $k \times n$ matrix $G(D)$ by

$$G(D) = [G_i^{(j)}(D)] = G_0 + G_1 D + G_2 D^2 + \cdots + G_m D^m$$

- This is the polynomial matrix representation of a convolutional code.
- The $k \times n$ matrix $g(l)$ consisting of impulse responses $g_i^{(j)}(l)$ and the $k \times n$ matrix $G(D)$ consisting of $G_i^{(j)}(D)$ form a Fourier pair.

Example

A (3,2) convolutional code with impulse response $g(l)$ and transfer function $G(D)$:

$$g(l) = \begin{pmatrix} 110 & 111 & 100 \\ 010 & 101 & 111 \end{pmatrix}$$

$$G(D) = \begin{pmatrix} 1+D & 1+D+D^2 & 1 \\ D & 1+D^2 & 1+D+D^2 \end{pmatrix}$$

LTI system representation

- Let us use u_i, c_i, (instead of $u(i)$, $c(i)$) to denote the information sequence, code sequence respectively, at time i.
- I.e., the infinite information and code sequence is

$$\begin{cases} u = u_0 u_1 u_2 \cdots u_i \cdots \\ v = v_0 v_1 v_2 \cdots v_i \cdots \end{cases}$$

- The u_i consists of k bits and v_i consists of n bits denoted by

$$\begin{cases} u_i = u_i^{(1)} u_i^{(2)} u_i^{(3)} \cdots u_i^{(k)} \\ v_i = v_i^{(1)} v_i^{(2)} v_i^{(3)} \cdots v_i^{(n)} \end{cases}$$

- Define the input sequence due to the ith stream, $1 \le i \le k$, as
$$u^{(i)} = u_0^{(i)} u_1^{(i)} u_2^{(i)} u_3^{(i)} \cdots$$
and the output sequence due to the jth stream, $1 \le j \le n$, as
$$v^{(j)} = v_0^{(j)} v_1^{(j)} v_2^{(j)} v_3^{(j)} \cdots$$
- A $[n, k, m]$ convolutional code can be represented as a MIMO LTI system with k input streams
$$(u^{(1)}, u^{(2)}, \cdots, u^{(k)}),$$
and n output streams
$$(v^{(1)}, v^{(2)}, \cdots, v^{(n)}),$$
and a $k \times n$ impulse response matrix $g(l) = \{g_i^{(j)}(l)\}$.

- The jth of the n output sequence $v^{(j)}$ is obtained by convolving the input sequence with the corresponding system impulse response
$$v^{(j)} = u^{(1)} \otimes g_1^{(j)} + u^{(2)} \otimes g_2^{(j)} + \cdots u^{(k)} \otimes g_k^{(j)} = \sum_{i=1}^{k} u^{(i)} \otimes g_i^{(j)}$$
- This is the origin of the name convolutional code.
- The impulse response $g_i^{(j)}$ of the ith input with the response to the jth output is found by stimulating the encoder with the discrete impulse $(1000\cdots)$ at the ith input and by observing the jth output when all other inputs are set to $(0000\cdots)$.

Polynomial generator matrix in frequency domain

Now introduce the delay operator D in the representation of input sequence, output sequence, and impulse response, i.e.,

1. Use z transform
$$u^{(i)} = u_0^{(i)} u_1^{(i)} u_2^{(i)} u_3^{(i)} \cdots \longleftrightarrow U_i(D) = \sum_{t=0}^{\infty} u_t^{(i)} D^t$$
$$v^{(j)} = v_0^{(j)} v_1^{(j)} v_2^{(j)} v_3^{(j)} \cdots \longleftrightarrow V_i(D) = \sum_{t=0}^{\infty} v_t^{(j)} D^t$$
$$g_i^{(j)} = (g_i^{(j)}(0), \cdots, g_i^{(j)}(m)) \longleftrightarrow G_i^{(j)}(D) = \sum_{l=0}^{m} g_i^{(j)}(l) D^l$$
2. $z\{u * g\} = U(D)G(D) = V(D)$
3. $V_j(D) = \sum_{i=1}^{k} U_i(D) \cdot G_i^{(j)}(D)$

We thus have

$$V(D) = U(D) \cdot G(D)$$

, where

$$\begin{aligned} U(D) &= (U_1(D), U_2(D), \ldots, U_k(D)) \\ V(D) &= (V_1(D), V_2(D), \ldots, V_n(D)) \\ G(D) &= \begin{pmatrix} \\ G_i^{(j)}(D) \\ \\ \end{pmatrix} \end{aligned}$$

Example 1

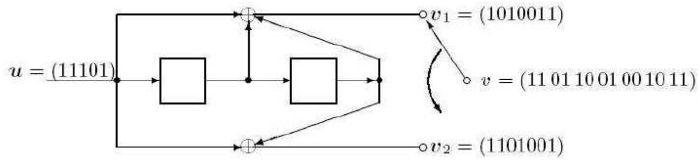


Figure 5: (2,1,2) convolutional code encoder

Input: $u = (1, 1, 1, 0, 1)$ in time domain
In z domain:

$$\begin{aligned} G^{(1)}(D) &= 1 + D + D^2 \\ G^{(2)}(D) &= 1 + D^2 \\ G(D) &= [1 + D + D^2, 1 + D^2] \\ U(D) &= 1 + D + D^2 + D^4 \\ V(D) &= U(D) \bullet G(D) \\ V_1(D) &= 1 + D^2 + D^5 + D^6 \\ V_2(D) &= 1 + D + D^3 + D^6 \end{aligned}$$

In time domain:

$$\begin{aligned} v_1 &= (1, 0, 1, 0, 0, 1, 1) \\ v_2 &= (1, 1, 0, 1, 0, 0, 1) \end{aligned}$$

Example 2

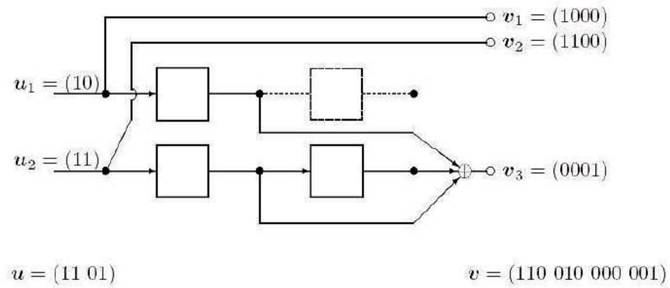


Figure 6: (3,2,2) convolutional code encoder

$$
\begin{aligned}
V_1(D) &= U_1(D) \\
V_2(D) &= U_2(D) \\
V_3(D) &= U_1(D) \bullet D + U_2(D) \bullet (D + D^2) \\
\begin{bmatrix} V_1(D) & V_2(D) & V_3(D) \end{bmatrix} &= \begin{bmatrix} U_1(D) & U_2(D) \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & D \\ 0 & 1 & D + D^2 \end{bmatrix} \\
G_1(D) &= \begin{bmatrix} 1 & 0 & D \\ 0 & 1 & D + D^2 \end{bmatrix}
\end{aligned}
$$

$$U_1 = 1 \quad U_2 = 1 + D$$

$$V = \begin{bmatrix} 1 & 1+D \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & D \\ 0 & 1 & D + D^2 \end{bmatrix} = \begin{bmatrix} 1 & 1+D & D^3 \end{bmatrix}$$

$$v_1 = (1, 0, 0, 0) \quad v_2 = (1, 1, 0, 0) \quad v_3 = (0, 0, 0, 1)$$

State diagram, tree, and trellis

State diagram

- Convolutional code 的編碼器，可看成一個 finite state machine
- 輸入(shift register)的內容，可用來描述 states，輸出 v_t 在時間 t 的值，由當時所在的 state σ_t 與 輸入 u_t 來決定
- 在 state diagram 上， nodes 為可能的 state ，路徑上標示著輸入與輸出 (u_t/v_t)

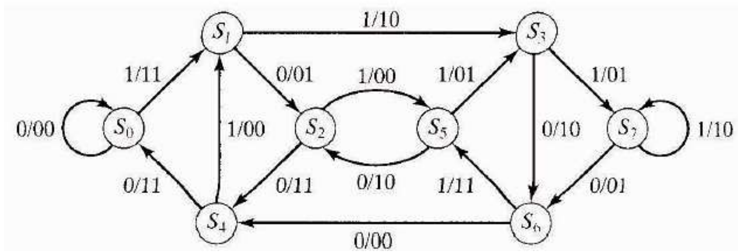


Figure 7: state diagram of (2,1,2) Convolutional code

Code Tree of Convolutional code

- 一個 (n,k,m) Convolutional code 的 codeword 可視為 code tree 上面的一個路徑
- 輸入的長度為 h ， code tree 會有 $(h+m+1)$ 個 level ，最左邊的 node(level 0) 稱為 origin node
- 在最先的 h levels， 每個 node 存在 2^k 個 branch ，位在 level (h+m)，最右邊的那 2^{hk} 個 nodes ，稱為 terminal nodes
- 從 origin node 到 terminal 的路徑稱為 code path，可用來表示一個 code word

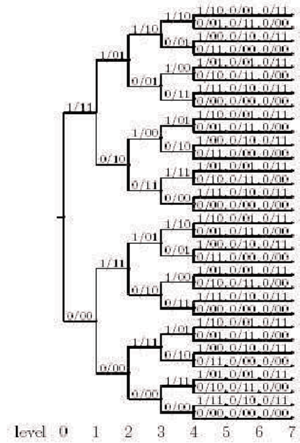


Figure 8: state tree of (2,1,2) Convolutional code

Trellises of Convolutional code

- 將 code tree 上的 node，有結構性的合併在相同的 state ，稱為 Convolutional code 的 code trellis
- 對於一 (n,k,m) 的 Convolutional code，state 的數量在 level m 為 2^K，當 $K=\sum_{j=1}^{k} K_j$,K_j 為第 j 個輸入的長度，在此 level 上，存在 2^K 個 nodes
- terminal node 只有一點，且會回到最初的狀態
- 從 origin node 到 terminal node 的路徑，可用來表示一個 codeword

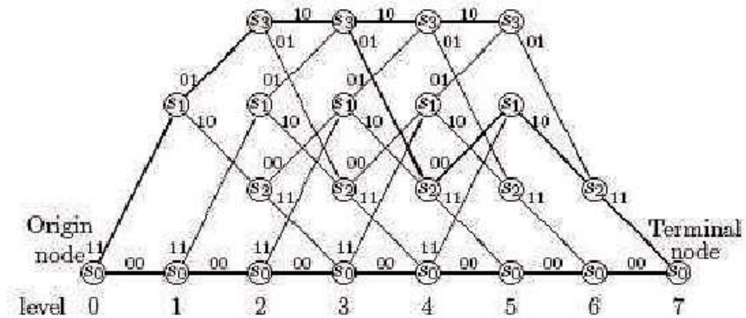


Figure 9: trellis of (2,1,2) Convolutional code

Structural properties of Convolutional codes

Convolutional encoder is a linear sequential circuit, it's operation can be describe by a state diagram.The state of an encoder is defined as its shift register contents.

For an (n,k,v) encoder
The encoder state σ_l at time unit l is the binary v-tuple

$$\sigma_l = (s_{l-1}^{(1)} s_{l-2}^{(1)} \dots s_{l-v_1}^{(1)} s_{l-1}^{(2)} s_{l-2}^{(2)} \dots s_{l-v_2}^{(2)} \cdots s_{l-1}^{(k)} s_{l-2}^{(k)} \dots s_{l-v_k}^{(k)})$$

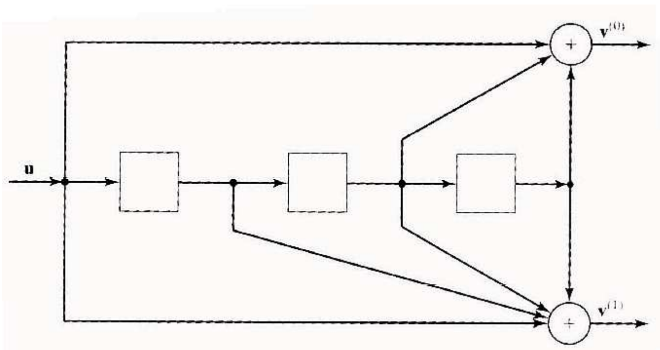
Each branch in the state diagram is labeled with the k inputs

$$(u_l^{(1)}, u_l^{(2)}, \dots, u_l^{(k)})$$

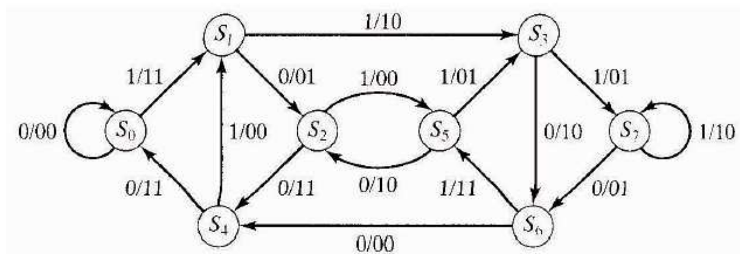
causing the transition and the n corresponding output

$$(v_l^{(1)}, v_l^{(2)}, \dots, v_l^{(n-1)})$$

(2,1,3) encoder



State diagrams for (2,1,3) encoder



Construction of minimal trellis

Consider an (n, k, m, d) Convolutional code. The trellis is principle infinite, but it has a very regular structure, consisting(after a short initial transient) of repeated copies of what we shall call the trellis module associated with $G(D)$.

- The trellis module consists of 2^m initial state and 2^m final states, with each initial state being connected by a directed edge to exactly 2^k final state.Thus the trellis module has 2^{k+m} edge
- Each edge is label with an n-symbol binary vector,namely the output produced by the encoder in response to the given state transition
- Each edge has length n, so the total edge length of the Convolutional trellis module is $n \cdot 2^{k+m}$
- Conventional trellis complexity of the trellis module can be defined as

$$\frac{n}{k} \cdot 2^{m+k}$$

Ex: The $(3, 2, 2)$ Convolutional code with canonical generator matrix given by

$$G_1(D) = \begin{pmatrix} 1+D & 1+D & 1 \\ D & 0 & 1+D \end{pmatrix}$$

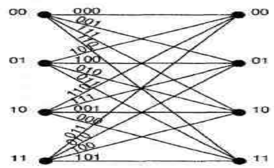


Figure 10: The trellis module of (3,2,2) Convolutional code

The total number of edge symbol is

$$3 \cdot 2^{2+2} = 48$$

so that the Convolutional trellis complexity corresponding to the matrix $G_1(D)$ is $48/2 = 24$ symbols per bits.

We can do substantially better than this, if we use the punctured Convolutional code.

- Begin with a $(N, 1, m)$ Convolutional code, and block it to depth k, i.e., group the input bit stream into blocks of k bits each, the result is an (Nk, k, m) Convolutional code
- Delete, or puncture, all but n symbols form each Nk-symbol output block,the result is an (n, k, m) Convolutional code.

- The punctured code can be represented by a trellis whose trellis module is built from k copies of the trellis module from the parent $(N, 1, m)$ code, each of which has only 2^{m+1} symbols
- The total number of symbols on the trellis module is $n \cdot 2^{m+1}$
- the trellis complexity of an (n, k, m) punctured code is

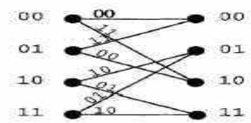
  $$\frac{n}{k} \cdot 2^{m+1}$$

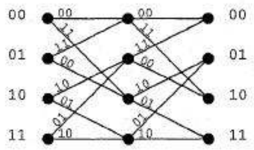
  which is a factor of 2^{k-1} smaller than the complexity of the conventional trellis

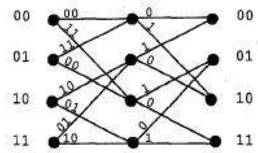
Ex:The $(2, 1, 2, 5)$ Convolutional code defined by the canonical generator matrix

$$G_2(D) = \begin{pmatrix} 1 + D + D^2 & 1 + D^2 \end{pmatrix}$$

The trellis module:







If $G(D)$ is a canonical generator matrix for an (n, k, m) Convolutional code C, then we can write $G(D)$ in the form

$$G(D) = G_0 + G_1 D + \cdots + G_L D^L$$

where $G_0, \ldots G_L$ are $k \times n$ scalar matrices,and L is the maximum degree of any entry of $G(D)$. The integer L is called the memory of the code

If we concatenate the $L+1$ matrices $G_0 G_1 \cdots G_L$, we obtain a $k \times (L+1)n$ scalar matrix

$$\tilde{G} = (G_0 G_1 \cdots G_L)$$

and its shifts can built a scalar generator matrix G_{scalar} for the code C

$$G_{scalar} = \begin{pmatrix} G_0 & G_1 & G_2 & & & \\ & G_0 & G_1 & G_2 & & \\ & & G_0 & G_1 & G_2 & \\ & & & G_0 & G_1 & G_2 \\ & & & & \ddots & \end{pmatrix}$$

The trellis module for the trellis associated with G_{scalar} corresponds to the $(L+1)k \times n$ matrix module

$$\hat{G} = \begin{pmatrix} G_L \\ G_{L-1} \\ \vdots \\ G_0 \end{pmatrix}$$

which repeatedly appears as a vertical slice in G_{scalar}

It is easy to show that the number of edge symbols in this trellis module is

$$\text{edge symbol count}= \sum_{j=1}^{n} 2^{a_j}$$

where a_j is the number of active entries in the jth column of the matrix $\hat{G}$

EX:consider the $(3,2,1)$ code with generator matrix

$$G_3(D) = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1+D & 1+D \end{pmatrix}$$

The scalar matrix $\tilde{G_3}$ corresponding to $G_3(D)$ is

$$\tilde{G_3} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

The matrix module corresponding to $\tilde{G_3}$ is

$$\hat{G_3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Thus $a_1 = 3$, $a_2 = 3$,and $a_3 = 3$
The corresponding trellis module has $2^3 + 2^3 + 2^3 = 24$ edge symbols
The resulting trellis complexity is $24/2 = 12$ symbols per bits

If we add the first row of $G_3(D)$ to the second row, the resulting generator matrix, which is still canonical is

$$G_3'(D) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1+D & D \end{pmatrix}$$

The scalar matrix $\tilde{G_3}'$ corresponding to $G_3'(D)$ is

$$\tilde{G_3}' = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

The matrix module corresponding to $\tilde{G_3}'$ is

$$\hat{G_3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Thus $a_1 = 2$, $a_2 = 3$,and $a_3 = 3$
The corresponding trellis module has $2^2 + 2^3 + 2^3 = 20$ edge symbols
The resulting trellis complexity is $20/2 = 12$ symbols per bits

But we can do it still better. If we multiply the first row of $G_3'(D)$ by D and add it to the second row, the resulting generator matrix, which is still canonical is

$$G_3''(D) = \begin{pmatrix} 1 & 0 & 1 \\ D & 1+D & 0 \end{pmatrix}$$

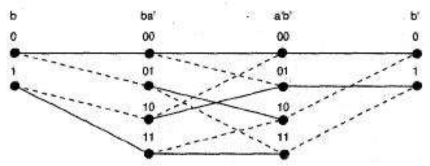
The scalar matrix $\tilde{G_3}''$ corresponding to $G_3''(D)$ is

$$\tilde{G_3} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

The matrix module corresponding to $\tilde{G_3}''$ is

$$\hat{G_3}'' = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Thus $a_1 = 2$, $a_2 = 3$,and $a_3 = 2$
The corresponding trellis module has $2^2 + 2^3 + 2^2 = 16$ edge symbols
The resulting trellis complexity is $16/2 = 8$ symbols per bits



In this sample, the ratio of the Convolutional trellis complexity to the minimal trellis complexity is $12/8 = 3/2$.

If this code were punctured, the ratio would be at least 2.

It is easy to see that there is no generator matrix for this code with spanlength less than 7, so that the trellis module shown below yields the minimal trellis for the code.

Alternatively, we can examine the scalar generator matrix for the code corresponding to $\tilde{G}_3''$

$$G_{scalar} = \begin{pmatrix} \underline{1} & 0 & \bar{1} & 0 & 0 & 0 & & & & & \\ 0 & \underline{1} & 0 & 1 & \bar{1} & 0 & & & & & \\ & & & \underline{1} & 0 & \bar{1} & 0 & 0 & 0 & & \\ & & & 0 & \underline{1} & 0 & 1 & \bar{1} & 0 & & \\ & & & & & & \underline{1} & 0 & \bar{1} & 0 & 0 & 0 \\ & & & & & & 0 & \underline{1} & 0 & 1 & \bar{1} & 0 \\ & & & & & & & & & & \ddots & \end{pmatrix}$$

we see that the G_{scalar} has the property that no column contains more than one underline entry, or more than one overline entry. Thus G_{scalar} has the LR property

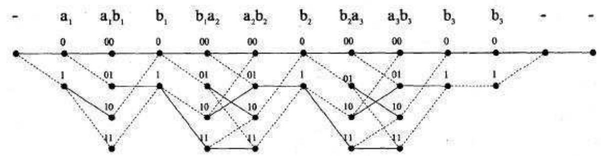
G_{scalar} is infinite, we can define the Mth truncation of the code C, denoted by $C^{[M]}$, as the $[(M+L)n, Mk]$ block code.

$$G_{scalar}^{[M]} = \begin{pmatrix} \underline{1} & 0 & \bar{1} & 0 & 0 & 0 & & & & & & \\ 0 & \underline{1} & 0 & 1 & \bar{1} & 0 & & & & & & \\ & & & \underline{1} & 0 & 1 & 0 & 0 & 0 & & & \\ & & & 0 & \underline{1} & 0 & 1 & \bar{1} & 0 & & & \\ & & & & & & & & \ddots & & & \\ & & & & & & \underline{1} & 0 & \bar{1} & 0 & 0 & 0 \\ & & & & & & 0 & \underline{1} & 0 & 1 & \bar{1} & 0 \end{pmatrix}$$

EX: $M = 3$ truncation, with corresponding scalar matrix

$$G^{[3]}_{scalar} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & & & & & & \\ 0 & 1 & 0 & 1 & 1 & 0 & & & & & & \\ & & & 1 & 0 & 1 & 0 & 0 & 0 & & & \\ & & & 0 & 1 & 0 & 1 & 1 & 0 & & & \\ & & & & & & 1 & 0 & 1 & 0 & 0 & 0 \\ & & & & & & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

This is the generator matrix for a $[12, 6]$ block code.the trellis of $G^{[3]}_{scalar}$ is shown below.



This trellis consists of two copies of the minimal trellis module of $G''_3(D)$, glued together,plus initial and final transient sections.

Generator matrix $G(D)$ produces a minimal trellis if and only if $G(D)$ has the property that the spanlength of the corresponding G cannot be reduced by an operation of the form

$$g_i(D) \leftarrow g_i(D) + D^l g_j(D)$$

Where $g_i(D)$ is the ith row of $G(D)$, and l is an integer in the range $0 \leq l \leq L$

The algebraic theory of Convolutional codes

Abstract

We defined in (n, k) Convolutional code C to be a k-dimensional subspace of $F(D)^n$, and a generator matrix of C to be a $k \times n$ matrix over $F(D)$, whose rowspace is C. We will discuss the various kinds of generator matrices that a Convolutional code can have, and settle on the **canonical** polynomial generator matrices as the preferred kind.

- PGM:If the entries of $G(D)$ are polynomials, then $G(D)$ is called a **polynomial generator matrix**.
- **Any Convolutional code has a polynomial generator matrix**, since if G is an arbitrary generator matrix for C, the matrix obtained from G by multiplying each row by the least common multiple of the denominators of the entries in that row is a PGM for C.

- Let $G(D) = (g_{ij}(D))$ be a $k \times n$ PGM for C. We now define the internal degree and external degree of $G(D)$ as follows
  - **intdeg** $G(D)$ = maximum degree of $G(D)$'s $k \times k$ minors.
  - **extdeg** $G(D)$ = sum of the row degrees of $G(D)$.

The following two definitions will be essential in our discussion of Convolutional codes.

- **DEFINITION.** A $k \times n$ polynomial matrix $G(D)$ is called **basic** if, among all polynomial matrices of the form $T(D)G(D)$, where $T(D)$ is a nonsingular $k \times k$ matrix over $F(D)$, it has the minimum possible internal degree.
- **DEFINITION.** A $k \times n$ polynomial matrix $G(D)$ is called **reduced** if, among all polynomial matrices of the form $T(D)G(D)$, where $T(D)$ is unimodular, $G(D)$ has the minimum possible external degree. Since any unimodular matrix is a product of elementary matrices, an equivalent definition is that a matrix is reduced if its external degree cannot be reduced by a sequence of elementary row operations.

- **THEOREM.** Let $G(D)$ be a $k \times n$ polynomial matrix.

  1. If $T(D)$ is any nonsingular $k \times k$ polynomial matrix, then

  $$\texttt{intdeg}T(D)G(D) = \texttt{intdeg}G(D) + \texttt{deg det}T(D)$$

  In particular, $\texttt{indeg}T(D)G(D) \geq \texttt{indeg}G(D)$, with equality if and only if $T(D)$ is unimodular.

  2. $\texttt{intdeg}G(D) \leq \texttt{exteg}G(D)$.

- **THEOREM.** A $k \times n$ polynomial matrix $G(D)$ is basic if and only if any one of the following six conditions is satisfied.

  1. The invariant factors of $G(D)$ are all 1.
  2. The gcd of the $k \times k$ minors of $G(D)$ is 1.
  3. $G(\alpha)$ has rank k for any α in the algebraic closure of F.
  4. $G(D)$ has a right $F[D]$ inverse, i.e. there exists a $n \times k$ polynomial matrix $H(D)$ such that $G(D)H(D) = I_k$.
  5. If $x(D) = u(D)G(D)$, and if $x(D) \in F[D]^n$, then $u(D) \in F[D]^k$.
  6. $G(D)$ is a submatrix of a unimodular matrix, i.e. there a $(n-k) \times n$ matrix $L(D)$ such that the determinant of the $n \times n$ matrix $\binom{G(D)}{L(D)}$ is a nonzeros element of F.

- **THEOREM.** A $k \times n$ polynomial matrix $G(D)$ is reduced if and only if any one of the following three conditions is satisfied.

  1. If we define the "indicator matrix for the highest-degree terms in each row" $\overline{G}$ by

  $$\overline{G}_{ij} = \texttt{coeff}_{D^{e_i}} g_{ij}(D)$$

  where e_i is the degree of $G(D)$'s i-th row, then $\overline{G}$ has rank k.

  2. $\texttt{extdeg}G(D) = \texttt{intdeg}G(D)$
  3. The "predictable degree property": For any k-dimensional polynomial vector, i.e. any $u(D) = (u_1(D), ..., u_k(D)) \in F[D]^k$

  $$\texttt{deg}(u(D)G(D)) = \max_{1 \leq i \leq k}(\texttt{deg}u_i(D) + \texttt{deg}g_i(D))$$

EXAMPLE. Here are eight generator matrices for a $(4, 2)$ Convolutional code over $GF(2)$.

1.

$$G_1 = \begin{bmatrix} \frac{1}{1+D+D^2} & 1 & \frac{1+D^2}{1+D+D^2} & \frac{1+D}{1+D+D^2} \\ 1 & \frac{1+D+D^2}{D} & D & \frac{1}{D} \end{bmatrix}$$

Basic× Reduced× intdeg× extdeg×

2.

$$G_2 = \begin{bmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{bmatrix}$$

Basic× Reduced× intdeg3 extdeg4

3.

$$G_3 = \begin{bmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 0 & 1+D & D & 1 \end{bmatrix}$$

Basic✓ Reduced× intdeg 1 extdeg 3

4.

$$G_4 = \begin{bmatrix} 1 & D & 1+D & 0 \\ 0 & 1+D & D & 1 \end{bmatrix}$$

Basic✓ Reduced× intdeg 1 extdeg 2

5.

$$G_5 = \begin{bmatrix} 1+D & 0 & 1 & D \\ D & 1+D+D^2 & D^2 & 1 \end{bmatrix}$$

Basic× Reduced✓ intdeg 3 extdeg 3

6.

$$G_6 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1+D & D & 1 \end{bmatrix}$$

Basic✓ Reduced✓ intdeg 1 extdeg 1

7.

$$G_7 = \begin{bmatrix} 1+D & 0 & 1 & D \\ 1 & D & 1+D & 0 \end{bmatrix}$$

Basic× Reduced✓ intdeg 2 extdeg 2

8.

$$G_8 = \begin{bmatrix} 1 & 0 & \frac{1}{1+D} & \frac{D}{1+D} \\ 0 & 1 & \frac{D}{1+D} & \frac{1}{1+D} \end{bmatrix}$$

Basic× Reduced× intdeg× extdeg×

- **DEFINITION.** Among all PGM's for a given Convolutional code C, those for which the external degree is as small as possible are called canonical PGM's. This minimal external degree is called the degree of the code C, and denoted `deg` C.

- **THEOREM.** A PGM $G(D)$ for the Convolutional code C is canonical if and only if it is both basic and reduced.

- **COROLLARY.** The minimal internal degree of any PGM for a given convolution code C is equal to the degree of C.

- **COROLLARY.** If G is any basic generator matrix for C then `intdeg`$G=$ `deg`C.

- **THEOREM.** If $e_1 \le e_2 \le ... \le e_k$ are the row degrees of a canonical generator matrix $G(D)$ for a Convolutional code C, and if $f_1 \le f_2 \le ... \le f_k$ are the row degrees of any other polynomial generator matrix, say $G'(D)$, for C, then $e_i \le f_i$, for $i = 1, ..., k$.

- **THEOREM.** The set of row degrees is the same for all canonical PGM's for a given code.

- Where $(e_1 \le e_2 \le ... \le e_k)$ are called the **Forney indices** of the code.

- The maximum of the Forney indices is called the **memory** of the code.

- An (n, k, m) code is called **optimal** if it has the maximum possible free distanceamong all codes with the same value of n, k and m.

- **EXAMPLE**

  1.

$$G_6 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1+D & D & 1 \end{bmatrix}$$

`Basic`✓ `Reduced`✓ `intdeg` 1 `extdeg` 1

Forney indices are $(0, 1)$

The Smith form

- The goal of the Smith algorithm (form), which is often called the invariant-factor algorithm, is to take an arbitrary $k \times n$ matrix G (with $k \le n$) over a Euclidean domain R, and by a sequence of elementary row and column operations, to reduce G to a $k \times n$ diagonal matrix $\Gamma = diag(\gamma_1, \dots, \gamma_r)$, whose diagonal entries are the invariant factors of G, i.e. $\gamma_i = \Delta_i / \Delta_{i-1}$, where Δ_i is the gcd of the $i \times i$ minors of G. (We take $\Delta_0 = 1$ by convention)

- **Theorem**:
  Let $G(D)$ be a $b \times c$, $b \le c$, binary polynomial matrix (i.e., $G(D)$=$(g_{ij}(D))$, where $g_{ij}(D) \in \mathbb{F}_2[D]$, $1 \le i \le b$, $1 \le j \le c$) of rank r. Then $G(D)$ can be written in the following manner:

$$G(D) = A(D)\Gamma(D)B(D)$$

where $A(D)$ and $B(D)$ are $b \times b$ and $c \times c$, respectively, binary polynomial matrices with unit determinants, and where $\Gamma(D)$ is the $b \times c$ matrix

$$\Gamma(D) = \begin{bmatrix} \gamma_1(D) & & & & & & \\ & \gamma_2(D) & & & & & \\ & & \ddots & & & & \\ & & & \gamma_r(D) & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \ldots 0 \end{bmatrix}$$

which is called the smith form of $G(D)$, and whose nonzero elements $\gamma_i(D) \in \mathbb{F}_2[D]$, $1 \le i \le r$, called the *invariant-factors* of $G(D)$, are unique polynomials that satisfy

$$\gamma_i(D) | \gamma_{i+1}(D), \quad i = 1, 2, \ldots, r-1$$

- Moreover, if we let $\Delta_i(D) \in \mathbb{F}_2[D]$ be the determinantal divisor of $G(D)$, that is, the greatest common divisor (gcd) of the $i \times i$ subdeterminants (minors) of $G(D)$, then

$$\gamma_i(D) = \frac{\Delta_i(D)}{\Delta_{i-1}(D)}$$

where $\Delta_0(D) = 1$ by convention and $i = 1, 2, \ldots, r$.

- Two types of elementary operations:
  - Type I
    The interchange of two rows(or two columns).
  - Type II
    The addition to all elements in one row (or column) of the corresponding elements in another row (or column) multiplied by a fixed polynomial in D.

- Example: To obtain the Smith form of the polynomial encoder illustrated in Figure 1 we start with its encoding matrix

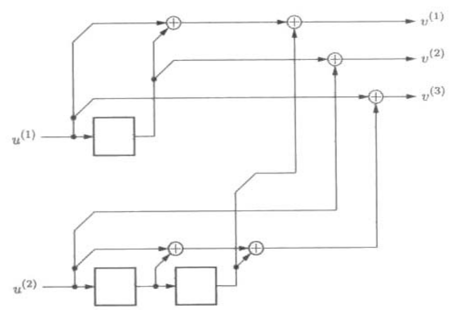


Figure 11: A rate $R = 2/3$ Convolutional encoder.

$$G(D) = \begin{bmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{bmatrix}$$

and interchange columns 1 and 3:

$$\begin{bmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & D & 1+D \\ 1+D+D^2 & 1 & D^2 \end{bmatrix}$$

Now the element in the upper-left corner has minimum degree.

To clear the rest of the first row, we can proceed with two operations simultaneously:

$$\begin{bmatrix} 1 & D & 1+D \\ 1+D+D^2 & 1 & D^2 \end{bmatrix} \begin{bmatrix} 1 & D & 1+D \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 1+D+D^2 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{bmatrix}$$

Next, we clear the rest of the first column:

$$\begin{bmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1+D+D^2 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{bmatrix}$$

We divide $1+D^2+D^3$ by $1+D+D^2+D^3$:

$$1+D^2+D^3 = (1+D+D^2+D^3)1 + D$$

Thus, we add column 2 to column 3 and obtain:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & 1+D^2+D^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & D \end{bmatrix}$$

Now we interchange column 2 and 3:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1+D+D^2+D^3 & D \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1+D+D^2+D^3 \end{bmatrix}$$

Repeating the previous step gives

$$1+D+D^2+D^3 = D(1+D+D^2)+1$$

and, hence, we multiply column 2 by $1+D+D^2$, add the product to column 3, and obtain

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & D & 1+D+D^2+D^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1+D+D^2 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Again we should interchange column 2 and 3:

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

and, finally, by adding D times the column 2 to column 3 we obtain the Smith form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \Gamma(D)$$

All invariant factors for this encoding matrix are equal to 1. By tracing these steps backward and multiplying $\Gamma(D)$ with the inverses of the elementary matrices (which are the matrices themselves), we obtain the matrix:

$$G(D) = A(D)\Gamma(D)B(D)$$

$$= \begin{bmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{bmatrix} \Gamma(D) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & D \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1+D+D^2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\times \begin{bmatrix} 1 & D & 1+D \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

and we conclude that

$$A(D) = \begin{bmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{bmatrix}$$

and

$$B(D) = \begin{bmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{bmatrix}$$

Thus, we have the following decomposition of the encoding matrix $G(D)$:

$$G(D) = \begin{bmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\times \begin{bmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{bmatrix}$$

The extension of the Smith form

- Let $G(D)$ be a $b \times c$ rational function matrix, and let $q(D) \in \mathbb{F}_2[D]$ be the least common multiple (lcm) of all denominators in $G(D)$. Then $q(D)G(D)$ is a polynomial matrix with Smith form decomposition

$$q(D)G(D) = A(D)\Gamma_q(D)B(D)$$

Dividing through by $q(D)$, we obtain the so-called invariant-factor decomposition of the rational matrix $G(D)$:

$$G(D) = A(D)\Gamma(D)B(D)$$

where

$$\Gamma(D) = \Gamma_q(D)/q(D)$$

with entries in $\mathbb{F}_2(D)$.

Thus

$$\Gamma(D) = \begin{bmatrix} \frac{\gamma_1(D)}{q(D)} & & & & & & \\ & \frac{\gamma_2(D)}{q(D)} & & & & & \\ & & \ddots & & & & \\ & & & \frac{\gamma_r(D)}{q(D)} & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \ldots 0 \end{bmatrix}$$

where $\frac{\gamma_1(D)}{q(D)}, \frac{\gamma_2(D)}{q(D)}, \frac{\gamma_r(D)}{q(D)}$ are called the invariant-factors of $G(D)$.

- Let

$$\frac{\gamma_i(D)}{q(D)} = \frac{\alpha_i(D)}{\beta_i(D)}, \quad i = 1, 2, \ldots, r$$

where the polynomials $\alpha_i(D)$ and $\beta_i(D)$ are relatively prime. Since $\gamma_i(D)|\gamma_{i+1}(D), \ i = 1, 2, \ldots, r-1$; that is,

$$q(D)\frac{\alpha_i(D)}{\beta_i(D)}|q(D)\frac{\alpha_{i+1}(D)}{\beta_{i+1}(D)}$$

we have

$$\alpha_i(D)\beta_{i+1}(D)|\alpha_{i+1}(D)\beta_i(D)$$

$$\Rightarrow \alpha_i(D)|\alpha_{i+1}(D) \text{ and } \beta_{i+1}(D)|\beta_i(D), \ i = 1, 2, \ldots, r-1$$

- **Example**:
The rate $R = \frac{2}{3}$ rational encoding matrix

$$G(D) = \begin{bmatrix} \frac{1}{1+D+D^2} & \frac{D}{1+D^3} & \frac{1}{1+D^3} \\ \frac{D^2}{1+D^3} & \frac{1}{1+D^3} & \frac{1}{1+D} \end{bmatrix}$$

has

$$q(d) = lcm(1 + D + D^2, \ 1 + D^3, \ 1 + D) = 1 + D^3$$

Thus, we have

$$q(D)G(D) = \begin{bmatrix} 1+D & D & 1 \\ D^2 & 1 & 1+D+D^2 \end{bmatrix}$$

Hence

$$G(D) = \begin{bmatrix} 1 & 0 \\ 1+D+D^2 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{1+D^3} & 0 & 0 \\ 0 & \frac{1}{1+D^3} & 0 \end{bmatrix}$$

$$\times \begin{bmatrix} 1+D & D & 1 \\ 1+D^2+D^3 & 1+D+D^2+D^3 & 0 \\ D+D^2 & 1+D+D^2 & 0 \end{bmatrix}$$

- The right-inverse of the generator matrix becomes

$$G^{-1}(D) = B^{-1}(D) \cdot \Gamma^{-1}(D) \cdot A^{-1}(D)$$

The inverted quadratic scrambler matrices $A^{-1}(D)$ and $B^{-1}(D)$ exist, since $A(D)$ and $B(D)$ have determinant 1.

Another version of extended Smith form

- Beginning with the matrix $G_0 = G$, it produces a sequence of $k \times n$ matrices $G_1, G_2, \ldots$, where G_{i+1} is derived from G_i by either an elementary row operation or an elementary column operation. We can represent this algebraically as

$$G_{i+1} = E_{i+1} G F_{i+1},$$

where E_{i+1} and F_{i+1} are $k \times k$ and $n \times n$ elementary matrices, respectively. If G_{i+1} is obtained from G_i via a row operation, then $F_{i+1} = I_n$, but if $G_{i+1}$is obtained from G_i via a column operation, then $E_{i+1} = I_k$. After a finite number N of steps, we obtain $G_N = \Gamma$.

- The extended Smith algorithm builds on the Smith algorithm. Whereas the Smith algorithm works only with the sequence $G_0, G_1, \ldots, G_N$, the extended Smith algorithm also works with a sequence of unimodular $k \times k$ matrices $X_0, \ldots, X_N$, and a sequence of unimodular $n \times n$ matrices $Y_0, \ldots, Y_N$.
- The sequences (X_i) and (Y_i) are initialized as $X_0 = I_k$, $Y_0 = I_n$, and updated via the rule

$$X_{i+1} = E_{i+1} X_i$$

$$Y_{i+1} = Y_i F_{i+1}$$

- The following simple lemma is the key to the extended Smith algorithm.

Lemma:

$$X_i G Y_i = G_i \texttt{ for } i = 0, 1, \ldots, N$$

- If we specialize above equation with $i = N$, we get

$$X_N G Y_N = \Gamma,$$

which is the desired "extended Smith diagonalization" of G.

- A convenient way to implement the extended Smith algorithm is to extend G to dimensions $(n+k)\times(n+k)$ as follows:

$$G' = \begin{bmatrix} G & I_k \\ I_n & 0_{n\times k} \end{bmatrix}$$

Then if the sequence of elementary row and column operations generated by the Smith algorithm applied to G are performed on the extended matrix G', after i iterations, the resulting matrix G'_i has the form

$$G'_i = \begin{bmatrix} G_i & X_i \\ Y_i & 0_{n\times k} \end{bmatrix}$$

- Example:

$$G = G_0 = \begin{bmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{bmatrix}$$

Then the corresponding matrix G' is

$$G' = G'_0 = \begin{bmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D & 1 & 0 \\ D & 1+D+D^2 & D^2 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

we obtained G'_1, G'_2, and G'_3 from G'_0 by successively adding $(1+D+D^2)$ times column 1 to column 2, $(1+D^2)$ times column 1 to column 3, and $(1+D)$ times column 1 to column 4.

$$G'_1 = \begin{bmatrix} 1 & 0 & 1+D^2 & 1+D & 1 & 0 \\ D & 1+D^3 & D^2 & 1 & 0 & 1 \\ 1 & 1+D+D^2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$G'_2 = \begin{bmatrix} 1 & 0 & 0 & 1+D & 1 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1 & 0 & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$G'_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ D & 1+D^3 & D+D^2+D^3 & 1+D+D^2 & 0 & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 1+D & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Next, adding D times row 1 to row 2, we obtain

$$G_4^{'} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D^3 & D+D^2+D^3 & 1+D+D^2 & 0 & 1 \\ 1 & 1+D+D^2 & 1+D^2 & 1+D & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Interchanging columns 2 and 4, we obtain

$$G_5^{'} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & D+D^2+D^3 & 1+D^3 & 0 & 1 \\ 1 & 1+D & 1+D^2 & 1+D+D^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Finally, adding D times column 2 to column 3 and $(1+D)$ times column 2 to column 4, we compute successively

$$G_6^{'} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & 0 & 1+D^3 & D & 1 \\ 1 & 1+D & 1+D & 1+D+D^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & D & 0 & 0 & 0 \end{bmatrix}$$

$$G_7^{'} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1+D+D^2 & 0 & 0 & D & 1 \\ 1 & 1+D & 1+D & D & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & D & 1+D & 0 & 0 \end{bmatrix}$$

Thus the extended Smith decomposition of the original matrix G is given by

$$\begin{bmatrix} 1 & 0 \\ D & 1 \end{bmatrix} \cdot G \cdot \begin{bmatrix} 1 & 1+D & 1+D & D \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & D & 1+D \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{bmatrix}$$

- The matrices X, Y , and Γ, contain much valuable information about the code C and the generator matrix G. To extract this information, however, we need to define several useful "pieces" of these matrices, which we call Γ_k, $\tilde{\Gamma}_k$, K, and H:

$$\begin{aligned} \Gamma_k &= \text{leftmost } k \text{ columns of } \Gamma \\ &= diag(\Gamma_1, \ldots, \Gamma_k) \text{ (dimensions: } k \times k). \\ \tilde{\Gamma}_k &= \gamma_k \cdot \Gamma_k^{-1} = diag(\gamma_k/\gamma_1, \ldots, \gamma_k/\gamma_k) \text{(dimensions: } k \times k). \\ K &= \text{leftmost } k \text{ columns of } Y. \text{ (dimensions: } n \times k). \\ H &= \text{rightmost } r \text{ columns of } Y. \text{ (dimensions: } n \times r). \end{aligned}$$

- **Theorem**: With the matrices Γ_r, $\tilde{\Gamma}_r$, K, and H define as in above equations, we have the following:
  1. A basic encoder for C: $G_{basic} = \Gamma_k^{-1} XG$. (That is, G_{basic} is obtained by dividing the i-th row of XG by the invariant factor γ_i, for $i = 1, \ldots, k$.)
  2. A polynomial inverse for G_{basic}: K.
  3. A polynomial pseudo-inverse for G, with factor γ_k: $K\tilde{\Gamma}X$. (In particular, if G is already basic, i.e. if $\Gamma_k = I_k$, then KX is a polynomial inverse for G.)
  4. A basic encoder for $C^\perp$ (parity-check matrix for C) H^T.

- **Example**: To illustrate the results of this appendix, we consider the following generator matrix G for a (4,2) binary Convolutional code:

$$G = \begin{bmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ D & 1+D+D^2 & D^2 & 1 \end{bmatrix}.$$

We found the extended invariant-factor decomposition of G to be $XGY = \Gamma$, where

$$\Gamma = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1+D+D^2 & 0 & 0 \end{bmatrix}, \; X = \begin{bmatrix} 1 & 0 \\ D & 1 \end{bmatrix},$$

$$Y = \begin{bmatrix} 1 & 1+D & 1+D & D \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & D & 1+D \end{bmatrix}$$

.

Thus

$$\Gamma_k = \begin{bmatrix} 1 & 0 \\ 0 & 1+D+D^2 \end{bmatrix} \tilde{\Gamma}_k = \begin{bmatrix} 1+D+D^2 & 0 \\ 0 & 1 \end{bmatrix}$$

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1+D & 0 & 0 & 1 \end{bmatrix}^T H = \begin{bmatrix} 1+D & 0 & 1 & D \\ D & 1 & 0 & 1+D \end{bmatrix}^T$$

Using the prescriptions in Theorem, we now quickly obtain the following.

- A basic encoder for C:

$$G_{basic} = \Gamma_k^{-1} XG = \begin{bmatrix} 1 & 1+D+D^2 & 1+D^2 & 1+D \\ 0 & 1+D & D & 1 \end{bmatrix}$$

- A polynomial inverse for G_{basic}:

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1+D & 0 & 0 & 1 \end{bmatrix}$$

- A polynomial pseudo-inverse for G, with factor $\gamma_2 = 1 + D + D^2$:

$$K\tilde{\Gamma}_k X = \begin{bmatrix} 1 & 0 & 0 & D \\ 1+D & 0 & 0 & 1 \end{bmatrix}$$

- A (basic) encoder for $C^\perp$:

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

Free distance and path enumerator

Outline

- Distance measures
  - Row and column distance
  - Extended distance measures

Column distance

- Definition(Column distance):
  The column distance d_j^c of order j of a generator matrix G(D) is the minimum Hamming distance of the first j+1 code blocks of two codewords $v_{[j+1]} = (v_0, v_1, ..., v_j)$ and $v'_{[j+1]} = (v'_0, v'_1, ..., v'_j)$, where the information sequences used for encoding $u_{[j+1]} = (u_0, u_1, ..., u_j)$ and $u'_{[j+1]} = (u'_0, u'_1, ..., u'_j)$ differ in the first information block, i.e. $u_0 \neq u'_0$.

- Because of the linearity of Convolutional codes, the column distance is equal to the minimum Hamming weight of all sequences $v_{[j+1]}$:
  $$d_j^c = \min_{u_0 \neq 0} wt(v_{[j+1]})$$
  where $v_{[j+1]} = (v_0, v_1, ..., v_j)$ are the first j+1 code blocks of all possible code sequences with $u_0 \neq 0$.
- When we considering the semi-infinite generator matrix G of the code, we obtain
  $$d_j^c = \min_{u_0 \neq 0} wt(u_{[j+1]} \cdot G^c_{[j+1]})$$

with the $k(j+1) \times n(j+1)$ generator matrix

$$G^c_{[j+1]} = \begin{bmatrix} G_0 & G_1 & ... & G_j \\ & G_0 & & G_{j-1} \\ & & \ddots & \vdots \\ & & & G_0 \end{bmatrix}$$

- Definition(Distance profile of the generator matrix): The first $m+1$ values of the column distance $d = (d^c_0, d^c_1, ..., d^c_m)$ of a given generator matrix G(D) with memory m are referred to as the distance profile of the generator matrix.
- Definition(Minimum distance): The column distance d^c_m of order m of a given generator matrix $G(D)$ with memory m is called the minimum distance d_m.

- Consider the state diagram of the generator matrix $G(D)$. The paths of length $j+1$ that are used for the calculation of d^c_j leave the zero state σ_0 at time $t=0$ and are in an arbitrary state $\sigma_{j+1} \in (0, 1, ..., 2^v - 1)$ after $j+1$ state transition. Consequently, the path relevant for j=0 are a part of the paths relevant for j=1, and so on. Therefore the column distance is a monotonically increasing function in j,

$$d^c_0 \leq d^c_1 \leq .... \leq d^c_j \leq ... \leq d^c_\infty$$

where a finite limit d^c_∞ for $j \to \infty$ exist.

Row distance

- Definition(Row distance): The row distance d^r_j of order j of a generator matrix G(D) with memory m is the minimum Hamming distance of the j+m+1 code blocks of two codewords $v_{[j+m+1]} = (v_0, v_1, ..., v_{j+m})$ and $v'_{[j+m+1]} = (v'_0, v'_1, ..., v'_{j+m})$, where the two information sequences used for encoding $u_{[j+m+1]} = (u_0, u_1, ..., u_{j+m})$ and $u'_{[j+m+1]} = (u'_0, u'_1, ..., u'_{j+m})$ with $(u_{j+1}, ..., u_{j+m}) = (u'_{j+1}, ..., u'_{j+m})$ are different in at least the first coordinate of the j+1 information blocks.

- The row distance can also be calculated as the minimum of the Hamming weights of all sequences $v_{[j+m+1]}$, in analogy with the column distance:

$$d_j^r = \min_{u_0 \neq 0} wt(v_{[j+m+1]})$$

where $v_{[j+m+1]} = (v_0, ..., v_j, ..., v_{j+m})$ are terminated code sequences of length j+m+1.

- We obtain

$$d_j^r = \min_{u_0 \neq 0} wt(u_{[j+m+1]} G^r_{[j+m+1]})$$

with the information sequence $u_{[j+m+1]} = (u_0, ..., u_j, u_{j+1}, ..., u_{j+m})$, where $(u_{j+1}, ..., u_{j+m})$ is equal to the termination sequence.

- The $k(j+m+1) \times n(j+m+1)$ generator matrix $G^r_{[j+m+1]}$ is given by

$$G^r_{[j+m+1]} = \begin{bmatrix} G_0 & G_1 & G_2 & \dots & G_j & \dots & G_{j+m} \\ & G_0 & G_1 & & & & G_{j+m+1} \\ & & \ddots & & & & \vdots \\ & & & G_0 & G_1 & \dots & G_m \\ & & & & G_0 & \dots & G_m - 1 \\ & & & & & \ddots & \vdots \\ & & & & & & G_0 \end{bmatrix}$$

If the generator matrix G(D) of the code is polynomial then $G_j = 0$ for $j > m$ and $(u_{j+1}, ..., u_{j+m}) = 0$ is the termination sequence.

- There is one sequence that diverges from zero state at time $t = 0$ and returns to the zero state at time $t = m$ and that has a specific weight d_0^r.
- Increasing the sequence length to $j = 1$ corresponds to increasing the number of valid code sequences that must be considered in finding the minimum-weight sequence. If a code sequence is found for $j = 1$ that has a hamming weight less than d_0^r then it is selected. Otherwise, we choose the original code sequence, and remain in the zero loop of zero state for the required number of j transitions without increasing the Hamming weight d_0^r.
- Therefore the row distance is a monotonically decreasing function in j, and we can write

$$d_0^r \geq d_1^r \geq ... \geq d_j^r \geq ... \geq d_\infty^r$$

where the limit $j \to \infty$ gives $d_\infty^r > 0$.

- The paths in the state diagram that were used for calculating the column distance d_∞^c go from the zero state to the zero loop, which is, according to the Hamming weight of the necessary state transitions, the 'closest'. On the other hand, the row distance d_∞^r is the minimum Hamming weight of all paths in the state diagram from zero state to the zero state and therefore in the zero loop.

$$d_0^c \leq d_1^c \leq .... \leq d_j^c \leq .... \leq d_\infty^c \leq d_\infty^r \leq ... \leq d_j^r \leq ... \leq d_0^r$$

- Theorem(Limit of row and column distance):
If $G(D)$ is a non-catastrophic generator matrix then the following relation holds true for the limits of the row and column distances:

$$d_\infty^c = d_\infty^r$$

$\because$ In the derivation of equation

$$d_0^c \leq d_1^c \leq .... \leq d_j^c \leq ... \leq d_\infty^c$$

, it was shown that the path d_∞^c returns to the zero loop of the state diagram and remains there. Only one zero loop, namely from zero directly to zero, exists for noncatastrophic generator matrices. Therefore d_∞^c is the minimum Hamming weight of a path leaving from and returning to zero. It is equal to the row distance d_∞^r, as shown in the derivation of equation

$$d_0^r \geq d_1^r \geq ... \geq d_j^r \geq ... \geq d_\infty^r$$

Free distance

- Definition(Free distance):
  The free distance d_f of a Convolutional code C with rate $R = \frac{k}{n}$ is defined as the minimum distance between two arbitrary codewords of C:
  $$d_f = \min_{v \neq v'} dist(v, v')$$
  Due to linearity, the problem of calculating distances can be interpreted as the problem of calculating Hamming weight.
- Theorem(Row, column and free distance):
  If G(D) is a non-catastrophic generator matrix then the following equation holds for the limits of the row and column distances:
  $$d_\infty^c = d_\infty^r = d_f$$

- Example(Row and column distance):
  Figure 1 shows the column and row distances of the generator matrix $G(D) = (1 + D + D^2 + D^3 \;\; 1 + D^2 + D^3)$ of a Convolutional code with rate $R = \frac{1}{2}$ and m=3.

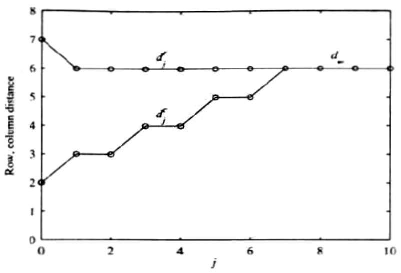


Figure 12: Row and column distance

$$\begin{aligned} G(D) &= (1\ 1) + (1\ 0)D + (1\ 1)D^2 + (1\ 1)D^3 \\ &= G_0 + G_1 D + G_2 D^2 + G_3 D^3 \end{aligned}$$

Column distance($j = 0, 1, 2, ..$)

$j = 0$:

$$G_{[1]}^c = [11] \Rightarrow d_0^c = \min_{u_0 \neq 0} wt(u_{[j+1]} \cdot G_{[j+1]}^c) = 2$$

$j = 1$:

$$G_{[2]}^c = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \Rightarrow d_1^c = 3$$

Row distance($j = 0, 1, 2, ...$)

$j = 0$:

$$G^r_{[j+m+1=0+3+1=4]} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\Rightarrow d^r_0 = \min_{u_0 \neq 0} wt(u_{[j+m+1]} \cdot G^r_{[j+m+1]}) = 7$$

$$\because (u_{j+1}, ..., u_{j+m}) = (u_1, u_2, u_3) = (000)$$

Extended distance measures

- Definition(Extended column distance):
  Let the generator matrix G of a Convolutional code C with rate $R = \frac{k}{n}$ be given. Then the extended column distance of order j is

$$d^{ec}_j = \min_{\sigma_0=0, \sigma_t \neq 0, 0<t\leq j} \{wt((u_0, u_1, ..., u_j)G^c_{[j+1]})\}$$

where

$$G^c_{[j+1]} = \begin{bmatrix} G_0 & G_1 & \cdots & G_m & & & \\ & G_0 & G_1 & \cdots & G_m & & \\ & & \ddots & \ddots & & \ddots & \\ & & & G_0 & G_1 & \cdots & G_m \\ & & & & G_0 & \cdots & G_{m-1} \\ & & & & & \ddots & \vdots \\ & & & & & & G_0 \end{bmatrix}$$

is the $k(j+1) \times n(j+1)$ truncated generator matrix.

- Just like the column distance d^c_j, the extended column distance d^{ec}_j is also a monotonically increasing function in j.
- For non-catastrophic generator matrices, the zero state and the consequently the associated zero loop are only permitted after j transitions, and a limit for $j \to \infty$ does not exist for the extended column distance.

Extended row distance

The state sequence $S = (\sigma_0, \sigma_1, ..., \sigma_t, ..., \sigma_j, \sigma_{j+1}, ..., \sigma_{j+m+1})$ used to define the code segments starts and ends in the zero state, i.e. $\sigma_0 = 0$ and $\sigma_{j+m+1} = 0$.

- Definition(Extended row distance):
  Let the generator matrix G of a Convolutional code C with rate $R = \frac{k}{n}$ be given. Then the extended row distance of order j is

$$d^{er}_j = \min_{u_j \neq 0, \sigma_0=0, \sigma_t \neq 0, 0<t\leq j} \{wt((u_0, u_1, ..., u_j)G^r_{[j+m+1]})\}$$

where

$$G^r_{[j+m+1]} = \begin{bmatrix} G_0 & G_1 & \cdots & G_m & & & \\ & G_0 & G_1 & \cdots & G_m & & \\ & & \ddots & \ddots & & \ddots & \\ & & & G_0 & G_1 & \cdots & G_m \end{bmatrix}$$

is a $k(j+m+1) \times n(j+1)$ truncated generator matrix.

Extended segment distance

The extended segment distance is defined by the state sequences $S = (\sigma_0, \sigma_1, ..., \sigma_m, \sigma_{m+1}, ..., \sigma_{j+m+1})$with σ_m and σ_{j+m+1} can assume arbitrary values. The state in between are not equal to the zero state.

- Definition(Extended segment distance):
  Let the generator matrix G of a Convolutional code C with rate $R = \frac{k}{n}$ be given. The extended segment distance of order j is

$$d_j^{es} = \min_{\sigma_t \neq 0, m < t \leq j+m} \{wt((u_0, u_1, ..., u_{j+m})G^s_{[j+1]})\}$$

where

$$G^s_{[j+1]} = \begin{bmatrix} G_m & & & \\ G_{m-1} & G_m & & \\ \vdots & G_{m-1} & \ddots & \\ G_0 & \vdots & \ddots & G_m \\ & G_0 & \cdots & G_{m-1} \\ & & \ddots & \vdots \\ & & & G_0 \end{bmatrix}$$

is a $k(j+1+m) \times n(j+1)$ truncated generator matrix.

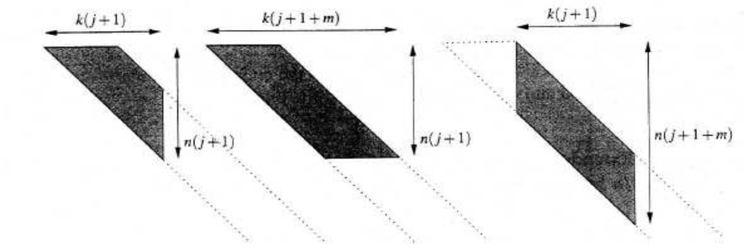


Figure 13: Graphical representation of the generator matrices corresponding to the extended distance properties of $G^{ec}_{[j+1]}$,$G^r_{[j+1]}$ and $G^s_{[j+1]}$(from left to right)

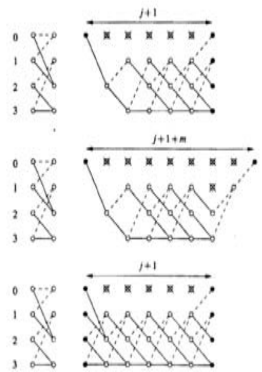
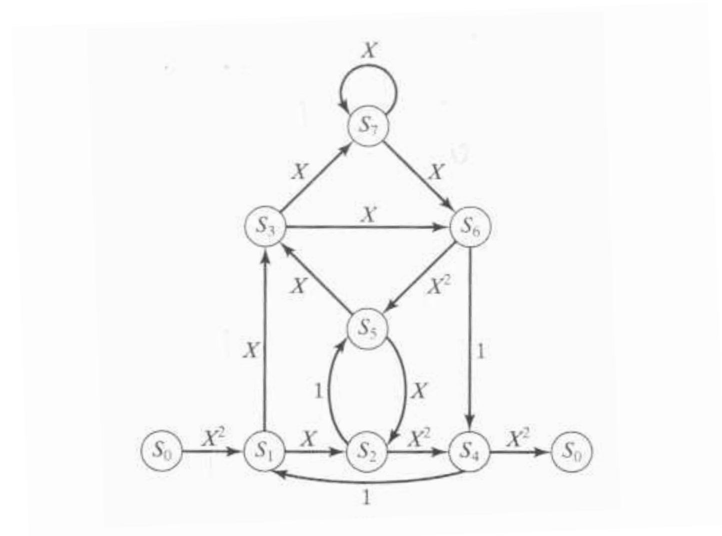


Figure 14: Diagrams showing the calculation of the extended distance measurements in trellis form:column, row and segment distances(from top to bottom).$j = 5$,$G(D) = (1 + D + D^2 \quad 1 + D)$,$m = 2$.

Weight Enumerating Function (WEF)

The state diagram can be modified to provide a complete description of the Hamming weights of all nonzero codewords, that is a codeword Weight Enumerating Function

- Zero weight branch around state S_0 is deleted
- Each branch is labeled with a branch gain X^d,where d is the weight of the n encoded bits on the branch
- The path gain is the product of the branch gains along a path, and the weight of the associated codeword is the power of X in the path gain



To determine the codeword WEF of a code by considering the modified state diagram of the encoder as a signal flow graph and applying Mason's gain formula to compute its transfer function

$$A(X) = \sum_d A_d X^d$$

A_d is the number of codewords of weight d.

- In a signal flow graph, a path connecting the initial state to the final state that does not go through any state twice is called a forward path
  (Let F_i be the gain of the ith forward path)
- A closed path starting at any state and returning to that state without going through any other state twice is called a cycle
  (Let C_i be the gain of the ith cycle)

let C_i be the gain of ith cycle, A set of cycles is nontouching if no state belongs to more than one cycle in the set.

- $\{i\}$ be the set of all cycles
- $\{i', j'\}$ be the set of all pairs of nontouching cycles
- $\{i''', j''', k''\}$ be the set of all triple of nontouching cycles, and so on

Define

$$\Delta = 1 - \sum_i C_i + \sum_{i',j'} C_{i'}C_{j'} - \sum_{i'',j'',k''} C_{i''}C_{j''}C_{k''} + \cdots$$

Δ_i is defined exactly like Δ butonly for that portion of the graph not touching the ith forward path

$$A(X) = \frac{\sum_i F_i \Delta_i}{\Delta}$$

Example:Computing the WEF of a (2,1,3) code

Cycle 1:	$s_1s_3s_7s_6s_5s_2s_4s_1$	$(c_1 = X^8)$
Cycle 2:	$s_1s_3s_7s_6s_4s_1$	$(c_2 = X^3)$
Cycle 3:	$s_1s_3s_6s_5s_2s_4s_1$	$(c_3 = X^7)$
Cycle 4:	$s_1s_3s_6s_4s_1$	$(c_4 = X^2)$
Cycle 5:	$s_1s_2s_5s_3s_7s_6s_4s_1$	$(c_5 = X^4)$
Cycle 6:	$s_1s_2s_5s_3s_6s_4s_1$	$(c_6 = X^3)$
Cycle 7:	$s_1s_2s_4s_1$	$(c_7 = X^3)$
Cycle 8:	$s_2s_5s_2$	$(c_8 = X)$
Cycle 9:	$s_3s_7s_6s_5s_3$	$(c_9 = X^5)$
Cycle 10:	$s_3s_6s_5s_3$	$(c_{10} = X^4)$
Cycle 11:	s_7s_7	$(c_{11} = X)$

There are 11 cycles

Cycle Pair 1:	(Cycle 2,Cycle 8)	$(c_2c_8 = X^4)$
Cycle Pair 2:	(Cycle 3,Cycle 11)	$(c_3c_{11} = X^8)$
Cycle Pair 3:	(Cycle 4,Cycle 8)	$(c_4c_8 = X^3)$
Cycle Pair 4:	(Cycle 4,Cycle 11)	$(c_4c_{11} = X^3)$
Cycle Pair 5:	(Cycle 6,Cycle 11)	$(c_6c_{11} = X^4)$
Cycle Pair 6:	(Cycle 7,Cycle 9)	$(c_7c_9 = X^8)$
Cycle Pair 7:	(Cycle 7,Cycle 10)	$(c_7c_{10} = X^7)$
Cycle Pair 8:	(Cycle 7,Cycle 11)	$(c_7c_11 = X^4)$
Cycle Pair 9:	(Cycle 8,Cycle 11)	$(c_8c_{11} = X^2)$
Cycle Pair 10:	(Cycle 10,Cycle 11)	$(c_{10}c_{11} = X^5)$

There are 10 pairs of nontouching cycles

Cycle Triple 1: (Cycle 4, Cycle 8, Cycle 11) $(C_4, C_8, C_11 = X^4)$

Cycle Triple 2: (Cycle 7, Cycle 10, Cycle 11) $(C_7, C_10, C_11 = X^8)$

There are 2 triples of nontouching cycles

There are no other sets of nontouching cycles. Therefore $\Delta = 1 - (x^8 + x^3 + x^7 + x^2 + x^4 + x^3 + x^3 + x + x^5 + x^4 + x) + (x^4 + x^8 + x^3 + x^3 + x^4 + x^8 + x^7 + x^4 + x^2 + x^5) - (x^4 + x^8) = 1 - 2x - x^3$

Forward Path 1: $s_0 s_1 s_3 s_7 s_6 s_5 s_2 s_4 s_0$ $(F_1 = X^{12})$

Forward Path 2: $s_0 s_1 s_3 s_7 s_6 s_4 s_0$ $(F_2 = X^7)$

Forward Path 3: $s_0 s_1 s_3 s_6 s_5 s_2 s_4 s_0$ $(F_3 = X^{11})$

Forward Path 4: $s_0 s_1 s_3 s_6 s_4 s_0$ $(F_4 = X^6)$

Forward Path 5: $s_0 s_1 s_2 s_3 s_5 s_7 s_6 s_4 s_0$ $(F_5 = X^8)$

Forward Path 6: $s_0 s_1 s_2 s_5 s_3 s_6 s_4 s_0$ $(F_6 = X^7)$

Forward Path 7: $s_0 s_1 s_2 s_4 s_0$ $(F_7 = X^7)$

There are 7 forward path

Forward path 1 and 5 touch all states

$$\Delta_1 = \Delta_5 = 1$$

The subgraph not touching forward path 3 and 6 shown in (a)

$$\Delta_3 = \Delta_6 = 1 - X$$

The subgraph not touching forward path 2 shown in (b)

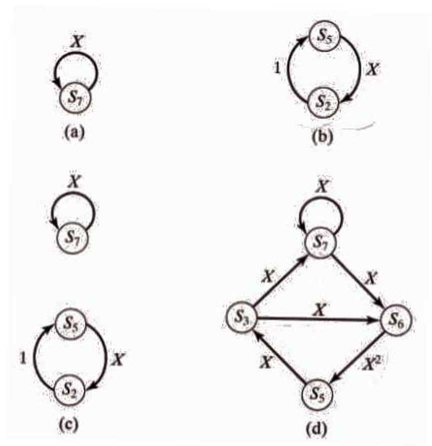
$$\Delta_2 = 1 - X$$

The subgraph not touching forward path 4 shown in (c)

$$\Delta_4 = 1 - (X + X) + (X^2) = 1 - 2X + X^2$$

The subgraph not touching forward path 7 shown in (d)

$$\Delta_7 = 1 - (X + X^4 + X^5) + (X^5) = 1 - X - X^4$$



$$
\begin{aligned}
A(X) &= \frac{\sum_i F_i \Delta_i}{\Delta} \\
&= \frac{X^6 + X^7 - X^8}{1 - 2X - X^3} \\
&= X^6 + 3X^7 + 5X^8 + 11X^9 + 25X^{10} + \ldots
\end{aligned}
$$

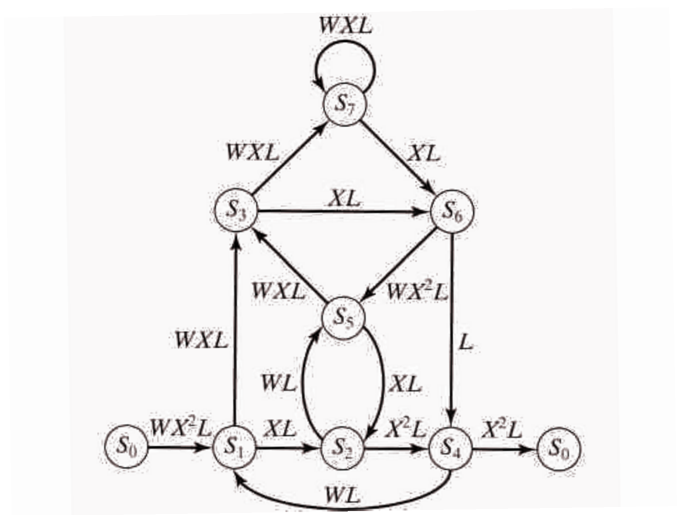
The codeword WEF A(X) provides a complete description of the weight distribution of all nonzero codewords

In this case there is one such codeword of weight 6, three of weight 7, five of weight 8, and so on

Input-Output Weight Enumerating Function (IOWEF)

If the modified state diagram is augmented by labeling each branch corresponding to a nonzero information block with W^w,where w is the weight of k information bits on that branch, and labeling each branch with L, then the codeword (IOWEF) is

$$A(W, X, L) = \sum_{w,d,l} A_{w,d,l} W^w X^d L^l$$



$$
\begin{aligned}
A(W, X, L) &= \frac{X^6W^2L^5 + X^7WL^4 - X^8W^2L^5}{1 - XW(L + L^2) - X^2W^2(L^4 - L^3) - X^3WL^3 - X^4W^2(L^3 - L^4)} \\
&= X^6W^2L^5 + X^7(WL^4 + W^3L^6 + W^3L^7) + X^8(W^2L^6 + W^4L^7 + W^4L^8 + 2W^4L^9
\end{aligned}
$$

This implies that the codeword of weight 6 has length-5 branches and an information weight of 2,one codeword of weight 7 has length-4 branches and information weight 1, another has length-6 branches and information weight 3, the third has length-7 branches and information weight 3, and so on

Termination, tailbiting, and puncturing

Punctured Convolutional Codes

1. Punctured convolutional codes are derived from the mother code by periodically deleting certain code bits in the code sequence of the mother code:

$$(n_m, k_m, m_m) \underrightarrow{p} (n_p, k_p, m_p)$$

where P is an $n_m \times k_p/k_m$ puncturing matrix with elements $p_{ij} \in \{0,1\}$

2. A 0 in P means the deletion of a bit and a 1 means that this position is remained the same in the puncture sequence
3. Let $p = n_m k_p/k_m$ be the puncture period (the number of all element in P)and $w \leq p$ be the number of 1 elements in P

It is clear that $w = n_p$. The rate of the puncture code is now

$$R_p = \frac{k_p}{n_p} = \frac{k_m p}{n_m}\frac{1}{w} = \frac{p}{w}R_m$$

4. Since $w \leq p$, we have $R_m \leq R_p$
5. w can not be too small in order to assure that $R_p \leq 1$

Example of the Punctured Convolutional Code

1. Consider the $(2,1,2)$ code with the generator matrix

$$G = \begin{pmatrix} 11 & 10 & 11 & & \\ & 11 & 10 & 11 & \\ & & \ddots & & \ddots \end{pmatrix}$$

2. Let

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

be the punctured matrix

3. The code rate of the punctured code is $\frac{k_p}{n_p} = \frac{6}{7}$

4. If lth code bit is removed then the respective column of the generator matrix of the mother code must be deleted

5.

11	01	01	01	01	10	11	01	01	01	01	10	11	01
11	10	11											
	11	10	11										
		11	10	11									
			11	10	11								
				11	10	11							
					11	10	11						
						11	10	11					
							11	10	11				
								11	10	11			
									11	10	11		
										11	10	11	
											11	10	11

6. We can obtain the generator matrix of the punctured code by deleting those un-bold columns in the above matrix

$$G_p = \left(\begin{array}{ccccccc|cccccc}
1 & 1 & 0 & 1 & & & & & & & & & \\
 & & 1 & 0 & 1 & & & & & & & & \\
 & & & 1 & 0 & 1 & & & & & & & \\
 & & & & 1 & 0 & 1 & & & & & & \\
 & & & & & 1 & 1 & 1 & 1 & & & & \\
 & & & & & & 1 & 1 & 0 & 1 & & & \\
\hline
 & & & & & & & 1 & 1 & 0 & 1 & & \\
 & & & & & & & & & 1 & 0 & 1 & \\
 & & & & & & & & & & 1 & 0 & 1 \\
 & & & & & & & & & & & \ddots & \ddots
\end{array} \right)$$

7. The above punctured code has $m_p = 2$
8. Punctured convolutional codes are very important in partical applications, and are used in many areas

Optimal decoding: Viterbi and BCJR decoding

- We will discuss three decoding algorithms:
  1. Viterbi decoding: 1967 by Viterbi
  2. SOVA decoding: 1989 by Hagenauer
  3. BCJR decoding: 1974 by Bahl etc.
- These algorithms are operated on the trellis of the codes and the complexity depends on the number of states in the trellis.
- We can apply these algorithms on block and convolutional codes since the former has the regular trellis and the latter has the irregular trellis.
- Viterbi and SOVA decoding minimize the codeword error probability and BCJR minimize the information bit error probability.

The Viterbi algorithm

- Assume that an information sequence $u = (u_0, u_1, \ldots, u_{h-1})$ of the length $K^* = kh$ is encoded.
- Then a codeword $v = (v_0, v_1, \ldots, v_{h+m-1})$ of length $N = n(h+m)$ is generated after the convolutional encoder.
- Thus, with zero-biting of mk zero bits, we have a $[n(h+m), kh]$ linear block code.
- After the discrete memoryless channel (DMC), a sequence $r = (r_0, r_1, \ldots, r_{h+m-1})$ is received. We will focus on a BSC, a binary-input/Q-ary output, and AWGN channels.
- A (ML) decoder for a DMC chooses $\hat{v}$ as the codeword v that maximizes the log-likelihood function $\log p(r|v)$.

- Since the channel is memoryless, we have

$$p(\mathbf{r}|\mathbf{v}) = \prod_{l=0}^{h+m-1} p(\mathbf{r}_l|\mathbf{v}_l) = \prod_{l=0}^{N-1} p(r_l|v_l)$$

$$\log p(\mathbf{r}|\mathbf{v}) = \sum_{l=0}^{h+m-1} \log p(\mathbf{r}_l|\mathbf{v}_l) = \sum_{l=0}^{N-1} \log p(r_l|v_l)$$

- We define the bit metric, branch metric, and path metric as the log-likelihood function of the corresponding conditional probability functions as follows:

$$M(r_l|v_l) = \log p(r_l|v_l),$$

$$M(\mathbf{r}_l|\mathbf{v}_l) = \log p(\mathbf{r}_l|\mathbf{v}_l),$$

$$M(\mathbf{r}|\mathbf{v}) = \log p(\mathbf{r}|\mathbf{v}).$$

- Then we can write the path metric $M(\mathbf{r}|\mathbf{v})$ as the sum of the branch metrics

$$M(\mathbf{r}|\mathbf{v}) = \sum_{l=0}^{h+m-1} M(\mathbf{r}_l|\mathbf{v}_l) = \sum_{l=0}^{h+m-1} \log p(\mathbf{r}_l|\mathbf{v}_l)$$

or the sum of the bit metrics

$$M(\mathbf{r}|\mathbf{v}) = \sum_{l=0}^{N-1} M(r_l|v_l) = \sum_{l=0}^{N-1} \log p(r_l|v_l)$$

- Similarly, a partial metric for the first t branch of a path can be written as

$$M([\mathbf{r}|\mathbf{v}]_t) = \sum_{l=0}^{t-1} M(\mathbf{r}_l|\mathbf{v}_l) = \sum_{l=0}^{nt-1} M(r_l|v_l)$$

- The basic operations of the Viterbi algorithm is the addition, comparison, and selection (ACS).
- At each time unit t, considering each state s_t and 2^k previous state s_{t-1} connecting to s_t, we compute the (optimal) partial metric of s_t by adding the branch metric connecting between s_t and s_{t-1} to the partial metric of s_{t-1} and selecting the largest metric.
- We store the optimum path (survivor) with the partial metric for each state s_t at time t.
- In other words, we have to store 2^ν survivor pathes along with its optimal partial metric from time m to time h.

- The above algorithm, when applied to the received sequence r from a DMC, find the path through the trellis with the largest metric, that is the maximum likelihood path (codeword).
- The final survivor $\hat{\mathbf{v}}$ in the Viterbi algorithm is the maximum likelihood path; that is

$$M(\mathbf{r}|\hat{\mathbf{v}}) \geq M(\mathbf{r}|\mathbf{v}), \text{ all } \mathbf{v} \neq \hat{\mathbf{v}}$$

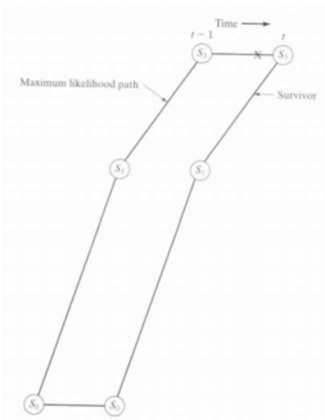


Figure 15: Elimination of the maximum likelihood path contradicts to the definition of optimal metric

- Consider a $(3, 1, 2)$ convolutional code with

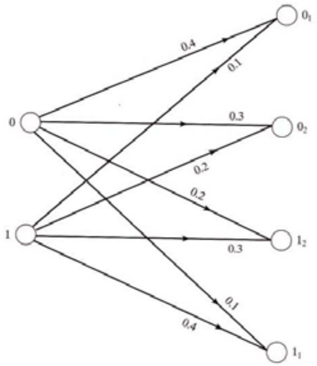
$$G(D) = [1 + D, 1 + D^2, 1 + D + D^2]$$

with $m = 2$ and $h = 5$.

- With zero-biting, we have a $[3(5 + 2), 5] = [21, 5]$ linear block code.
- The first $m = 2$ unit and the last $m = 2$ unit of the trellis correspond to the departure of the initial state s_0 and the return of the final state s_0.
- The central units correspond to the replica of the state diagram with $2^k = 2$ branches leaving and entering each state.
- Each branch is labelled with u_i/v_i of k-bit input u_i and n-bit output v_i.

Viterbi Algorithm for a Binary-input, Quaternary-Output DMC

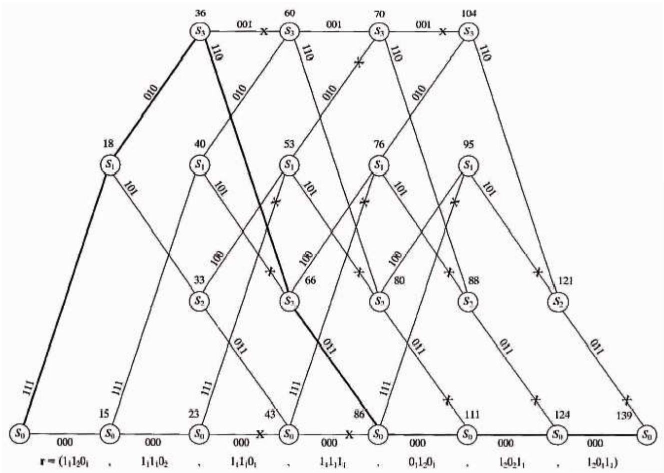
Consider the binary-input, quaternary-output (Q=4) DMC



$v_l \setminus r_l$	0_1	0_2	1_2	1_1
0	−0.4	−0.52	−0.7	−1.0
1	−1.0	−0.7	−0.52	−0.4

$v_l \setminus r_l$	0_1	0_2	1_2	1_1
0	10	8	5	0
1	0	5	8	10

Metric table for the binary/Quaternary channel



The quaternary received sequence is

$$r = (1_1 1_2 0_1, 1_1 1_1 0_2, 1_1 1_1 0_1, 1_1 1_1 1_1, 0_1 1_2 0_1, 1_2 0_2 1_1, 1_2 0_1 1_1)$$

The final survivor is

$$\hat{v} = (111, 010, 110, 011, 000, 000, 000)$$

The decoded information sequence is

$$\hat{u} = (1, 1, 0, 0, 0)$$

Viterbi Algorithm for a BSC

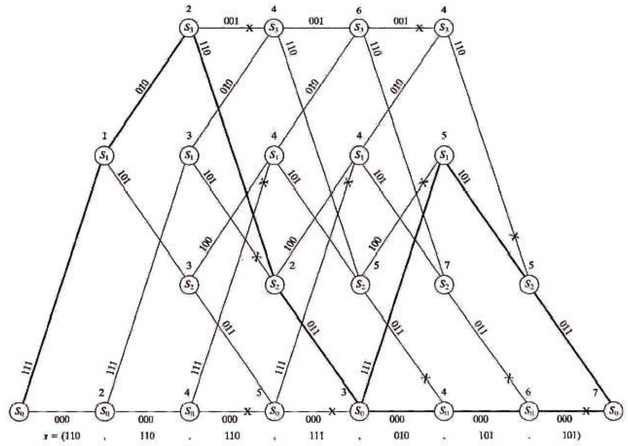
- In BSC with transition probability $p < 1/2$, the received sequence r is binary and the log-likelihood function becomes

$$\log p(\mathbf{r}|\mathbf{v}) = d(\mathbf{r},\mathbf{v}) \log \frac{p}{1-p} + N \log(1-p),$$

where $d(r,v)$ is the Hamming distance between r and v.

- Because $log\ \frac{p}{1-p} < 0$ and $N \log(1-p)$ is a constant for all v, an MLD for a BSC chooses v as the codeword $\hat{v}$ that minimizes the Hamming distance

$$d(\mathbf{r},\mathbf{v}) = \sum_{l=0}^{h+m-1} d(\mathbf{r}_l,\mathbf{v}_l) = \sum_{l=0}^{N-1} d(r_l, v_l)$$



The received sequence is

$$r = (110, 110, 110, 111, 010, 101, 101)$$

The final survivor is

$$\hat{v} = (111, 010, 110, 011, 111, 101, 011)$$

The decoded information sequence is

$$\hat{u} = (1, 1, 0, 0, 1)$$

That final survivor has a metric of 7 means that no other path through the trellis differs from r in fewer than seven positions.

- In the BSC case, maximizing the log-likelihood function is equivalent to finding the codeword v that is closest to the received sequence r in Hamming distance.
- In the AWGN case, maximizing the log-likelihood function is equivalent to finding the codeword v that is closest to the received sequence r in Euclidean distance.

Viterbi Algorithm for an AWGN

- Consider the AWGN channel with binary input, i.e.,

$$v = (v_0, v_1, \cdots, v_{N-1}),$$

where $v_i \in \{1, -1\}$.

- The bit metric $M(r_l|v_l)$ for an AWGN with binary input of unit energy and PSD $N_0/(2E_s)$ is

$$p(r_l|v_l) = \sqrt{\frac{E_s}{\pi N_0}} e^{-\frac{E_s}{N_0}(r_l - v_l)^2}$$

- The path metric can be simplified as

$$\begin{aligned} M(r|v) &= \log p(r|v) \\ &= -\frac{E_s}{N_0}\sum_{l=0}^{N-1}(r_l - v_l)^2 + \frac{N}{2}\log\frac{E_s}{\pi N_0}. \end{aligned}$$

- A codeword v minimize the Euclidean distance $\sum_{l=0}^{N-1}(r_l - v_l)^2$ also maximize the log-likelihood function $\log p(r|v)$.
- We can define a new path metric of v as

$$M(r|v) = \sum_{l=0}^{N-1}(r_l - v_l)^2$$

and the Viterbi algorithm finds the optimal path v with minimum Euclidean distance from r.

- The path metric can also be simplified as

$$\begin{aligned} M(r|v) &= \log p(r|v) \\ &= -\frac{E_s}{N_0}\sum_{l=0}^{N-1}(r_l^2 - 2r_l v_l + v_l^2) + \frac{N}{2}\log\frac{E_s}{\pi N_0} \\ &= \frac{2E_s}{N_0}\sum_{l=0}^{N-1} r_l \cdot v_l - \frac{E_s}{N_0}(|r|^2 + N) + \frac{N}{2}\log\frac{E_s}{\pi N_0} \end{aligned}$$

- A codeword v maximize the correlation $r \cdot v$ also maximize the log-likelihood function $\log p(r|v)$.
- We can define a new path metric of v as $M(r|v) = \sum_{l=0}^{N-1} r_l \cdot v_l$ and the Viterbi algorithm finds the optimal path v with maximum correlation from r.

The Soft-Output Viterbi Algorithm

- In a series or parallel concatenated decoding system, usually one decoder passes reliability information about its decoded outputs to the other decoder, which refer to the soft in/soft out decoding.
- The combination of the hard-decision output and the reliability indicator is called a soft output.

- At time unit $l = t + 1$, the partial path metric that must be maximized by the Viterbi algorithm for a binary-input, continous-output AWGN channel given the partial received sequence $[r]_{t+1} = (r_0, r_1, \ldots, r_t)$ is given as

$$M([\mathbf{r}|\mathbf{v}]_{t+1}) = ln\{p([\mathbf{r}|\mathbf{v}]_{t+1})p([\mathbf{v}]_{t+1})\}$$

- A priori probability $p([v]_{t+1})$ is included since these will not be equally likely when a priori probability of the information bits are not equally likely

$$\begin{aligned} M([\mathbf{r}|\mathbf{v}]_{t+1}) &= \log\{[\prod_{l=0}^{t-1} p(\mathbf{r}_l|\mathbf{v}_l)p(u_l)]p(\mathbf{r}_t|\mathbf{v}_t)p(u_t)\} \\ &= \log\{[\prod_{l=0}^{t-1} p(\mathbf{r}_l|\mathbf{v}_l)\} + \{\sum_{j=0}^{n-1} \log[p(r_t^{(j)}|v_t^{(j)})] + \log[p(u_t)]\} \end{aligned}$$

By multiplying each term in the second sum by 2 and introducing constants $C_r^{(j)}$ and C_u as follows:

$$\{\sum_{j=0}^{n-1}[2\log[p(r_t^{(j)}|v_t^{(j)})] - C_r^{(j)}] + [2\log[p(u_t)] - C_u]\}$$

where the constants

$$C_r^{(j)} \equiv \log[p(r_t^{(j)}|v_t^{(j)} = +1)] + \log[p(r_t^{(j)}|v_t^{(j)} = -1)], j = 0, 1, \ldots, n-1$$

$$C_u \equiv \log[p(u_t = +1)] + \log[p(u_t = -1)]$$

Similarly, we modify the first sum by the same way and obtain the modified partial path metric $M^*([\mathbf{r}|\mathbf{v}]_t)$ as

$$\begin{aligned} M^*([\mathbf{r}|\mathbf{v}]_{t+1}) &= M^*([\mathbf{r}|\mathbf{v}]_t) + \sum_{j=0}^{n-1}\{2\log[p(r_t^{(j)}|v_t^{(j)})] - C_r^{(j)}\} \\ &\quad +\{2\log[p(u_t)] - C_u\} \\ &= M^*([\mathbf{r}|\mathbf{v}]_{t-1}) + \sum_{j=0}^{n-1} v_t^{(j)} \log[\frac{p(r_t^{(j)}|v_t^{(j)} = +1)}{p(r_t^{(j)}|v_t^{(j)} = -1)}] \\ &\quad +u_t \log[\frac{p(u_t = +1)}{p(u_t = -1)}] \end{aligned}$$

- The log-likelihood ratio, or L-value, of a received symbol r at the output with binary inputs $v = \pm 1$ is defined as

$$L(r) = \log[\frac{p(r|v = +1)}{p(r|v = -1)}]$$

- The L-value of an information bit u is define as

$$L(u) = \log[\frac{p(u = +1)}{p(u = -1)}]$$

- A large positive $L(r)$ indicates a high reliability that $v = 1$ and a large negative $L(r)$ indicates a high reliability that $v = -1$.

- Since he bit metric $M(r_l|v_l)$ for an AWGN with binary input of unit energy and PSD $N_0/(2E_s)$ (SNR=$1/(N_0/E_s) = E_s/N_0$) is

$$p(r_l|v_l) = \sqrt{\frac{E_s}{\pi N_0}} e^{-\frac{E_s}{N_0}(r_l - v_l)^2},$$

the L-value of r is thus

$$L(r) = \log[\frac{p(r|v=+1)}{p(r|v=-1)}] = (4E_s/N_0)r$$

- Defining $L_C \equiv 4E_s/N_0$ as the channel reliability factor, the modified metric for SOVA decoding can be written by

$$M^*([\mathbf{r}|\mathbf{v}]_{t+1}) = M^*([\mathbf{r}|\mathbf{v}]_t) + \sum_{j=0}^{n-1} L_c v_t^{(j)} r_t^{(j)} + u_t L(u_t)$$

- Assume that a comparison is being made at state

$$s_i, i = 0, 1, \ldots, 2^\nu - 1,$$

between the maximum likelihood (ML) path $[v]_t$ and an incorrect path $[v']_t$ at time $l = t$

- Define the metric difference between $[v]_t$ and $[v']_t$ as

$$\Delta_{t-1}(S_i) = \frac{1}{2}\{M^*([\mathbf{r}|\mathbf{v}]_t) - M^*([\mathbf{r}|\mathbf{v}']_t)\}$$

- The probability $P(C)$ that the ML path is correctly seleted at time t is given by

$$P(C) = \frac{p([\mathbf{v}|\mathbf{r}]_t)}{p([\mathbf{v}|\mathbf{r}]_t) + p([\mathbf{v}'|\mathbf{r}]_t)}$$

Since

$$p([\mathbf{v}|\mathbf{r}]_t) = \frac{p([\mathbf{r}|\mathbf{v}]_t)p([\mathbf{v}]_t)}{p(r)} = \frac{e^{M([\mathbf{r}|\mathbf{v}]_t)}}{p(r)}$$

and

$$p([\mathbf{v}'|\mathbf{r}]_t) = \frac{p([\mathbf{r}|\mathbf{v}']_t)p([\mathbf{v}']_t)}{p(r)} = \frac{e^{M([\mathbf{r}|\mathbf{v}']_t)}}{p(r)}$$

$$P(C) = \frac{e^{\Delta_{t-1}(S_i)}}{1 + e^{\Delta_{t-1}(S_i)}}$$

Finally,the log-likelihood ratio is

$$\log\{\frac{P(C)}{[1 - P(C)]}\} = \Delta_{t-1}(S_i)$$

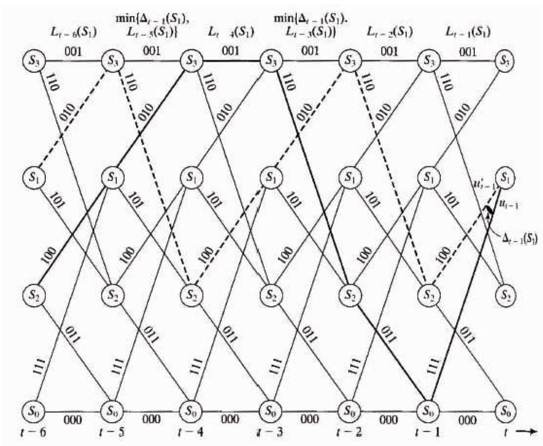
A reliability vector at time $l = m + 1$ for state S_i as follows:

$$L_{m+1}(S_i) = [L_0(S_i), L_1(S_i), \ldots, L_m(S_i)]$$

$$L_l(S_i) \equiv \begin{cases} \Delta_m(S_i) & , u_l \neq u'_l \\ \infty & , u_l \neq u'_l \end{cases}$$

$$l = 0, 1, \ldots, m$$

the reliability of a bit position is either∞, if it is not affected by the path decision



- The above SOVA decoding is one-way version since we have to store the optimal survivor, the optimal partial metric, and reliability vector for each state s_t in time unit t.
- We can have two-way version for Viterbi and SOVA algorithm.
- In two-way version, we have to do forward recursion and backward recursion simultaneously and store the optimal forward metric and optimal backward metric, but we do not have to store the optimal path.
- We make the final decision on each information bit by adding the optimal forward metric, optimal backward metric, and the branch metric.

Step 1: Forward recursion

- step 1-1:
  - $i = 0$;
  - partial path metric $\mu_f(s_0^0) = 0$; $\mu_f(s_1^0) = \mu_f(s_2^0) = \mu_f(s_3^0) = \infty$
- step 1-2:
  - $i = i + 1$
  - for $l = 0, 1, 2, 3$

$$\begin{aligned}\mu_f(s_l^i) \quad = \quad \min\{&\mu_f(s_0^{i-1}) + Branch(s_0^{i-1}, s_l^i),\\ &\mu_f(s_1^{i-1}) + Branch(s_1^{i-1}, s_l^i),\\ &\mu_f(s_2^{i-1}) + Branch(s_2^{i-1}, s_l^i),\\ &\mu_f(S_3^{i-1}) + Branch(s_3^{i-1}, s_l^i)\}\end{aligned}$$

  where $Branch(\cdot, \cdot)$ is the branch metric.
- step 1-3: Go to step 2 until $i = h + m$.

Step 2: Backward recursion

- step 2-1:
  - $i = h + m$;
  - partial path metric $\mu_{h+m}(s_0^\tau) = 0$; $\mu_{h+m}(s_1^\tau) = \mu_{h+m}(s_2^\tau) = \mu_{h+m}(s_3^\tau) = \infty$
- step 2-2:
  - $i = i - 1$
  - for $l = 0, 1, 2, 3$

$$\begin{aligned}\mu_\tau(s_l^i) \quad = \quad min\{&\mu_\tau(s_0^{i+1}) + Branch(s_0^{i+1}, s_l^i)\}\\ &\mu_\tau(s_1^{i+1}) + Branch(s_1^{i+1}, s_l^i),\\ &\mu_\tau(s_2^{i+1}) + Branch(s_2^{i+1}, s_l^i),\\ &\mu_\tau(s_3^{i+1}) + Branch(s_3^{i+1}, s_l^i)\}\end{aligned}$$

  where $Branch(\cdot, \cdot)$ is the branch metric.
- step 2-3: Go to step 2 until $i = 0$.

Step 3: Soft decision

- step 3-1: $i = 0$
- step 3-2:
  - $i = i + 1$
  - $$\lambda_i^0 = min_{s,\bar{s}}\{\mu_f(S_s^{i-1}) + Branch_0(S_s^{i-1}, S_{\bar{s}}^i) + \mu_\gamma(S_{\bar{s}}^i)\}$$
    where $Branch_0(S_s^{i-1}, S_{\bar{s}}^i)$ is the branch metric from node S_s^{i-1} to $S_{\bar{s}}^i$ which corresponds to $c_i = 0$ (if no such branch exists, the branch metric is assigned infinity)
  - $$\lambda_i^1 = min_{s,\bar{s}}\{\mu_f(S_s^{i-1}) + Branch_1(S_s^{i-1}, S_{\bar{s}}^i) + \mu_\gamma(S_{\bar{s}}^i)\}$$
    where $Branch_1(S_s^{i-1}, S_{\bar{s}}^i)$is the branch metric from node S_s^{i-1} to $S_{\bar{s}}^i$ which corresponds to $c_i = 1$
- step 3-3: $\Lambda(i) = \lambda_i^0 - \lambda_i^1$

The complexity of SOVA

- We can modify the SOVA algorithm to reduce the decoding complexity.
- The computational complexity of the SOVA is upper-bounded by twive of the VA.
- In fact, the computational complexity of the SOVA is about 1.5 times of the VA.

The BCJR Algorithm

- In 1974 Bahl, Cocke, Jelinek, and Ravi introduced a MAP decoder, called the BCJR algorithm
- The computation complexity of the BCJR algorithm is greater than the Viterbi algorithm
- Viterbi decoding is preferred in the case of equally likely information bits.When the information bits are not equally, however better performance is achieved with MAP decoding.

We describe the BCJR algorithm for the case of rate $R = 1/n$ convolutional codes used on a binary-input, continous-output AWGN channel and on a DMC.

Our presentation is based on the log-likelihood ratios, or L-values. The decoder input are the received sequence r and a priori L-value of the information bits $L_a(u_l)$, $l = 0, 1, \ldots, h-1$.

As in the case of the SOVA, we do not assume that the information bits are equally likely. The algorithm calculates the a posteriori L-values

$$L_(u_l) \equiv \log[\frac{P(u_l=+1|\mathbf{r})}{P(u_l=-1|\mathbf{r})}]$$

called the *APP* L-values, of each information bits, and the decoder output is given by

$$\hat{u}_l = \begin{cases} +1 \quad if\ L(u_l) > 0 \\ \\ -1 \quad if\ L(u_l) < 0 \end{cases} , l = 0, 1, \ldots, h-1$$

In iterative decoding, the APP L-values can be taken as the decoder outputs, resulting in a SISO decoding algorithm.

We begin our development of the BCJR algorithm by rewriting the APP value $P(u_l=+1|\mathbf{r})$ as follows:

$$P(u_l=+1|\mathbf{r}) = \frac{p(u_l=+1,\mathbf{r})}{P(\mathbf{r})} = \frac{\sum_{\mathbf{u}\in\mathbf{U}_l^+} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}{\sum_{\mathbf{u}} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}$$

Where $\mathbf{U}_l^+$ is the set of all information sequence $\mathbf{u}$ such that $u_l=+1$, $\mathbf{v}$ is the transmitted codeword corresponding to the information sequence $\mathbf{u}$, and $p(r|v)$ is the pdf of the received sequence $\mathbf{r}$ given $\mathbf{v}$

Rewriting $P(u_l=-1|\mathbf{r})$ in the same way, we can write the expression for the APP L-value as

$$L(u_l) = \ln\left[\frac{\sum_{\mathbf{u}\in\mathbf{U}_l^+} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}{\sum_{\mathbf{u}\in\mathbf{U}_l^-} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}\right]$$

$\mathbf{U}_l^-$ is the set of all information sequence $\mathbf{u}$ such that $u_l=-1$

First, making use of the trellis structure of the code, we can reformulate $p(u_l=+1|\mathbf{r})$ as follow:

$$P(u_l=+1|\mathbf{r}) = \frac{p(u_l=+1,\mathbf{r})}{P(\mathbf{r})} = \frac{\sum_{(s,s')\in\Sigma_l^+} p(s_l=s', s_{l+1}=s,\mathbf{r})}{P(\mathbf{r})}$$

where Σ_l^+ is the set of all state pairs $s_l=s'$ and $s_{l+1}=s$ that correspond to the input bit $u_l=+1$ at time l. Reformulating the expression $p(u_l=-1|\mathbf{r})$ in the same way, we can now write the APP L-value as

$$L(u_l) = \ln\left\{\frac{\sum_{(s,s')\in\Sigma_l^+} p(s_l=s', s_{l+1}=s,\mathbf{r})}{\sum_{(s,s')\in\Sigma_l^-} p(s_l=s', s_{l+1}=s,\mathbf{r})}\right\}$$

where Σ_l^- is the set of all state pairs $s_l=s'$ and $s_{l+1}=s$ that correspond to the input bit $u_l=-1$ at time l.

The joint pdf's $p(s', s, \mathbf{r})$ can be evaluated recursively.

$$p(s', s, \mathbf{r}) = p(s', s, \mathbf{r}_{t<l}, \mathbf{r}_l, \mathbf{r}_{t>l})$$

where $\mathbf{r}_{t<l}$ represents the portion of the received sequence $\mathbf{r}$ before time l, and $\mathbf{r}_{t>l}$ represents the portion of the received sequence $\mathbf{r}$ after time l. Application of Bayes' rule yields

$$\begin{aligned} \Rightarrow p(s', s, \mathbf{r}) &= p(\mathbf{r}_{t>l}|s', s, \mathbf{r}_{t<l}, \mathbf{r}_l)p(s', s, \mathbf{r}_{t<l}, \mathbf{r}_l) \\ &= p(\mathbf{r}_{t>l}|s', s, r_{t<l}, r_l)p(s, \mathbf{r}_l|s', \mathbf{r}_{t<l})p(s', \mathbf{r}_{t<l}) \\ &= p(\mathbf{r}_{t>l}|s)P(s, \mathbf{r}_l|s')p(s', \mathbf{r}_{t<l}) \end{aligned}$$

where the last equality follows from the fact that probability of the received branch at time l depend only on the state and input bit at time l.
Define :

$$\alpha_l(s') \equiv p(s', \mathbf{r}_{t<l})$$

$$\gamma_l(s', s) \equiv p(s, \mathbf{r}_l|s')$$

$$\beta_{l+1}(s) \equiv p(\mathbf{r}_{t>l}|s)$$

$$P(s', s, \mathbf{r}) = \beta_{l+1}(s)\gamma_l(s', s)\alpha_l(s')$$

Forward recursion:

$$\begin{aligned} \alpha_{l+1}(s) &= p(s, r_{t<l+1}) = \sum_{s' \in \sigma_l} p(s', s, r_{t<l+1}) \\ &= \sum_{s' \in \sigma_l} p(s, r_l|s', r_{t<l})p(s', r_{t<l}) \\ &= \sum_{s' \in \sigma_l} p(s, r_l|s')p(s', r_{t<l}) \\ &= \sum_{s' \in \sigma_l} \gamma_l(s', s)\alpha_l(s') \end{aligned}$$

Backward recursion:

$$\begin{aligned} \beta_l(s') &= P(r_{t>l-1}|s') = \sum_{s \in \sigma_{l+1}} P(r_{t>l-1}, s|s') = \sum_{s \in \sigma_{l+1}} \frac{P(r_l, s, s')}{P(s')} \\ &= \sum_{s \in \sigma_{l+1}} \frac{P(r_l, s, s', r_{t>l})}{P(s')} = \sum_{s \in \sigma_{l+1}} \frac{P(r_{t>l}|s, s', r_l)P(s, s', r_l)}{P(s')} \\ &= \sum_{s \in \sigma_{l+1}} \frac{P(r_{t>l}|s)P(s, s', r_l)}{P(s')} = \sum_{s \in \sigma_{l+1}} \frac{P(r_{t>l}|s)P(s, s', r_l)}{P(s')} \\ &= \sum_{s \in \sigma_{l+1}} \frac{\beta_{l+1}(s)P(s, r_l|s')P(s')}{P(s')} = \sum_{s \in \sigma_{l+1}} \beta_{l+1}(s)\gamma_l(s, s') \end{aligned}$$

We can write the branch matric $\gamma_l(s', s)$ as

$$\begin{aligned}\gamma_l(s,s') &= P(s,\mathbf{r}_l|s') = \frac{p(s',s,\mathbf{r}_l)}{p(s')} \\ &= \left[\ \frac{p(s,s')}{p(s')}\ \right]\left[\ \frac{p(s',s,\mathbf{r}_l)}{p(s,s')}\ \right] \\ &= P(s|s')P(\mathbf{r}_l|s',s) = P(u_l)P(\mathbf{r}_l|\mathbf{v}_l)\end{aligned}$$

For a continous-output AWGN channel, if $s' \to s$ is a valid state transition,

$$\gamma_l(s's) = P(u_l)p(\mathbf{r}_l|\mathbf{v}_l) = P(u_l)\left(\ \sqrt{\frac{E_s}{\pi N_0}}\ \right)^n e^{-\frac{E_s}{N_0}\|\mathbf{r}_l-\mathbf{v}_l\|^2}$$

where $\| \mathbf{r}_l - \mathbf{v}_l \|^2$ is the squared Eucliden distance between the received branch $\mathbf{r}_l$ and the transmitted branch $\mathbf{v}_l$ at time l

The constant term $\left(\ \sqrt{\frac{E_s}{\pi N_0}}\ \right)^n$ always appears raised to the power h in the expression for the pdf $p(s', s, \mathbf{r})$. Thus, $\left(\ \sqrt{\frac{E_s}{\pi N_0}}\ \right)^{nh}$ will be a factor of every term in the numerator and denominator summations of $L(u_l)$, and its effect will cancel. The result in the modified branch metric:

$$\gamma_l(s's) = P(u_l)e^{-\frac{E_s}{N_0}\|\mathbf{r}_l-\mathbf{v}_l\|^2}$$

We expression a priori probabilities $p(u_l = \pm 1)$ as exponential term by writing:

$$\begin{aligned}p(u_l=\pm 1) &= \frac{[p(u_l=+1)/p(u_l=-1)]^{\pm 1}}{1+[p(u_l=+1)/p(u_l=-1)]^{\pm 1}} \\ &= \frac{e^{\pm L_a(u_l)}}{1+e^{\pm L_a(u_l)}} \\ &= \frac{e^{-L_a(u_l)/2}}{1+e^{-L_a(u_l)}}e^{u_l L_a(u_l)/2} \\ &= A_l e^{u_l L_a(u_l)/2}\end{aligned}$$

then

$$\begin{aligned}\gamma_l(s,s') &= A_l e^{u_l L_a(u_l)/2} e^{-(E_s/N_0)\|\mathbf{r}_l-\mathbf{v}_l\|^2} \\ &= A_l e^{u_l L_a(u_l)/2} e^{(2E_s/N_0)(\mathbf{r}_l\cdot\mathbf{v}_l)-\|\mathbf{r}_l\|^2-\|\mathbf{v}_l\|^2} \\ &= A_l e^{-(\|\mathbf{r}_l\|^2+n)} e^{u_l L_a(u_l)/2} e^{(L_c/2)(\mathbf{r}_l\cdot\mathbf{v}_l)} \\ &= A_l B_l e^{u_l L_a(u_l)/2} e^{(L_c/2)(\mathbf{r}_l\cdot\mathbf{v}_l)}, l=0,1,\ldots,h-1 \\ \gamma_l(s',s) &= P(u_l)e^{-(E_s/N_0)\|\mathbf{r}_l-\mathbf{v}_l\|^2} \\ &= e^{-(E_s/N_0)\|\mathbf{r}_l-\mathbf{v}_l\|^2} \\ &= B_l e^{(L_c/2)(\mathbf{r}_l\cdot\mathbf{v}_l)}, l=h,h+1,\ldots,K-1\end{aligned}$$

where $B_l \equiv \| \mathbf{r}_l \|^2 + n$ is a constant independent of the codeword $\mathbf{v}_l$, and $L_c = 4E_S/N_0$ is the channel reliability factor.

Using the log-domain metric:

$$\gamma_l^*(s', s) \equiv \ln \gamma_l(s', s) = \begin{cases} \frac{u_l L_a(u_l)}{2} + \frac{L_c}{2}\mathbf{r}_l \cdot \mathbf{v}_l & l = 0, 1, \ldots, h-1 \\ \frac{L_c}{2}\mathbf{r}_l \cdot \mathbf{v}_l & l = h, h+1, \ldots, K-1 \end{cases}$$

We define a max calculate function:

$$max^*(x, y) = \ln(e^x + e^y) = max(x, y) + \ln(1 + e^{-|x-y|})$$

$$\begin{aligned} \alpha_{l+1}^*(s) \equiv \ln \alpha_{l+1}(s) &= \ln \sum_{s' \in \sigma_l} \gamma_l(s', s)\alpha_l(s') \\ &= \ln \sum_{s' \in \sigma_l} e^{[\gamma_l^*(s', s) + \alpha_l^*(s')]} \\ &= max^*_{s' \in \sigma_l}[\gamma_l^*(s', s) + \alpha_l^*(s')] \end{aligned}$$

$$l = 0, 1, \ldots, K-1$$

$$\alpha_0^*(s) \equiv \ln \alpha_0(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}$$

$$\begin{aligned} \beta_l^*(s) \equiv \ln \beta_l(s') &= \ln \sum_{s \in \sigma_{l+1}} \gamma_l(s', s)\beta_{l+1}(s) \\ &= \ln \sum_{s \in \sigma_{l+1}} e^{[\gamma_l^*(s', s) + \beta_{l+1}^*(s)]} \\ &= max^*_{s' \in \sigma_{l+1}}[\gamma_l^*(s', s) + \beta_{l+1}^*(s)] \end{aligned}$$

$$l = K-1, K-2, \ldots, 0$$

$$\beta_K^*(s) \equiv \ln \beta_K(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases}$$

Termination using MAX-LOG

$$P(s', s, \mathbf{r}) = e^{\beta^*_{l+1}(s) + \gamma^*_l(s',s) + \alpha^*_l(s')}$$

and

$$L(u_l) = \ln \left\{ \sum_{(s',s)\in\Sigma_l^+} e^{\beta^*_{l+1}(s)+\gamma^*_l(s',s)+\alpha^*_l(s')} \right\}$$

$$- \ln \left\{ \sum_{(s',s)\in\Sigma_l^-} e^{\beta^*_{l+1}(s)+\gamma^*_l(s',s)+\alpha^*_l(s')} \right\}$$

Log-Domain BCJR Algorithm

1. Initial the forward and backward metrics $\alpha^*_0(s)$ and $\beta^*_k(s)$
2. compute the branch metrics $\gamma^*_l(s', s)$, $l = 0, 1, \ldots, K-1$
3. compute the forward metrics $\alpha^*_{l+1}(s)$, $l = 0, 1, \ldots, K-1$
4. compute the backward metrics $\beta^*_l(s')$, $l = K-1, K-2, \ldots, 0$
5. compute4 the APP L-values $L(u_l)$, $l = 0, 1, \ldots, h-1$
6. (Optional) compute the hard decisions $\hat{u}_l$, $l = 0, 1, \ldots, h-1$

Ex: BCJR decoding of a (2,1,1) systematic recursive Convolutional code on an AWGN channel

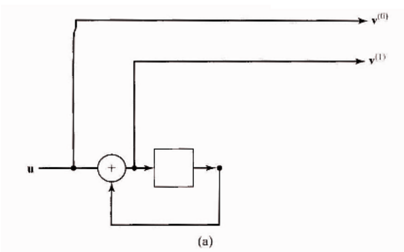


Figure 16: (2,1,1) systematic feedback encoder

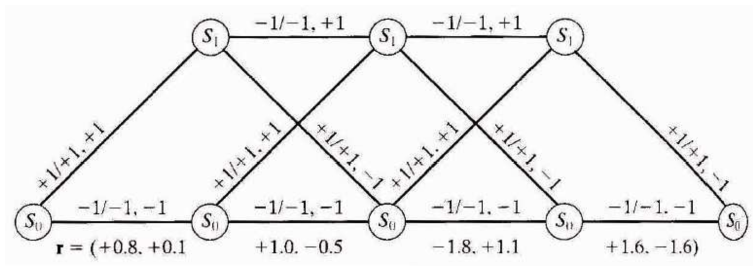


Figure 17: decoding trellis for the (2,1,1) encoder

$$
\begin{aligned}
\gamma_0^*(S_0,S_0) &= -\frac{1}{2}L_a(u_0)+\frac{1}{2}r_0\cdot v_0 \\
&= \frac{1}{2}(-0.8-0.1) = -0.45 \\
\gamma_0^*(S_0,S_1) &= +\frac{1}{2}L_a(u_0)+\frac{1}{2}r_0\cdot v_0 \\
&= \frac{1}{2}(0.8+0.1) = 0.45 \\
\gamma_1^*(S_0,S_0) &= -\frac{1}{2}L_a(u_0)+\frac{1}{2}r_0\cdot v_0 \\
&= \frac{1}{2}(-1.0+0.5) = -0.25 \\
\gamma_1^*(S_0,S_1) &= +\frac{1}{2}L_a(u_0)+\frac{1}{2}r_0\cdot v_0 \\
&= \frac{1}{2}(1.0-0.5) = 0.25
\end{aligned}
$$

$$
\begin{aligned}
\gamma_1^*(S_1,S_1) &= -\frac{1}{2}L_a(u_1)+\frac{1}{2}r_1\cdot v_1 \\
&= \frac{1}{2}(-1.0-0.5) = -0.75 \\
\gamma_1^*(S_1,S_0) &= +\frac{1}{2}L_a(u_1)+\frac{1}{2}r_1\cdot v_1 \\
&= \frac{1}{2}(1.0+0.5) = 0.75 \\
\gamma_2^*(S_0,S_0) &= -\frac{1}{2}L_a(u_2)+\frac{1}{2}r_2\cdot v_2 \\
&= \frac{1}{2}(-1.8+1.1) = 0.35 \\
\gamma_2^*(S_0,S_1) &= +\frac{1}{2}L_a(u_2)+\frac{1}{2}r_2\cdot v_2 \\
&= \frac{1}{2}(-1.8+1.1) = -0.35
\end{aligned}
$$

$$
\begin{aligned}
\gamma_2^*(S_1,S_1) &= -\frac{1}{2}L_a(u_2)+\frac{1}{2}r_2\cdot v_2 \\
&= \frac{1}{2}(1.8+1.1) = 1.45 \\
\gamma_2^*(S_1,S_0) &= +\frac{1}{2}L_a(u_2)+\frac{1}{2}r_2\cdot v_2 \\
&= \frac{1}{2}(-1.8-1.1) = -1.45 \\
\gamma_3^*(S_0,S_0) &= \frac{+1}{2}r_3\cdot v_3 \\
&= \frac{1}{2}(-1.6+1.6) = 0 \\
\gamma_3^*(S_1,S_0) &= \frac{+1}{2}r_3\cdot v_3 \\
&= \frac{1}{2}(1.6+1.6) = 1.6
\end{aligned}
$$

compute the log-domain forward metrics

$$
\begin{aligned}
\alpha_1^*(S_0) &= [\gamma_0^*(S_0,S_0)+\alpha_0^*(S_0)] = -0.45+0 = -0.45 \\
\alpha_1^*(S_1) &= [\gamma_0^*(S_0,S_1)+\alpha_0^*(S_0)] = 0.45+0 = 0.45 \\
\alpha_2^*(S_0) &= max^*\{[\gamma_1^*(S_0,S_0)+\alpha_1^*(S_0)],[\gamma_1^*(S_1,S_0)+\alpha_1^*(S_1)]\} \\
&= max^*\{[(-0.25)+(-0.45)],[(0.75)+(0.45)]\} \\
&= max^*(-0.70,+1.20) = 1.20+ln(1+e^{-|-1.9|}) = 1.34 \\
\alpha_2^*(S_1) &= max^*\{[\gamma_1^*(S_0,S_1)+\alpha_1^*(S_0)],[\gamma_1^*(S_1,S_1)+\alpha_1^*(S_1)]\} \\
&= max^*(-0.20,-0.30) = -0.20+ln(1+e^{-|0.1|}) = 0.44
\end{aligned}
$$

compute the log-domain backward metrics

$$
\begin{aligned}
\beta_3^*(S_0) &= [\gamma_3^*(S_0,S_0)+\beta_4^*(S_0)] = 0+0=0\\
\beta_3^*(S_1) &= [\gamma_3^*(S_1,S_0)+\beta_4^*(S_0)] = 1.60+0=1.60\\
\beta_2^*(S_0) &= max^*\{[\gamma_2^*(S_0,S_0)+\beta_3^*(S_0)],[\gamma_2^*(S_0,S_1)+\beta_3^*(S_1)]\}\\
&= max^*\{[(0.35)+(0)],[(-0.35)+(1.60)]\}\\
&= max^*(0.35,1.25) = 1.25+ln(1+e^{-|-0.9|}) = 1.59\\
\beta_2^*(S_1) &= max^*\{[\gamma_2^*(S_1,S_0)+\beta_3^*(S_0)],[\gamma_2^*(S_1,S_1)+\beta_3^*(S_1)]\}\\
&= max^*(-1.45,3.05) = 3.05+ln(1+e^{-|-4.5|}) = 3.06\\
\beta_1^*(S_0) &= max^*\{[\gamma_1^*(S_0,S_0)+\beta_2^*(S_0)],[\gamma_1^*(S_0,S_1)+\beta_2^*(S_1)]\}\\
&= max^*(1.34,3.31) = 3.44\\
\beta_1^*(S_1) &= max^*\{[\gamma_1^*(S_1,S_0)+\beta_2^*(S_0)],[\gamma_1^*(S_1,S_1)+\beta_2^*(S_1)]\}\\
&= max^*(2.34,2.31) = 3.02
\end{aligned}
$$

Finally compute the APP L-values for the three information bits

$$
\begin{aligned}
L(u_0) &= [\beta_1^*(S_1)+\gamma_0^*(S_0,S_1)+\alpha_0^*(S_0)] - [\beta_1^*(S_0)+\gamma_0^*(S_0,S_0)+\alpha_0^*(S_0)]\\
&= (3.47)-(2.99) = +0.48\\
L(u_1) &= max^*\{[\beta_2^*(S_0)+\gamma_1^*(S_1,S_0)+\alpha_1^*(S_1)],[\beta_2^*(S_1)+\gamma_1^*(S_0,S_1)+\alpha_1^*(S_0)]\}\\
&\quad -max^*\{[\beta_2^*(S_0)+\gamma_1^*(S_0,S_0)+\alpha_1^*(S_0)],[\beta_2^*(S_1)+\gamma_1^*(S_1,S_1)+\alpha_1^*(S_1)]\}\\
&= max^*[(2.79),(2.86)] - max^*[(0.89),(2.76)]\\
&= (3.52)-(2.90) = +0.62\\
L(u_2) &= max^*\{[\beta_3^*(S_0)+\gamma_2^*(S_1,S_0)+\alpha_2^*(S_1)],[\beta_3^*(S_1)+\gamma_2^*(S_0,S_1)+\alpha_2^*(S_0)]\}\\
&\quad -max^*\{[\beta_3^*(S_0)+\gamma_2^*(S_0,S_0)+\alpha_2^*(S_0)],[\beta_3^*(S_1)+\gamma_2^*(S_1,S_1)+\alpha_2^*(S_1)]\}\\
&= max^*[(-1.01),(2.59)] - max^*[(1.69),(3.49)]\\
&= (2.62)-(3.64) = -1.02
\end{aligned}
$$

The hard-decision outputs of the BCJR decoder for the three information bits:

$$\hat{u} = (+1,+1,-1)$$

EX:BCJR decoding of a (2,1,2) Nonsystematic Convolutional code on a DMC

Assume a binary-input, 8-ary output DMC with transition probabilities $p(r_l^{(j)}|v_l^{(j)})$ given by the following table:

$v_l^{(j)} \setminus r_l^{(j)}$	0_1	0_2	0_3	0_4	1_4	1_3	1_2	1_1
0	0.434	0.197	0.167	0.111	0.058	0.023	0.008	0.002
1	0.002	0.008	0.023	0.058	0.111	0.167	0.197	0.434

Let $u=(u_0,u_1,u_2,u_3,u_4,u_5)$ denote the input vector of length $K=h+m=6$ and $v=(v_0,v_1,v_2,v_3,v_4,v_5)$ the codeword length $N=nK=12$

$$
p(u_l=0) = \begin{cases} 2/3, & l=0,1,2,3 \quad (informationbits)\\ 1, & l=4,5 \quad (terminationbits)\end{cases}
$$

The information bits are not equally likely

The received vector is given by

$$r=(1_40_1,0_41_3,1_40_4,0_41_4,0_41_2,0_10_2)$$

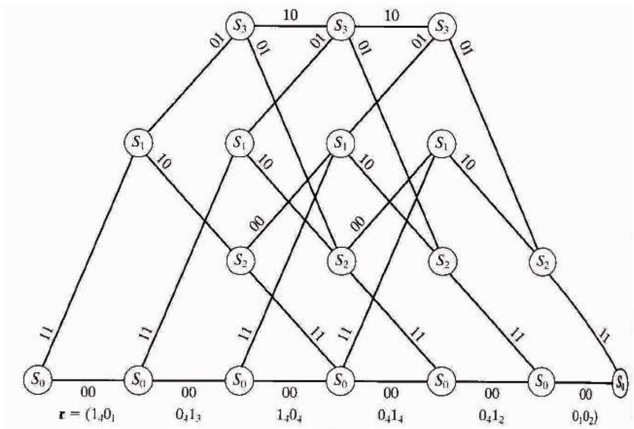


Figure 18: decoding trellis for the (2,1,2) code with K=6 and N=12

compute the (probability-domain) branch metrics

$$
\begin{aligned}
\gamma_0(S_0,S_0) &= p(u_0=0)p(1_4 0_1|00) = (2/3)p(1_4|0)p(0_1|0) \\
&= (2/3)(0.058)(0.434) = 0.01678 \\
\gamma_0(S_0,S_1) &= p(u_0=1)p(1_4 0_1|11) = (1/3)p(1_4|1)p(0_1|1) \\
&= (1/3)(0.111)(0.002) = 0.000074 \\
\vdots &= \vdots
\end{aligned}
$$

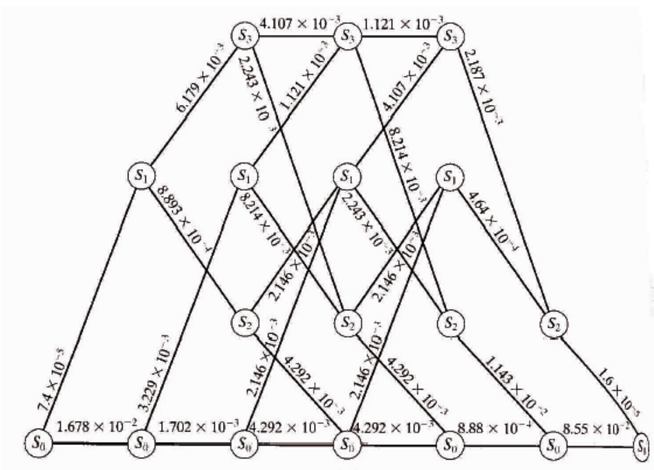


Figure 19: branch metric values $\gamma_l(S', S)$

compute the (probability-domain) normalized forward metrics

$$
\begin{aligned}
\alpha_1(S_0) &= \gamma_0(S_0,S_0)\alpha_0(S_0) = (0.01678)(1) = 0.01678 \\
\alpha_1(S_1) &= \gamma_0(S_0,S_1)\alpha_0(S_0) = (0.000074)(1) = 0.000074 \\
a_1 &= \alpha_0(S_0) + \alpha_1(S_1) = 0.01678 + 0.000074 = 0.016854 \\
A_1(S_0) &= \alpha_1(S_0)/a_1 = (0.01678)/(0.016854) = 0.9956 \\
A_1(S_1) &= 1 - A_1(S_0) = 1 - 0.9956 = 0.0044 \\
\vdots &= \vdots
\end{aligned}
$$

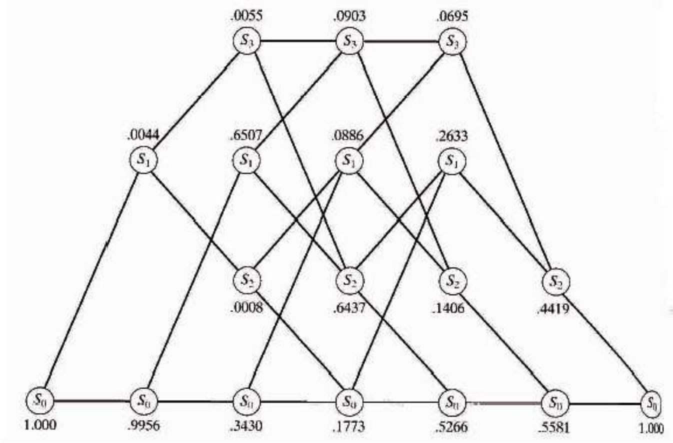


Figure 20: normalized forward metric values $\alpha_l(S)$

compute the (probability-domain) normalized backward metrics

$$\begin{aligned}
\beta_5(S_0) &= \gamma_5(S_0,S_0)\beta_6(S_0) = (0.0855)(1) = 0.0855\\
\beta_5(S_2) &= \gamma_5(S_2,S_0)\beta_6(S_0) = (0.000016)(1) = 0.000016\\
b_5 &= \beta_5(S_0)+\beta_5(S_2) = 0.0855 + 0.000016 = 0.085516\\
B_5(S_0) &= \beta_5(S_0)/b_5 = (0.0855)/(0.085516) = 0.9998\\
B_5(S_2) &= 1 - B_5(S_0) = 1 - 0.9998 = 0.0002\\
\vdots &= \vdots
\end{aligned}$$

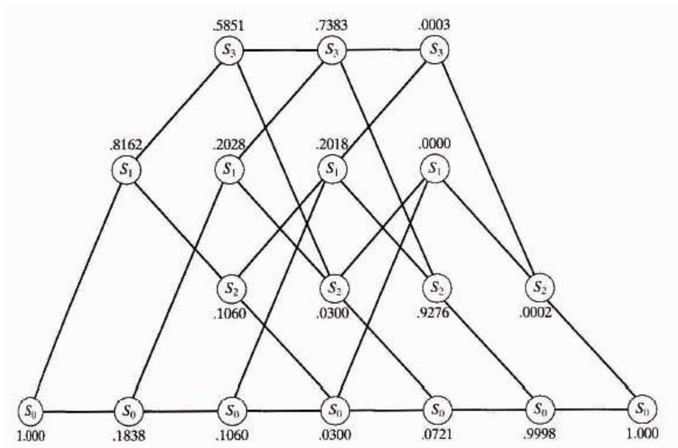


Figure 21: normalized backward metric values $\beta_l(S')$

compute the APP L-value

$$\begin{aligned}
L(u_0) &= ln\{\frac{B_1(S_1)\gamma_0(S_0,S_1)A_0(S_0)}{B_1(S_0)\gamma_0(S_0,S_0)A_0(S_0)}\}\\
&= ln\{\frac{(0.8162)(0.000074)(1)}{(0.1838)(0.01678)(1)}\} = -3.933\\
L(u_1) &= ln\{\frac{B_2(S_1)\gamma_1(S_0,S_1)A_1(S_0)+B_2(S_3)\gamma_1(S_1,S_3)A_1(S_1)}{B_2(S_0)\gamma_1(S_0,S_0)A_1(S_0)+B_2(S_2)\gamma_1(S_1,S_2)A_1(S_1)}\}\\
&= ln\{\frac{(0.2028)(0.003229)(0.9956)+(0.5851)(0.006179)(0.0044)}{(0.1060)(0.001702)(0.9956)+(0.1060)(0.0008893)(0.0044)}\}\\
&= +1.311
\end{aligned}$$

$$L(u_2) = -1.234$$

$$L(u_3) = -8.817$$

$$\hat{u} = (\hat{u}_0,\hat{u}_1,\hat{u}_2,\hat{u}_3) = (0,1,1,0)$$

Suboptimal decoding: sequential decoding

Suboptimum Decoding Of Convolutional Code

The Viterbi and BCJR decoding are not achievable in practice at rates close to capacity. This is because the decoding effort is fixed and grows exponentially with constraint length, and thus only short constraint length codes can be used.

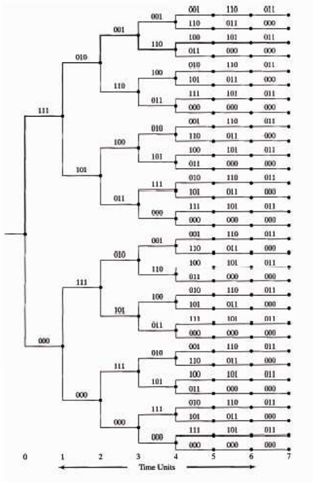
The ZJ Algorithm

1. Load the stack with the origin node in the tree,whose metric is taken to be zero
2. Compute the metric of the successors of the top path in the stack
3. Delete the top path from the stack
4. Insert the new paths from the stack and rearrange the stack in order of decreasing metric values
5. If the top in the stack ends at a terminal node in the tree, stop. Otherwise, return to step 2

ex:
The code tree for the $(3, 1, 2)$ feed forward encoder with

$$G(D) = \begin{pmatrix} 1 + D & 1 + D^2 & 1 + D + D^2 \end{pmatrix}$$

is shown below, and the information sequence of length $h = 5$.



Bit Metric For BSC

For a BSC with transition probability p, $p(r_l = 0) = p(r_l = 1) = 1/2$ for all l, and the bit metrics are given by

$$M(r_l|v_l) = \begin{cases} log_2 2p - R & r_l \neq v_l \\ log_2 2(1-p) - R & r_l = v_l \end{cases}$$

ex: For R=1/3 and p=0.1,

$$M(r_l|v_l) = \begin{cases} -2.65 \\ +0.52 \end{cases}$$

$v_i \setminus r_i$	0	1
0	+1	-5
1	−5	+1

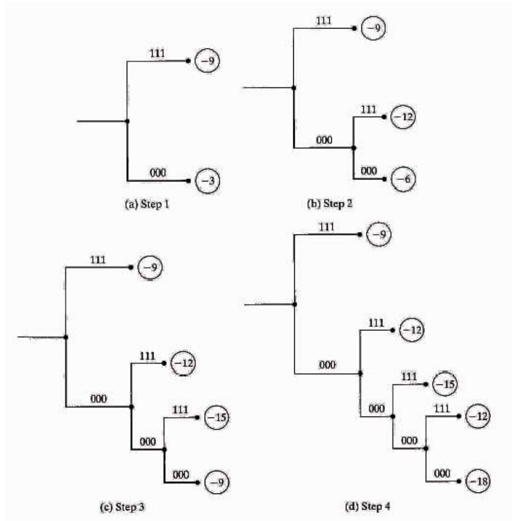
Consider the application of the ZJ algorithm to the code tree, assume a codeword is transmitted from this code over a BSC with $p = 0.10$, and the sequence

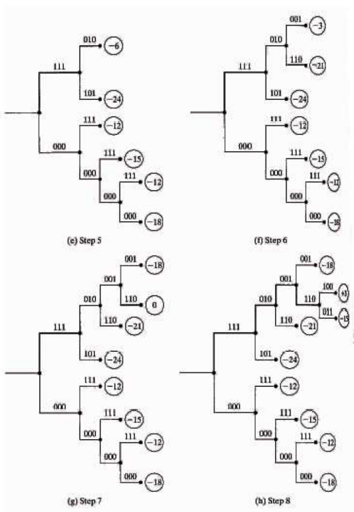
$$r = (010, 010, 001, 110, 100, 101, 011)$$

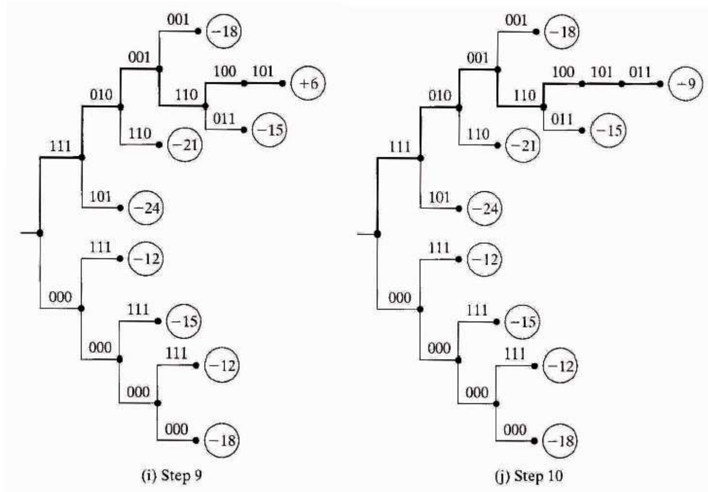
is received Using the integer metric table, we shown the contents of the stack aftereach step of the algorithm and decoding process in next slide

Step 1	Step 2	Step 3	Step 4	Step 5
0(−3)	00(−6)	000(−9)	1(−9)	11(−6)
1(−9)	1(−9)	1(−9)	0001(−12)	0001(−12)
	01(−12)	01(−12)	01(−12)	01(−12)
		001(−15)	001(−15)	001(−15)
			0000(−18)	0000(−18)
				10(−24)

Step 6	Step 7	Step 8	Step 9	Step 10
111(−3)	1110(0)	11101(+3)	111010(+6)	1110100(+9)
0001(−12)	0001(−12)	0001(−12)	0001(−12)	0001(−12)
01(−12)	01(−12)	01(−12)	01(−12)	01(−12)
001(−15)	001(−15)	11100(−15)	11100(−15)	11100(−15)
0000(−18)	1111(−18)	001(−15)	001(−15)	001(−15)
110(−21)	0000(−18)	1111(−18)	1111(−18)	1111(−18)
10(−24)	110(−21)	0000(−18)	0000(−18)	0000(−18)
	10(−24)	110(−21)	110(−21)	110(−21)
		10(−24)	10(−24)	10(−24)


(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4


(e) Step 5 (f) Step 6 (g) Step 7 (h) Step 8


(i) Step 9 (j) Step 10

The final decoding path is

$$\hat{v} = (111, 010, 001, 110, 100, 101, 011)$$

corresponding to the information sequence $\hat{u} = (11101)$

Note that $\hat{v}$ disagrees with r in only 2 positions, the fraction of error in r is $2/21 = 0.095$ which is roughly equal to the channel transition probability of $p = 0.10$

The situation is somewhat different when the received sequence r is very noisy.

From the same code, channel, matric table assume that the sequence

$$r = (110, 110, 110, 111, 010, 101, 101)$$

is received. The contents of the stack after each step of the algorithm are shown:

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
1(−3)	11(−6)	110(−3)	1100(−6)	11000(−9)	0(−9)	1101(−12)
0(−9)	0(−9)	0(−9)	0(−9)	0(−9)	1101(−12)	01(−12)
	10(−12)	10(−12)	1101(−12)	1101(−12)	10(−12)	10(−12)
		111(−21)	10(−12)	10(−12)	11001(−15)	11001(−15)
			111(−21)	11001(−15)	110000(−18)	110000(−18)
				111(−21)	111(−21)	00(−18)
						111(−21)

Step 8	Step 9	Step 10	Step 11	Step 12	Step 13	Step 14
11011(−9)	01(−12)	10(−12)	11001(−15)	110010(−12)	101(−15)	011(−15)
01(−12)	10(−12)	11001(−15)	101(−15)	101(−15)	011(−15)	110110(−18)
10(−12)	11001(−15)	011(−15)	011(−15)	011(−15)	110110(−18)	110000(−18)
11001(−15)	110110(−18)	110110(−18)	110110(−18)	110110(−18)	110000(−18)	1010(−18)
110000(−18)	110000(−18)	110000(−18)	110000(−18)	110000(−18)	00(−18)	00(−18)
00(−18)	00(−18)	00(−18)	00(−18)	00(−18)	1100100(−21)	1100100(−21)
111(−21)	111(−21)	010(−21)	100(−21)	100(−21)	100(−21)	100(−21)
11010(−27)	11010(−27)	111(−21)	010(−21)	010(−21)	010(−21)	010(−21)
		11010(−27)	111(−21)	111(−21)	111(−21)	111(−21)
			11010(−27)	11010(−27)	11010(−27)	1011(−24)
						11010(−27)

Step 15	Step 16	Step 17	Step 18	Step 19	Step 20
110110(−18)	110000(−18)	0110(−18)	1010(−18)	00(−18)	1100100(−21)
110000(−18)	0110(−18)	1010(−18)	00(−18)	1100100(−21)	10100(−21)
0110(−18)	1010(−18)	00(−18)	1100100(−21)	10100(−21)	01100(−21)
1010(−18)	00(−18)	1100100(−21)	01100(−21)	01100(−21)	001(−21)
00(−18)	1100100(−21)	100(−21)	100(−21)	100(−21)	100(−21)
1100100(−21)	100(−21)	010(−21)	010(−21)	010(−21)	010(−21)
100(−21)	010(−21)	111(−21)	111(−21)	111(−21)	111(−21)
010(−21)	111(−21)	0111(−24)	0111(−24)	0111(−24)	0111(−24)
111(−21)	0111(−24)	1011(−24)	1011(−24)	1011(−24)	1011(−24)
0111(−24)	1011(−24)	1100000(−27)	1100000(−27)	1100000(−27)	1100000(−27)
1011(−24)	1101100(−27)	1101100(−27)	1101100(−27)	1101100(−27)	1101100(−27)
11010(−27)	11010(−27)	11010(−27)	01101(−27)	10101(−27)	10101(−27)
			11010(−27)	01101(−27)	01101(−27)
				11010(−27)	11010(−27)
					000(−27)

The algorithm terminates after 20 decoding steps, and the final decoded path is

$$v = (111, 010, 110, 011, 111, 101, 011)$$

corresponding to the information sequence

$$\hat{u} = (11001)$$

In this example, the sequence decoder performs 20 computations, whereas the Viterbi algorithm would again require only 15 computations

This is because the number of computations performed by a sequential decoder is a random variable, the computation load of the Viterbi algorithm is fixed.

The fraction of errors in the received sequence r is $7/21 = 0.333$ which is much greater than the channel transition probability of $p = 0.10$

Trellises of Codes

Communication & Coding Laboratory

Dept. of Electrical Engineering,
National Chung Hsing University
E-mail: g9364106@mail.nchu.edu.tw

- **Chapter 8: Trellises of Codes**
  1. Introduction
  2. Finite-state machine model and trellis representation of a code
  3. Bit-level trellises for binary linear block codes
  4. State labeling
  5. Structural properties of bit-level trellises
  6. State labeling and trellis construction based on the parity-check matrix
  7. Trellis complexity and symmetry
  8. Trellis sectionalization and parallel decomposition
  9. Cartesian product

Introduction

- Constructing and representing codes with graphs have long been interesting problems to many coding theorist.

  ⇒ The most commonly known graphical representation of a code is the trellis representation.
- A code trellis diagram is simply an edge–labeled directed graph in which every path represents a codeword (or a code sequence for a convolutional code).
- This representation makes it possible to implement maximum likelihood decoding (MLD) of a code with a significant reduction in decoding complexity.

The history of trellis representation

- Trellis representation was first introduced by Forney in 1973 as a means of explaining the decoding algorithm for convolutional codes devised by Viterbi.
- Trellis representation of linear block codes was first presented by Bahl, Cocke, Jelinek, and Raviv in 1974.
- In 1988, Forney showed that some block codes, such as RM codes and some lattice codes, have relatively simple trellis structures.

Finite-state machine model

- Let $\Gamma = 0,1,2,...$ denote the entire encoding interval (or span) that consists of a sequence of encoding time instants.
- A unit encoding interval:
  The interval between two consecutive time instants.
- During this unit encoding interval, code symbols are generated at the output of the encoder based on

  { The current input information symbols
  The past information symbols that are stored in the memory

  , according to a certain encoding rule.

- A specific state of the encoder at that time instant is defined by the information symbols stored in the memory at any encoding time define .
- The state of the encoder at time-i is defined by those information symbols, stored in the memory at time–i, that affect the current output code symbols during the interval from time–i to time–$(i + 1)$ and future output code symbols.
- A state transition:
  As new information symbols are shifted into the memory some old information symbols may be shifted out of the encoder, and there is a transition from one state to another state.

- With there definitions of a state and a state transition, the encoder can be modeled as a finite-state machine, as shown in Figure 1.

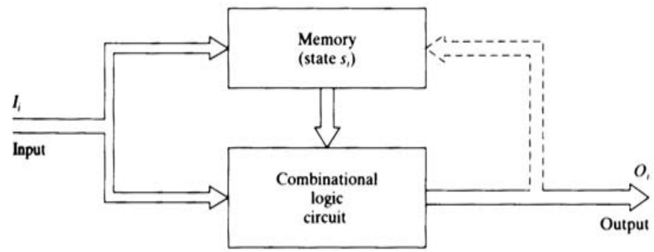


Figure 1: A finite-state machine model for an encoder with finite memory.

Trellis diagram

- The dynamic behavior of the encoder can be graphically represented by a state diagram in time.
- It consists of levels of nodes and edges connecting the nodes of one level to the nodes of the next level.

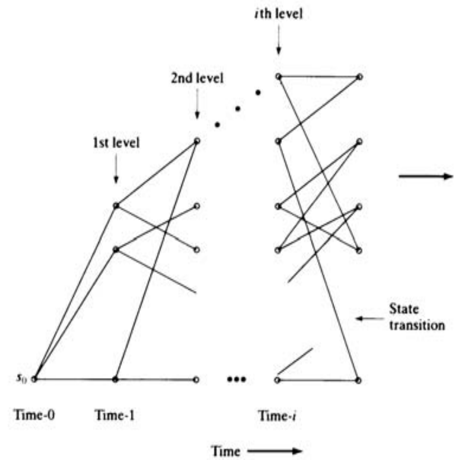


Figure 2: Trellis representation of a finite-state encoder.

Trellis representation of a code

- This state transition, in the trellis diagram, is represented by a directed edge (commonly called a branch). Each branch has a label.
- The set of allowable states at a given time instant i is called the state space of the encoder at time-i, denoted by $\Sigma_i(C)$.
- A state $s_i \in \sum_i(C)$ is said to be reachable if there exists an information sequence that takes (drives) the encoder from the initial state s_0 to the state s_i at time-i.
- In the trellis, every node at level-i for $i \in \Gamma$ is connected by a path (defined as a sequence of connected branches) from the initial node.

- Every node in the trellis has at least one incoming branch except for the initial node and at least one outgoing branch except for a node that represents the final state of the encoder at the end of the entire encoding interval.
- In this graphical representation of a code, there is a one-to-one correspondence between a codeword (or code sequence) and a path in the code trellis.
- Every path in the code trellis represents a codeword.

- For $i \in \Gamma$, let

  I_i: The input information block

  O_i: The output code block

  , during the interval from time-i to time-$(i+1)$.
- The dynamic behavior of the encoder for a linear code is governed by two functions:
  1. Output function:
     $$O_i = f_i(s_i,\ I_i),$$
     where $f_i(s_i,\ I_i) \neq f_i(s_i,\ I_i^{'})$ for $I_i \neq I_i^{'}$.
  2. State transition function:
     $$s_{i+1} = g_i(s_i,\ I_i),$$
     where $s_i \in \Sigma_i(\mathrm{C})$ and $s_{i+1} \in \Sigma_{i+1}(\mathrm{C})$ are called the current and next states, respectively.

- A code trellis is said to be time-invariant if there exists a finite period $\{0, 1, ..., v\} \subset \Gamma$ and a state space $\Sigma(C)$ such that
  1. $\Sigma_i(C) \subset \Sigma(C)$ for $0 < i < v$, and $\Sigma_i(C) = \Sigma(C)$ for $i \geq v$
  2. $f_i = f$ and $g_i = g$ for all $i \in \Gamma$.
- In general:

Block code	A trellis diagram is time-varying.
convolutional code	A trellis diagram is usually time-invariant.

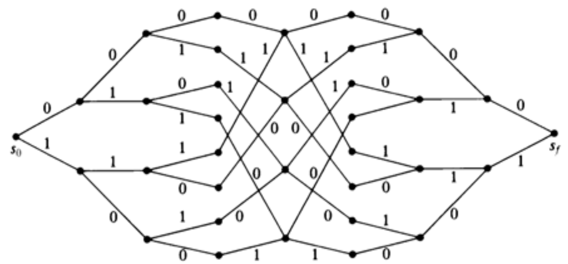


Figure 3: A time-varying trellis diagram for a block code.

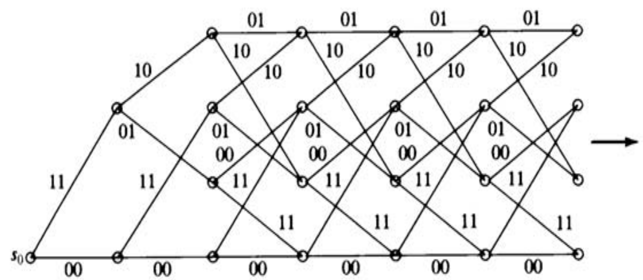


Figure 4: A time-invariant trellis diagram for a convolutional code.

Bit-level trellis for binary block codes

- Definition:
  An n-section bit-level trellis diagram for a binary linear block code C of length n, denote by T, is a directed graph consisting of $n+1$ levels of nodes (called **states**) and branches (also called **edges**) such that:
  1. For $0 \leq i \leq n$, the nodes at the ith level represent the states in the state space $\sum_i(C)$ of the encoder $E(C)$ at time-i.
     - At time-0 (or the zeroth level) there is only one node, denoted by s_0, called **the initial node** (or state).
     - At time-n (or the nth level), there is only one node, denoted by s_f (or s_n), called **the final node** (or state).
  2. For $0 \leq i < n$, a branch in the section of the trellis T between the ith level and the $(i+1)$th level (or time–i and time–$(i+1)$) connects a state $s_i \in \sum_i(C)$ to a state $s_{i+1} \in \sum_{i+1}(C)$ and is labeled with a code bit v_i that represents the encoder output in the bit interval from time-i to time–$(i+1)$. **A branch represents a state transition**.

3. Except for the initial node, every node has at least one, but no more than two, incoming branches. Except for the final node, every node has at least one, but no more than two, outgoing branches. **The initial node has no incoming branches. The finial node has no outgoing branches**. Two branches diverging from the same node have different labels; they represent two different transitions from the same starting state.
4. There is a directed path from the initial node s_0 to the final node s_f with a label sequence $(v_0, v_1, \ldots, v_{n-1})$ if and only if $(v_0, v_1, \ldots, v_{n-1})$ is a codeword in C.

- To Define
  $$\rho_i(C) \triangleq \log_2 |\Sigma_i(C)|,$$
  which is called the "state space dimension at time-i".
- We simply use ρ_i for $\rho_i(C)$ for simplicity. The sequence $(\rho_0,\rho_1,\ldots,\rho_n)$ is called the state space dimension profile.
- **Example**:
  From Figure 3 we find that the state space complexity profile and the state space dimension profile for the $(8, 4)$ RM code are $(1, 2, 4, 8, 4, 8, 4, 2, 1)$ and $(0, 1, 2, 3, 2, 3, 2, 1, 0)$, respectively.

- To facilitate the code trellis construction, we arrange the generator matrix G in a special form.
  Let $v = (v_0, v_1, \ldots, v_{n-1})$ be a nonzero binary n-tuple.
- The first nonzero component of v is called the leading 1 of v, and the last nonzero component of v is called the trailing 1 of v.
- A generator matrix G for C is said to be in trellis oriented form (TOF) if the following two conditions hold:
  1. The leading 1 of each row of G appears in a column before the leading 1 of any row below it.
  2. No two rows have their trailing 1's in the same column.

- Any generator matrix for C can be put in TOF by two steps of Gaussian elimination.[a]
- **Example**:
Consider the first-order RM code of length 8, RM(1,3). It is an $(8,4)$ code with the following generator matrix [b]:

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

[a]Note that a generate matrix in TOF is not necessarily in systematic form.
[b]It is not in TOF.

By interchanging the second and the fourth rows, we have

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

We add the fourth row of the matrix to the first, second, and third rows. These additions result in the following matrix in TOF:

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

where TOGM stands for trellis oriented generator matrix.

- Let $g = (g_0, g_1, \ldots, g_{n-1})$ be a row in $G_{\texttt{TOGM}}$ for code C.
  1. Let $\emptyset(g) = (i, i+1, \ldots, j)$ denote the smallest index interval that contains all the nonzero components of g.
  2. This say that $g_i = 1$ and $g_j = 1$, and they are the leading and trailing 1's of g, respectively.
  3. This interval $\emptyset(g) = (i, i+1, \ldots, j)$ is called the digit (or bit) span of g.
  4. We define the time span of g, denoted by $\tau(g)$, as the following time interval: $\tau(g) \triangleq (i, i+1, \ldots, j+1)$.
  5. For simplicity, we write $\emptyset(g) = [i, j]$, and $\tau(g) = [i, j+1]$, we define the active time span of g, denoted by $\tau_a(g)$, as the time interval. $\tau_a(g) \triangleq [i+1, j]$, for $j > i$.

- We define the active time span of g, denote by $\tau_a(g)$, as the time interval

$$\tau_a(g) \triangleq \begin{cases} [i+1, j], & \text{for } j > i \\ \emptyset(\texttt{empty set}), & \text{for } j = i. \end{cases}$$

- **Example**: To consider the TOGM of the $(8,4)$ RM code

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

  - We can find that the bit spans of the rows are $\emptyset(g_0) = [0,3]$, $\emptyset(g_1) = [1,6]$, $\emptyset(g_2) = [2,5]$, $\emptyset(g_3) = [4,7]$.
  - The time spans of the rows are $\tau(g_0) = [0,4]$, $\tau(g_1) = [1,7]$, $\tau(g_2) = [2,6]$, $\tau(g_3) = [4,8]$.
  - The active time spans of the rows are $\tau_a(g_0) = [1,3]$, $\tau_a(g_1) = [2,6]$, $\tau_a(g_2) = [3,5]$, $\tau_a(g_3) = [5,7]$.

- Now, we give a mathematical formulation of the state space of the n-section bit-level trellis for an (n,k) linear code C over $GF(2)$ with a TOGM $G_{\texttt{TOGM}}$.
- At time-i, $0 \le i \le n$, we divide the rows of $G_{\texttt{TOGM}}$ into three disjoint subsets:
  1. G_i^p consists of those rows of $G_{\texttt{TOGM}}$ whose bit spans are contained in the interval $[0, i-1]$.
  2. G_i^f consists of those rows of $G_{\texttt{TOGM}}$ whose bit spans are contained in the interval $[i, n-1]$.
  3. G_i^s consists of those rows of $G_{\texttt{TOGM}}$ whose active time spans contain time-i.

- **Example**: To consider the TOGM of the $(8,4)$ RM code

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

At time–2, we find that

$$G_2^p = \phi,\ G_2^f = \{g_2, g_3\},\ G_2^s = \{g_0, g_1\},$$

where ϕ denotes the empty set. At time–4, we find that

$$G_4^p = \{g_0\},\ G_4^f = \{g_3\},\ G_4^s = \{g_1, g_2\},$$

- Let A_i^p, A_i^f, and A_i^s denote the subsets of information bits, $a_0, a_1, \ldots, a_{k-1}$, that correspond to the rows of G_i^p, G_i^f, and G_i^s, respectively.
- The bits in A_i^s are the information bits stored in the encoder memory that affect the current output code bit v_i and the future output code bits beyond time-i. There information bits in A_i^s hence define a state of the encoder $E(C)$ for the code C at time-i.
- Let $\rho_i \triangleq |A_i^s| = |G_i^s|$.
  Then, there are 2^{ρ_i} distinct states in which the encoder $E(C)$ can reside at time–i; each state is defined by a specific combination of the ρ_i[a] information bits in A_i^s[b].

[a]The parameter ρ_i is the dimension of the state space $\sum_i(C)$.
[b]The set A_i^s is called the state defining information set at time–i

Table 1: Partition of the TOGM of the $(8, 4)$ RM code

Time i	G_i^p	G_i^f	G_i^s	ρ_i
0	ϕ	$\{g_0, g_1, g_2, g_3\}$	ϕ	0
1	ϕ	$\{g_1, g_2, g_3\}$	$\{g_0\}$	1
2	ϕ	$\{g_2, g_3\}$	$\{g_0, g_1\}$	2
3	ϕ	$\{g_3\}$	$\{g_0, g_1, g_2\}$	3
4	$\{g_0\}$	$\{g_3\}$	$\{g_1, g_2\}$	2
5	$\{g_0\}$	ϕ	$\{g_1, g_2, g_3\}$	3
6	$\{g_0, g_2\}$	ϕ	$\{g_1, g_3\}$	2
7	$\{g_0, g_1, g_2\}$	ϕ	$\{g_3\}$	1
8	$\{g_0, g_1, g_2, g_3\}$	ϕ	ϕ	0

- For $0 \leq i < n$, suppose the encoder $E(C)$ is in state $s_i \in \Sigma_i(C)$. From time-i to time-$(i+1)$, $E(C)$ generates a code bit v_i and moves from state s_i to a state $s_{i+1} \in \Sigma_{i+1}(C)$.
- Let
$$G_i^s = g_1^{(i)}, g_2^{(i)}, ..., g_{\rho_i}^{(i)}$$
and
$$A_i^s = a_1^{(i)}, a_2^{(i)}, \ldots, a_{\rho_i}^{(i)},$$
where $\rho_i = |A_i^s| = |G_i^s|$. The current state s_i of the encoder is defined by a specific combination of the information bits in A_i^s.

- Let g^* denote the row in G_i^f whose leading 1 is at bit position i. Let g_i^* denote the ith component of g^*. Then, $g_i^* = 1$.
- Let a^* denote the information bit that corresponds to row g^*.
- The output code bit v_i generated during the bit interval between time-i and time-$(i+1)$ is given by
$$v_i = a^* + \Sigma_{l=1}^{\rho_i} a_l^{(i)} \cdot g_{l,i}^{(i)},$$
where $g_{l,i}^{(i)}$ is the ith component of $g_l^{(i)}$ in G_i^s.
- Note that the information bit a^* begins to affect the output of the encoder $E(C)$ at time-i. For this reason, bit a^* is regarded as the current input information bit.

- The output code bit v_i can have two possible values depending on the current input information bit a^*.
- Suppose there is no such row g^* in G_i^f. Then, the output code bit v_i at time-i is given by
$$v_i = \Sigma_{l=1}^{\rho_i} a_l^{(i)} \cdot g_{l,i}^{(i)}.$$
that is, $a^* = 0$ (this is called a dummy information bit).

- **Example**: To consider the TOGM of the $(8,4)$ RM code

$$G_{\text{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

  – From Table 1 we find that at time–2, $G_2^p = \phi$, $G_2^f = \{g_2, g_3\}$, and $G_2^s = \{g_0, g_1\}$. Therefore, $A_2^s = \{a_0, a_1\}$, the information bit a_0 and a_1 define the state of the encoder at time–2, and there are four distinct state defined by four combinations of values of a_0 and a_1, $\{00, 01, 10, 11\}$.

  – We also find that $g^* = g_2$. Hence, the current input information bit at time–2 is $a^* = a_2$. The current output code bit v_2 is given by

$$\begin{aligned} v_2 &= \boxed{a_2} + a_0 \cdot g_{02} + a_1 \cdot g_{12} \\ &= \boxed{a_2} + a_0 \cdot 1 + a_1 \cdot 1 \\ &= \boxed{a_2} + a_0, \end{aligned}$$

  where $\boxed{a_2}$ denotes the current input.

  – For every state defined by a_0 and a_1, v_2 has two possible values depending on a_2. In the trellis there are two branches diverging from each state at time–2, as shown in Figure 3.

  – Now, consider time–3. For $i = 3$, we find that $G_3^p = \phi$, $G_3^f = \{g_3\}$, and $G_3^s = \{g_0, g_1, g_2\}$. Therefore, $A_3^s = \{a_0, a_1, a_2\}$, and the information bits in A_3^s define eight states at time–3, as shown in Figure 3.

  – There is no row g^* in G_3^f with leading 1 at bit position $i = 3$. Hence, we set the current input information bit $a^* = 0$. The output code bit v_3 is given by $v_i = \Sigma_{l=1}^{\rho_i} a_l^{(i)} \cdot g_{l,i}^{(i)}$,

$$\begin{aligned} v_3 &= a_0 \cdot g_{03} + a_1 \cdot g_{13} + a_2 \cdot g_{23} \\ v_3 &= a_0 \cdot 1 + a_1 \cdot 1 + a_2 \cdot 1 \quad . \\ &= a_0 + a_1 + a_2 \end{aligned}$$

  – In the trellis, there is only one branch diverging from each of the eight states, as shown in Figure 3.

- Let g^0 denote the row in G_i^s whose trailing 1 is at bit position-i, that is, the ith component g_i^0 of g^0 is the last nonzero component of g^0 [a].
- Let a^0 be the information bit in A_i^s that corresponds to row g^0. Then, at time–$(i+1)$,

$$G_{i+1}^s = \left(G_i^s \setminus \{g^0\}\right) \cup \{g^*\}$$

  and

$$A_{i+1}^s = (A_i^s \setminus \{a^0\}) \cup \{a^*\}$$

  The information bits in A_{i+1}^s define the state space $\Sigma_{i+1}(C)$ at time-$(i+1)$.

[a] Note that this row g^0 may not exist.

- Now, we want to construct the $(i+1)$th section from time-i to time-$(i+1)$. The state space $\Sigma_i(C)$ is known. The $(i+1)$th section is constructed as follows:
  1. Determine G^s_{i+1} and A^s_{i+1}. Form the state space $\Sigma_{i+1}(C)$ at time-$(i+1)$.
  2. For each state $s_i \in \Sigma_i(C)$, determine its state transition(s) based on the change from A^s_i to A^s_{i+1}. Connect s_i to its adjacent state(s) in $\Sigma_{i+1}(C)$ by branches.
  3. For each state transition, determine the output code bit v_i from the output function of $v_i = a^* + \Sigma_{l=1}^{\rho_i} a_l^{(i)} \cdot g_{l,i}^{(i)}$ or $v_i = \Sigma_{l=1}^{\rho_i} a_l^{(i)} \cdot g_{l,i}^{(i)}$, and label the corresponding branch in the trellis with v_i.

- **Example**: To consider the TOGM of the $(8, 4)$ RM code

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

  – Suppose the trellis for this code has been constructed up to time–4. At this point, we find from Table 1 that $G^s_4 = \{g_1, g_2\}$. The state space $\sum_4(C)$ is defined by $A^s_4 = \{a_1, a_2\}$. There are four states at time–4, which are determined by the four combinations of a_1 and $a_2 : \{(0,0), (0,1), (1,0), (1,1)\}$.

  – To construct the trellis section from time–4 to time–5, we find that there is no row g^0 in $G^s_4 = \{g_1, g_2\}$, but there is a row g^* in $G^f_4 = \{g_3\}$, which is g_3.

  – Therefore, at time–5, we have

$$G^s_5 = \{g_1, g_2, g_3\}$$

  and

$$A^s_5 = \{a_1, a_2, a_3\}.$$

  The state space $\sum_5(C)$ is then defined by the three bits in A^s_5. The eight states in $\sum_5(C)$ are defined by the eight combinations of $a_1, a_2, ,$ and a_3 : $\{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$.

  – Suppose the current state s_4 at time–4 is defined by (a_1, a_2). Then the next state at time–5 is either the state s_5 defined by $(a_1, a_2, a_3 = 0)$ or the state s'_5 defined by $(a_1, a_2, a_3 = 1)$. The output code bit v_4 is given by

$$v_4 = \boxed{a_3} + a_1 \cdot 1 + a_2 \cdot 1.$$

  which has two values depending on whether the current input bit a_3 is 0 or 1.

  – Connecting each state at time–4 to its two adjacent states at time–5 by branches and labeling each branch with the corresponding code bit v_4 for either $a_3 = 0$ or $a_3 = 1$, we complete the trellis section from time–4 to time–5. To construct the next trellis section from time–5 to time–6, we first find that there is a row g^0 in $G^s_5 = \{g_1, g_2, g_3\}$, which is g_2, and there is no row g^* in $G^f_5 = \phi$. Therefore,

$$G^s_6 = G^s_5 \ \{g_2\} = \{g_1, g_3\},$$

  and

$$A^s_6 = \{a_1, a_3\}.$$

  – From the change from A^s_5 to A^s_6, we find that two states defined by $(a_1, a_2 = 0, a_3)$ and $(a_1, a_2 = 1, a_3)$ at time–5 move into the same state defined by (a_1, a_3) at time–6.

– The two connecting branches are labeled with

$$v_5 = \boxed{0} + a_1 \cdot 0 + (a_2 = 0) \cdot 1 + a_3 \cdot 1,$$

and

$$v_5 = \boxed{0} + a_1 \cdot 0 + (a_2 = 1) \cdot 1 + a_3 \cdot 1,$$

respectively, where $\boxed{0}$ denotes the dummy input. This complete the construction of the trellis section from time–5 to time–6. Continue with this construction process until the trellis terminates at time–8.

State labeling

- In a code trellis, each state is labeled by a fixed sequence (or a given name).
- This labeling can be accomplished by using a k-tuple $l(s)$ with components corresponding to the k information bits, a_0, a_1,..., a_{k-1}, in a message.
- At time–i, all the components of $l(s)$ are set to zero except for the components at the positions corresponding to the information bits in $A_l^s = \{a_1^{(i)}, a_2^{(i)}, \ldots, a_{\rho_i}^{(i)}\}$.

- **Example**:
  Consider the (8,4) code given in before Example. At time–4, the state-defining information set is $A_4^s = \{a_1, a_2\}$.
  - There are four states corresponding to four combinations of a_1 and a_2. Therefore, the label for each of these four states is given by $(0, a_1, a_2, 0)$.
  - At time–5, $A_5^s = \{a_1, a_2, a_3\}$, and there are eight states. The label for each of these eight states is given by $(0, a_1, a_2, a_3)$.

- **The trellis construction procedure**:
  Suppose the trellis T has been constructed up to section-i. At this point, G_i^s, A_i^s, and $\sum_i(C)$ are known. The $(i+1)$th section is constructed as follows:
  1. Determine G_{i+1}^s and A_{i+1}^s from $G_{i+1}^s = \left(G_i^s \setminus \{g^0\}\right) \cup \{g^*\}$ and $A_{i+1}^s = (A_i^s \setminus \{a^0\}) \cup \{a^*\}$.
  2. Form the state space $\sum_{i+1}(C)$ at time–$(i+1)$ and label each state in $\sum_{i+1}(C)$ based on A_{i+1}^s. The state in $\sum_{i+1}(C)$ form the nodes of the code trellis T at the $(i+1)$th level.
  3. For each state $s_i \in \sum_i(C)$ at time–i, determined its transition(s) to the state(s) in $\sum_{i+1}(C)$ based on the information bits a^* and a^0. For each transition from a state $s_i \in \sum_i(C)$ to a state $s_{i+1} \in \sum_{i+1}(C)$, connect the state s_i to the state s_{i+1} by a branch (s_i, s_{i+1}).
  4. For each state transition (s_i, s_{i+1}), determine the output code bit v_i and label the branch (s_i, s_{i+1}) with v_i.

- **Example**: To consider the TOGM of the (8, 4) RM code

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

  – For $0 \leq i \leq 8$, we determined the submatrix G_i^s and the state-defining information set A_i^s as listed in Table 2.
  – From A_i^s we form the label for each state in $\sum_i(C)$ as shown in Table 2. The state transitions from time–i to time–$(i+1)$ are determined by change from A_i^s to A_{i+1}^s.
  – Following the trellis construction procedure given described, we obtain the 8-section trellis diagram for the (8.4) RM code as shown in Figure 5. Each state in the trellis is labeled by a 4-tuple.

i	G_i^s	a^*	a^0	A_i^s	State label
0	ϕ	a_0	–	ϕ	(0000)
1	$\{g_0\}$	a_1	–	$\{a_0\}$	$(a_0 000)$
2	$\{g_0, g_1\}$	a_2	–	$\{a_0, a_1\}$	$(a_0 a_1 00)$
3	$\{g_0, g_1, g_2\}$	–	a_0	$\{a_0, a_1, a_2\}$	$(a_0 a_1 a_2 0)$
4	$\{g_1, g_2\}$	a_3	–	$\{a_1, a_2\}$	$(0 a_1 a_2 0)$
5	$\{g_1, g_2, g_3\}$	–	a_2	$\{a_1, a_2, a_3\}$	$(0 a_1 a_2 a_3)$
6	$\{g_1, g_3\}$	–	a_1	$\{a_1, a_3\}$	$(0 a_1 0 a_3)$
7	$\{g_3\}$	–	a_3	$\{a_3\}$	$(000 a_3)$
8	ϕ	–	–	ϕ	(0000)

Table 2: State–defining sets and labels for the 8–section trellis for (8, 4) RM code.

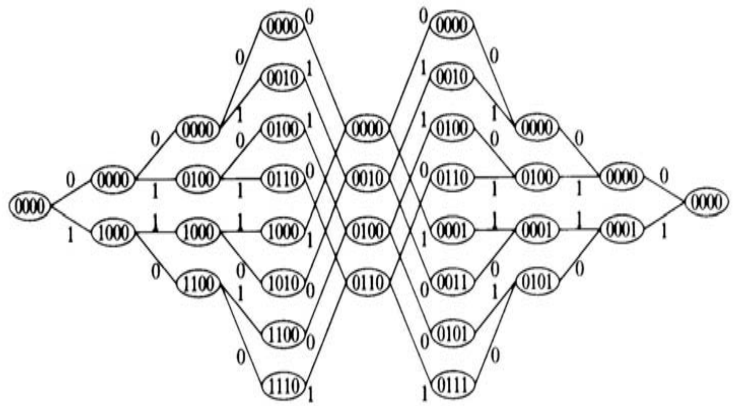


Figure 5: The 8–section trellis diagram for (8, 4) RM code with state labeling by the state–defining information set.

- Let $(\rho_0, \rho_1, \ldots, \rho_n)$ be the state space dimension profile of the trellis. We define
$$\rho_{\max}(C) \triangleq \max_{0 \leq i \leq n} \rho_i,$$
which is simply the maximum state dimension of the trellis.
- Because $\rho_i = |G_i^s| = |A_i^s|$ for $0 \leq i \leq n$, and G_i^s is a submatrix of the generator matrix of C, we have
$$\rho_{\max}(C) \leq k.$$
- For each state $s_i \in \sum_i(C)$, we form a $\rho_{\max}(C)$–tuple, denote by $l(s_i)$, in which the first ρ_i components are simply $a_1^{(i)}$, $a_2^{(i)}$, ... , $a_{\rho_i}^{(i)}$, and the remaining $\rho_{\max}(C) - \rho_i$ components are set to 0; that is,
$$l(s_i) \triangleq (a_1^{(i)}, a_2^{(i)}, \ldots, a_{\rho_i}^{(i)}, 0, 0, \ldots, 0).$$
Then, $l(s_i)$ is the label for the state s_i.

- **Example**:
  For the $(8,4)$ RM code, the state space dimension profile of its 8–section trellis is $(0,1,2,3,2,3,2,1,0)$.
  - Hence, $\rho_{\max}(C) = 3$. Using 3 bits for labeling the states as described previously, we give the state labels in Table 3.
  - Compared with the state labeling given in before example, 1 bit is saved.

i	A_i^s	State label
0	ϕ	(000)
1	$\{a_0\}$	$(a_0 00)$
2	$\{a_0, a_1\}$	$(a_0 a_1 0)$
3	$\{a_0, a_1, a_2\}$	$(a_0 a_1 a_2)$
4	$\{a_1, a_2\}$	$(a_1 a_2 0)$
5	$\{a_1, a_2, a_3\}$	$(a_1 a_2 a_3)$
6	$\{a_1, a_3\}$	$(a_1 a_3 0)$
7	$\{a_3\}$	$(a_3 00)$
8	ϕ	(000)

Table 3: State labeling for the $(8,4)$ RM code using $\rho_{\max}(C) = 3$ bits.

- **Example**:
  Consider the second-order RM code of length 16.
  It is a $(16,11)$ code with a minimum distance of 4. We obtain the following TOGM:

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \\ g_6 \\ g_7 \\ g_8 \\ g_9 \\ g_{10} \end{bmatrix} = \begin{bmatrix} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&1&0&1&1&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&1&1&1&0&1&0&0&0&1&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&1&1&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \end{bmatrix}$$

- From this TOGM, we easily find the state space dimension profile, $(0,1,2,3,3,4,4\ ,4,3,4,4,4,3,3,2,1,0)$.
- The maximum state space dimension is $\rho_{\max}(C) = 4$.
- We can use 4 bits to label the trellis states.
  The state-defining information sets and state labels at each time instant are given in Table 4, shown in Figure 6.

i	G_i^s	a^*	a^0	A_i^s	State label
0	ϕ	a_0	–	ϕ	(0000)
1	$\{g_0\}$	a_1	–	$\{a_0\}$	$(a_0 000)$
2	$\{g_0, g_1\}$	a_2	–	$\{a_0, a_1\}$	$(a_0 a_1 00)$
3	$\{g_0, g_1, g_2\}$	a_3	a_0	$\{a_0, a_1, a_2\}$	$(a_0 a_1 a_2 0)$
4	$\{g_1, g_2, g_3\}$	a_4	–	$\{a_1, a_2, a_3\}$	$(a_1 a_2 a_3 0)$
5	$\{g_1, g_2, g_3, g_4\}$	a_5	a_2	$\{a_1, a_2, a_3, a_4\}$	$(a_1 a_2 a_3 a_4)$
6	$\{g_1, g_3, g_4, g_5\}$	a_6	a_1	$\{a_1, a_3, a_4, a_5\}$	$(a_1 a_3 a_4 a_5)$
7	$\{g_3, g_4, g_5, g_6\}$	–	a_4	$\{a_3, a_4, a_5, a_6\}$	$(a_3 a_4 a_5 a_6)$
8	$\{g_3, g_5, g_6\}$	a_7	–	$\{a_3, a_5, a_6\}$	$(a_3 a_5 a_6 0)$
9	$\{g_3, g_5, g_6, g_7\}$	a_8	a_6	$\{a_3, a_5, a_6, a_7\}$	$(a_3 a_5 a_6 a_7)$
10	$\{g_3, g_5, g_7, g_8\}$	a_9	a_5	$\{a_3, a_5, a_7, a_8\}$	$(a_3 a_5 a_7 a_8)$
11	$\{g_3, g_7, g_8, g_9\}$	–	a_7	$\{a_3, a_7, a_8, a_9\}$	$(a_3 a_7 a_8 a_9)$
12	$\{g_3, g_8, g_9\}$	a_{10}	a_3	$\{a_3, a_8, a_9\}$	$(a_3 a_8 a_9 0)$
13	$\{g_8, g_9, g_{10}\}$	–	a_9	$\{a_8, a_{10}\}$	$(a_8 a_9 a_{10} 0)$
14	$\{g_8, g_{10}\}$	–	a_8	$\{a_{10}\}$	$(a_8 a_{10} 00)$
15	$\{g_{10}\}$	–	a_{10}	ϕ	$(a_{10} 000)$
16	ϕ	–	–	ϕ	(0000)

Table 4: State-defining sets and labels for the 16–section trellis for the $(16, 11)$ RM code.

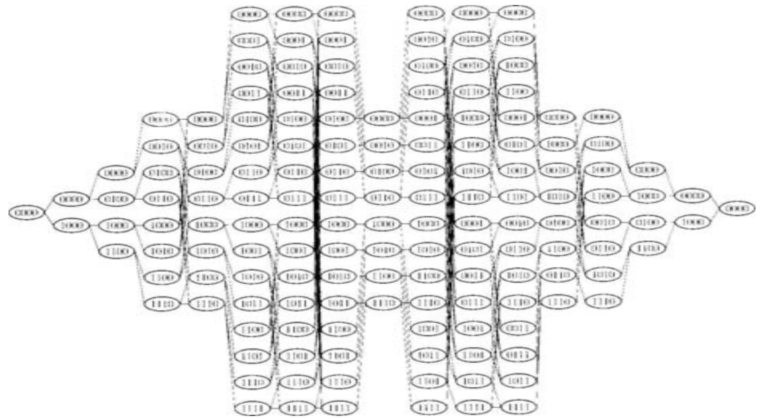


Figure 6: 16–section trellis for the $(16, 11)$ RM code with state labeling by the state-defining information sets.

Structure properties of trellises

- For $0 \leq i < j < n$, let $C_{i,j}$ denote the subcode of C consisting of those codewords in C whose nonzero components are confined to the span of $j - i$ consecutive positions in the set $\{i, i+1, \ldots, j-1\}$.
- Clearly, every codeword in $C_{i,j}$ is of the form

$$(\underbrace{0, 0, \ldots, 0}_{i}, v_i, v_{i+1}, \ldots, v_{j-1}, \underbrace{0, 0, \ldots, 0}_{n-j}),$$

and $C_{i,j}$ is a subcode of C.

- The two subcodes $C_{0,i}$ and $C_{i,n}$ are spanned by the rows in G_i^p and G_i^f, respectively, and they are called the past and future subcodes of C with respective to time–i.
- For a linear code B, let $k(B)$ denote its dimension. Then, $k(C_{0,i}) = |G_i^p|$, and $k(C_{i,n}) = |G_i^f|$.
- The dimension of the state space $\sum_i(C)$ at time-i is

$$\rho_i(C) = |G_i^s|,$$

then, it follows from the definitions of G_i^s, G_i^p, and G_i^f that

$$\begin{aligned} \rho_i(C) &= k - |G_i^p| - |G_i^f| \\ &= k - k(C_{0,i}) - k(C_{i,n}). \end{aligned}$$

- The direct–sum of $C_{0,i}$ and $C_{i,n}$[a], denote by $C_{0,i} \oplus C_{i,n}$, is a subcode of C with dimension $k(C_{0,i}) + k(C_{i,n})$. The partition $C/(C_{0,i} \oplus C_{i,n})$ consists of

$$\begin{aligned} |C/(C_{0,i} \oplus C_{i,n})| &= 2^{k-k(C_{0,i})-k(C_{i,n})} \\ &= 2^{\rho_i} \end{aligned}$$

- Let S_i denote the subspace of C that is spanned by the rows in the submatrix G_i^s. Then, each codeword in S_i is given by

$$\begin{aligned} v &= (a_1^{(i)}, a_2^{(i)}, \ldots, a_{\rho_i}^{(i)}) \cdot G_i^s \\ &= a_1^{(i)} \cdot g_1^{(i)} + a_2^{(i)} \cdot g_2^{(i)} + \ldots + a_{\rho_i}^{(i)} \cdot g_{\rho_i}^{(i)} \end{aligned}$$

where $a_l^{(i)} \in A_i^s$ for $1 \leq l \leq \rho_i$. [b]

[a] Note that $C_{0,i}$ and $C_{i,n}$ have only the all zero codeword **0** in common.

[b] We see that there is one-to-one correspondence between v and the state $s_i \in \sum_i(C)$ defined by $(a_1^{(i)}, a_2^{(i)}, \ldots, a_{\rho_i}^{(i)})$.

- It follows from the definitions of G_i^p, G_i^f, and G_i^s that

$$C = S_i \oplus (C_{0,i} \oplus C_{i,n})$$

The 2^{ρ_i} codewords in S_i can be used as the representatives for the cosets in the partition $C/(C_{0,i} \oplus C_{i,n})$. Therefore, S_i is the coset representative space for the partition $C/(C_{0,i} \oplus C_{i,n})$.

- For $0 \leq i < j < n$, let $p_{i,j}(C)$ denote the linear code of length $j - i$ obtained from C by removing the first i and last $n - j$ components of each codeword in C. Every codeword in $p_{i,j}(C)$ is of the form

$$(v_i, v_{i+1}, \ldots, v_{j-1}).$$

This code is called a punctured(or truncated) code of C.

- Let $C_{i,j}^{tr}$ denote the punctured code of the subcode $C_{i,j}$; that is,

$$C_{i,j}^{tr} \triangleq p_{i,j}(C_{i,j})$$

- It follows from the structure of the TOGM G_{TOGM} that

$$k(p_{i,j}(C)) = k - k(C_{0,i}) - k(C_{j,n})$$

and

$$k(C_{i,j}^{tr}) = k(C_{i,j}).$$

Consider the punctured code $p_{0,i}$(C).

- From $k(p_{i,j}(C)) = k - k(C_{0,i}) - k(C_{j,n})$ we find that

$$k(p_{0,i}(C)) = k - k(C_{i,n}).$$

State labeling and trellis construction based on the parity-check matrix

- Consider a binary (n, k) linear block code C with a parity-check matrix

$$\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_j, \ldots, \mathbf{h}_{n-1}],$$

where, for $0 \leq j < n$, $\mathbf{h}_j$ denotes the jth column of $\mathbf{H}$ and is a binary $(n - k)$-tuple.

- A binary n-tuple $v = (v_0, v_1, \ldots, v_{n-1})$ is a codeword in C if and only if

$$v \cdot \mathbf{H}^T = (\underbrace{0, 0, \ldots, 0}_{n-k}).$$

- Let $\mathbf{0}_{n-k}$ denote the all-zero $(n-k)$-tuple $(0, 0, ..., 0)$.

  For $0 \leq i < n$, let H_i denote the submatrix that consists of the first i columns of H; that is,

$$\mathbf{H}_i = [\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{i-1}] \,.$$

- It is clear that the rank of $\mathbf{H}_i$ is at most $n-k$; that is,

$$\texttt{Rank}(\mathbf{H}_i) \leq n-k.$$

- For each codeword $c \in C_{0,i}^{tr}$,

$$c \cdot \mathbf{H}_i^T = \mathbf{0}_{n-k}.$$

  Therefore, $C_{0,i}^{tr}$ is the null space of $\mathbf{H}_i$.

- Let $L(s_0,s_i)$ denote the set of paths in the trellis T for C that connect the initial state s_0 to a state s_i in the state space $\sum_i(C)$ at time i.
- **Definition**: For $0 \leq i < n$, the label of a state $s_i \in \sum_i(C)$ based on a parity-check matrix $\mathbf{H}$ of C, denoted by $l(s_i)$, is defined as the binary $(n-k)$-tuple

$$l(s_i) \triangleq \mathbf{a} \cdot \mathbf{H}_i^T,$$

  for any $\mathbf{a} \in L(s_0, s_i)$.
  - For $i = 0$, $\mathbf{H}_i = \phi$, and the initial state s_0 is labeled with the all-zero $(n-k)$–tuple, $\mathbf{0}_{n-k}$.
  - For $i = n$, $L(s_0, s_f) = C$, and the final state s_f is also labeled with 0_{n-k}.

- For every path $(v_0, v_1, \ldots, v_{i-1}) \in L(s_0, s_i)$, the path $(v_0, v_1, \ldots, v_{i-1}, v_i)$ obtained by concatenating $(v_0, v_1, \ldots, v_{i-1})$ with the branch v_i is a path that connects the initial state s_0 to the state s_{i+1} through the state s_i.
- Hence, $(v_0, v_1, \ldots, v_{i-1}, v_i) \in L(s_0, s_{i+1})$. Then, it follows from the preceding definition of a state label that

$$\begin{aligned} l(s_{i+1}) &= (v_0, v_1, \ldots, v_{i-1}, v_i) \cdot \mathbf{H}_{i+1}^T \\ &= (v_0, v_1, \ldots, v_{i-1}) \cdot \mathbf{H}_i^T + v_i \cdot \mathbf{h}_i^T \\ &= l(s_i) + v_i \cdot \mathbf{h}_i^T \end{aligned}$$

- The foregoing expression simply says that given the label of the starting state s_i, and the output code bit v_i during the interval between time–i and time–$(i+1)$, the label of the destination state s_{i+1} is uniquely determined.

A procedure for constructing the n–section bit-level trellis diagram for a binary (n, k) linear block code C by state labeling using the parity-check matrix of the code.

The $(i+1)$-section of the code trellis is constructed by taking the following four steps:

1. Identify the special row g^* in the submatrix G_i^f and its corresponding information bit a^*. Identify the special row g^0 in the submatrix G_i^s. Form the submatrix G_{i+1}^s by including g^* in G_i^s and excluding g^0 from G_i^s.
2. Determine the set of information bits

$$A_{i+1}^s = (a_1^{(i+1)}, a_2^{(i+1)}, \ldots, a_{\rho_{i+1}}^{(i+1)})$$

   that correspond to the rows in G_{i+1}^s. Define and label the states in $\sum_{i+1}(C)$.

3. For each state $s_i \in \sum_i(C)$, form the next output v_i code bit from either $v_i = a^* + \sum_{l=1}^{\rho_i} a_l^{(i)} \cdot g_{l,i}^{(i)}$ (if there is such a row g^* in G_i^f at time-i) or $v_i = \sum_{l=1}^{\rho_i} a_l^{(i)} \cdot g_{l,i}^{(i)}$ (if there is such no row g^* in G_i^f at time-i).
4. For each possible value of v_i, connect the state s_i to the state $s_{i+1} \in \sum_{i+1}(C)$ with label

$$l(s_{i+1}) = l(s_i) + v_i \cdot h_i^T.$$

Label the connecting branch, denoted by $L(s_i, s_{i+1})$, with v_i. This completes the construction of the $(i+1)$th section of the trellis.

Repeat the preceding steps until the entire code trellis is constructed.

- **Example**:
Suppose we choose the parity-check matrix as follows:

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

  – Using this parity-check matrix for state labeling and following the foregoing trellis construction steps, we obtain the 8–section trellis with state label shown in Figure 7.

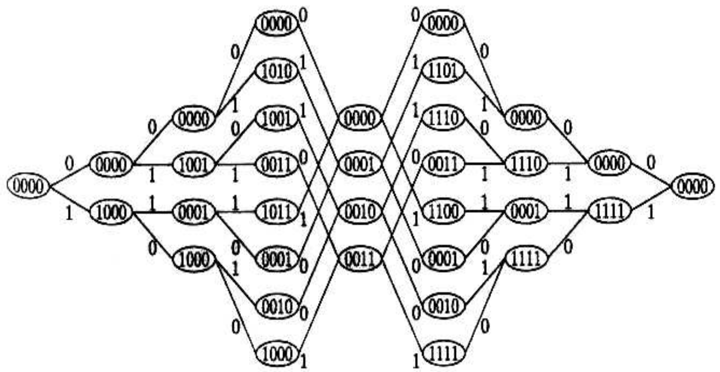


Figure 7: An 8–section trellis for the $(8, 4)$ RM code with state labeling by the parity–check matrix.

– To illustrate the construction process, we assume that the trellis has been completed up to time–3.

– At this instant, $G_3^s = \{g_0, g_1, g_2\}$ and $A_3^s = \{a_0, a_1, a_2\}$ are known. The eight states in $\sum_3(C)$ are defined by the eight combinations of a_0, a_1, and a_2.

– These eight states and their labels are given in Table 5.

	State defined by (a_0, a_1, a_2)	State label
$s_3^{(0)}$	(000)	(0000)
$s_3^{(1)}$	(001)	(1010)
$s_3^{(2)}$	(010)	(1001)
$s_3^{(3)}$	(011)	(0011)
$s_3^{(4)}$	(100)	(1011)
$s_3^{(5)}$	(101)	(0001)
$s_3^{(6)}$	(110)	(0010)
$s_3^{(7)}$	(111)	(1001)

Table 5: Labels of the states at time–3 for the $(8, 4)$ RM code based on the parity-check matrix.

– The submatrix $\mathbf{H}_4$ is

$$\mathbf{H}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

From $p_{0,4}(v_j)$, with $0 \le j \le 3$ and $\mathbf{H}_4$, we can determine the labels of the four states, $s_4^{(0)}, s_4^{(1)}, s_4^{(2)}$, and $s_4^{(3)}$, in $\sum_4(C)$, which are given in Table 6.

	State defined by (a_1, a_2)	State label
$s_4^{(0)}$	(00)	(0000)
$s_4^{(1)}$	(01)	(0001)
$s_4^{(2)}$	(10)	(0010)
$s_4^{(3)}$	(11)	(0011)

Table 6: Labels of states at time–4 for the $(8, 4)$ RM code.

– Now, suppose the encoder is in the state $s_3^{(2)}$ with label $l(s_3^{(2)}) = (1001)$ at time–3.

– Because no such row g^* exists at time $i = 3$, the output code bit v_3 is computed as follows:

$$\begin{aligned} v_3 &= a_0 \cdot g_{03} + a_1 \cdot g_{13} + a_2 \cdot g_{23} \\ &= 0 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 \\ &= 1. \end{aligned}$$

– The state $s_3^{(2)}$ is connected to the state in $\sum_4$(C) with label

$$\begin{aligned} l(s_3^{(2)}) + v_3 \cdot \mathbf{h}_3^T &= (1001) + 1 \cdot (1011) \\ &= (0010), \end{aligned}$$

which is state $s_4^{(2)}$, as shown in Figure 8.

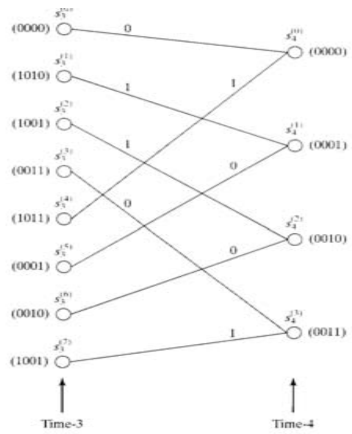


Figure 8: State labels at the two ends of the fourth section of the trellis for the $(8, 4)$ RM code.

- State labeling based on the parity-check matrix requires n-k bits to label each state of the trellis. Therefore, we show that

$$\rho_{\max}(C) \leq \min\{k, n-k\}.$$

- State labeling using $\rho_{\max}$(C) bits to label each state in the trellis is the most economical labeling method.

Trellis complexity and symmetry

- Trellis complexity is, in general, measured in terms of the state and branch complexities.
- For a binary (n, k) linear block code C, $\rho_{\max}(C)$ must satisfy the following bound:

$$\rho_{\max}(C) \leq \min\{k, n-k\}.$$

This bound was first proved by Wolf. In general, this bound is quite loose.

- For example:
  - To Consider the $(8, 4)$ RM code. For this code, $k = n - k = 4$; however, $\rho_{\max}(C) = 3$.
  - The third-order RM code RM$(3, 6)$ of length 64 is a $(64, 42)$ linear code. For this code, $\min\{k, n-k\} = 22$; however $\rho_{\max}(C) = 14$.
- We see that there is a big gap between the bound and $\rho_{\max}(C)$; however, for cyclic (or shortened cyclic) codes, the bound $\min\{k, n-k\}$ gives the exact state complexity.

- Let $C^\perp$ denote the dual code of C.
  $C^\perp$ is an $(n, n-k)$ linear block code. For $0 \le i < n$, let $\sum_i(C^\perp)$ denote the state space of $C^\perp$ at time-i.
- There is a one-to-one correspondence between the state in $\sum_i(C^\perp)$ and the cosets in the partition $p_{0,i}(C^\perp)/C_{0,i}^{\perp,tr}$, where $C_{0,i}^{\perp,tr}$ denotes the truncation of $C_{0,i}^{tr}$, in the interval $[0, i-1]$.
  Therefore, the dimension of $\sum_i(C^\perp)$ is given by

$$\rho_i(C^\perp) = k(p_{0,i}(C^\perp) - k(C_{0,i}^{\perp,tr}).$$

- Note that $p_{0,i}(C^\perp)$ is the dual code of $C_{0,i}^{tr}$, and $C_{0,i}^{\perp,tr}$ is the dual code of $p_{0,i}(C)$. Therefore

$$\begin{aligned} k(p_{0,i}(C^\perp)) &= i - k(C_{0,i}^{tr}) \\ &= i - k(C_{0,i}) \end{aligned}$$

  and $k(C_{0,i}^{\perp,tr}) = i - k(p_{0,i}(C))$.
- It follows that $\rho_i(C^\perp) = k(p_{0,i}(C^\perp) - k(C_{0,i}^{\perp,tr})$ through $k(C_{0,i}^{\perp,tr}) = i - k(p_{0,i}(C))$ that

$$\rho_i(C^\perp) = k(p_{0,i}(C)) - k(C_{0,i}).$$

  Because $k(p_{0,i}(C)) = k - k(C_{i,n})$, we have

$$\rho_i(C^\perp) = k - k(C_{0,i}) - k(C_{i,n}).$$

- From

$$\rho_i(C) = k - k(C_{0,i}) - k(C_{i,n}) \text{ and } \rho_i(C^\perp) = k - k(C_{0,i}) - k(C_{i,n}),$$

  we find that for $0 \le i \le n$,

$$\rho_i(C^\perp) = \rho_i(C).$$

  ⇒This expression says that C and its dual code $C^\perp$ have the same state complexity.
- We can analyze the state complexity of a code trellis from either the code or its dual code.
  - In general, they have different branch complexities
  - They have the same state complexity.

- **Example**:
  The dual code of this the first-order code RM code RM$(1, 4)$ of length 16. It is a $(16, 5)$ code with a minimum distance of 8. Its TOGM is

$$G_{\mathtt{TOGM}} = \begin{bmatrix} 1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&1&0&1&0&1&0&1&1&0&1&0&1&0&1&0 \\ 0&0&1&1&0&0&1&1&1&1&0&0&1&1&0&0 \\ 0&0&0&0&1&1&1&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \end{bmatrix}.$$

  - From this matrix, we find the state space dimension profile of the code,

$$(0, 1, 2, 3, 3, 4, 4, 4, 3, 4, 4, 4, 3, 3, 2, 1, 0),$$

    which is exactly the same as the state space dimension profile of the trellis od the $(16, 11)$ RM code.

- The state-defining information sets and state labels are given in Table 7.
- The 16–section trellis for the code is shown in Figure 9.
- State labeling is done based on the state-defining information sets.
- State labeling based on the parity–check matrix of the code would require 11 bits.

i	G_i^s	a^*	a^0	A_i^s	State label
0	ϕ	a_0	–	ϕ	(0000)
1	$\{g_0\}$	a_1	–	$\{a_0\}$	(a_0000)
2	$\{g_0, g_1\}$	a_2	–	$\{a_0, a_1\}$	(a_0a_100)
3	$\{g_0, g_1, g_2\}$	–	–	$\{a_0, a_1, a_2\}$	$(a_0a_1a_20)$
4	$\{g_0, g_1, g_2\}$	a_3	–	$\{a_0, a_1, a_2\}$	$(a_0a_1a_20)$
5	$\{g_0, g_1, g_2, g_3\}$	–	–	$\{a_0, a_1, a_2, a_3\}$	$(a_0a_1a_2a_3)$
6	$\{g_0, g_1, g_2, g_3\}$	–	–	$\{a_0, a_1, a_2, a_3\}$	$(a_0a_1a_2a_3)$
7	$\{g_0, g_1, g_2, g_3\}$	–	a_0	$\{a_0, a_1, a_2, a_3\}$	$(a_0a_1a_2a_3)$
8	$\{g_1, g_2, g_3\}$	a_4	–	$\{a_1, a_2, a_3\}$	$(a_1a_2a_30)$
9	$\{g_1, g_2, g_3, g_4\}$	–	–	$\{a_1, a_2, a_3, a_4\}$	$(a_1a_2a_3a_4)$
10	$\{g_1, g_2, g_3, g_4\}$	–	–	$\{a_1, a_2, a_3, a_4\}$	$(a_1a_2a_3a_4)$
11	$\{g_1, g_2, g_3, g_4\}$	–	a_3	$\{a_1, a_2, a_3, a_4\}$	$(a_1a_2a_3a_4)$
12	$\{g_1, g_2, g_4\}$	–	–	$\{a_1, a_2, a_4\}$	$(a_1a_2a_40)$
13	$\{g_1, g_2, g_4\}$	–	a_2	$\{a_1, a_2, a_4\}$	$(a_1a_2a_40)$
14	$\{g_1, g_4\}$	–	a_1	$\{a_1, a_4\}$	(a_1a_400)
15	$\{g_4\}$	–	a_4	$\{a_4\}$	(a_4000)
16	ϕ	–	–	ϕ	(0000)

Table 7: State-defining sets and labels for the 16–section trellis for $(16, 5)$ RM code.

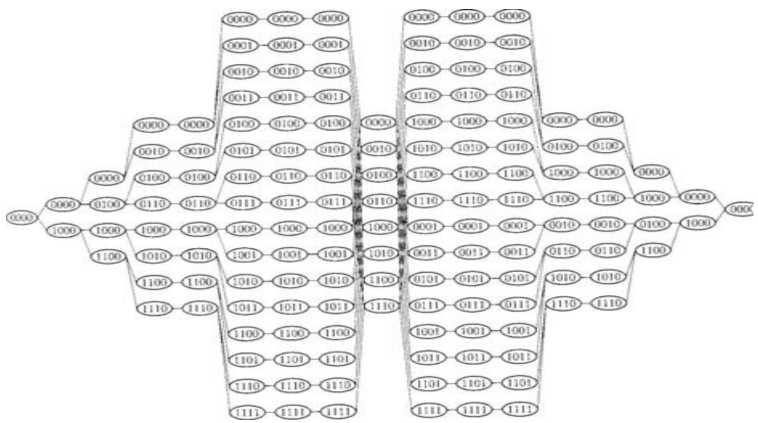


Figure 9: A 16-section trellis for $(16, 5)$ RM code with state labeling by the state-defining information set.

- Let T be an n-section trellis for an (n, k) code C with state space dimension profile $(\rho_0, \rho_1, \ldots, \rho_n)$.
- The trellis T is said to be minimal if for any other n-section trellis T' for C with state space dimension profile $(\rho'_0, \rho'_1, \ldots, \rho'_n)$ the following inequality holds:

$$\rho_i \leq \rho'_i,$$

for $0 \leq i \leq n$.

- A minimal trellis is unique within isomorphism.
- A minimal trellis results in a minimal total number of states in the trellis. In fact, the inverse is also true: a trellis with a minimum total number of states in a minimal trellis.

- From

$$\begin{aligned}\rho_i(C) &= k - \mid G_i^p \mid - \mid G_i^f \mid \\ &= k - k(C_{0,i}) - k(C_{i,n})\end{aligned}$$

we see that the state space dimension ρ_i at time-i depends on the dimensions of the past and future codes,$C_{0,i}$ and $C_{i,n}$.

$\Rightarrow$ For a given code C, $k(C_{0,i})$ and $k(C_{i,n})$ are fixed.

- Given an (n, k) linear code C, a permutation of the orders of the bit (or symbol) positions result in an equivalent code C' with the same weight distribution and the same error performance; however, different permutations of bit positions may result in different dimensions for $C_{0,i}$, $C_{i,n}$ and ρ_i at time-i.

- A permutation that yields the smallest state space dimension at every time of the code trellis is called an optimum permutation (or bit ordering).

$\Rightarrow$ Optimum permutation is hard to find.

$\Rightarrow$ Optimum permutations for RM codes are known, but they are unknown for other classes of codes.

- The branch complexity of an n-section trellis diagram for an (n, k) linear code C is defined as the total number of branches in the trellis.
- An n-section trellis diagram for an (n, k) linear block code is said to be a minimal branch (or edge) trellis diagram if it has the smallest branch complexity.
- A minimal trellis diagram has the smallest branch complexity.
- We define

$$I_i(a^*) = \begin{cases} 1, & \texttt{if } a^* \nsubseteq A_i^f \\ 2, & \texttt{if } a^* \subseteqq A_i^f \end{cases}$$

Let ε denote the total number of branches in T.

- Then

$$\begin{aligned}\varepsilon &= \sum_{i=0}^{n-1} \left| \sum_i (C) \right| \cdot I_i(a^*) \\ &= \sum_{i=0}^{n-1} 2^{\rho_i} \cdot I_i(a^*).\end{aligned}$$

For $0 \leq i < n$, $2^{\rho_i} \cdot I_i(a^*)$ is simply the number of branches in the ith section of trellis T.

- **Example**:
  To consider the TOGM of the $(8,4)$ RM code

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

  – From Table 2 we find that

$$I_0(a^*) = I_1(a^*) = I_2(a^*) = I_4(a^*) = 2$$

  and

$$I_3(a^*) = I_5(a^*) = I_6(a^*) = I_7(a^*) = 1.$$

  – The state space dimension profile of the 8–section trellis for the code is $(0,1,2,3,2,3,2,1,0)$.

– From $\varepsilon = \sum_{i=0}^{n-1} 2^{\rho_i} \cdot I_i(a^*)$ we have

$$\begin{aligned} \varepsilon &= 2^0 \cdot 2 + 2^1 \cdot 2 + 2^2 \cdot 2 + 2^3 \cdot 1 + 2^2 \cdot 2 + 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 1 \\ &= 2+4+8+8+8+8+4+2 \\ &= 44. \end{aligned}$$

- Consider an (n,k) cyclic code C over $GF(2)$ with generator polynomial

$$g(X) = 1 + g_1X + g_2X + \ldots + g_{n-k-1}X^{n-k-1} + X^{n-k}.$$

  A generator matrix for this code is given in

$$\begin{bmatrix} g_0 & g_1 & g_2 & \ldots & g_{n-k} & 0 & \ldots & \ldots & 0 \\ 0 & g_0 & g_1 & \ldots & \ldots & g_{n-k} & 0 & \ldots & 0 \\ 0 & 0 & g_0 & \ldots & \ldots & \ldots & g_{n-k} & 0\ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & g_{n-k} \end{bmatrix}.$$

- For convenience, we reproduce it here

$$\begin{aligned} G &= \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} \\ &= \begin{bmatrix} 1 & g_1 & g_2 & \ldots & \ldots & g_{n-k-1} & 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & g_1 & g_2 & \ldots & \ldots & g_{n-k-1} & 1 & 0 & \ldots & 0 \\ & & & & & \ddots & & & & & \\ 0 & 0 & \ldots & 0 & 1 & g_1 & g_2 & \ldots & \ldots & g_{n-k-1} & 1 \end{bmatrix} \end{aligned}$$

- We readily see that the generator matrix is already in trellis–oriented form.

- For $0 \le i < k$, the time span of the i-row g_i is

$$\tau(g_i) = [i, n-k+1+i].$$

(or the bit span $\phi(g_i) = [i, n-k+i]$). The active time spans of all the rows have the same length, $n-k$.

- Now, we consider the n-section bit-level trellis for this (n,k) cyclic code. There are two cases to be considered:
  1. $k > n-k$.
  2. $k \le n-k$.

- Consider the case $k > n-k$, we see that the maximum state space dimension is $\rho_{\max}(C) = n-k$, and the state space profile is

$$(0, 1, \ldots, n-k-1, n-k, \ldots, n-k, n-k-1, \ldots, 1, 0)$$

- Consider the case $k \le n-k$, we see that the maximum state space dimension is $\rho_{\max}(C) = k$, and the state space profile is

$$(0, 1, \ldots, k-1, k, \ldots, k, k-1, \ldots, 1, 0)$$

- Combining the results of the preceding two cases, we conclude that for an (n,k) cyclic code, the maximum state space dimension is

$$\rho_{\max}(C) = \min[k, n-k].$$

- This is to say that a code in cyclic form has the worst state complexity; that is, it meets the upper bound on the state complexity.

To reduce the state complexity of a cyclic code, a proper permutation on the bit position is needed.

- **Example**:
  Consider the $(7,4)$ cyclic hamming code generated by $g(x) = 1 + X + x^3$.
  Its generator matrix in TOF is

$$G = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

  - By examining the generator matrix, we find that the trellis state space dimension profile is $(0, 1, 2, 3, 3, 2, 1, 0)$. Therefore, $\rho_{\max}(C) = 3$.
  - The 7-section trellis for this code is shown in Figure 10, the state-defining information sets and the state labels are given in Table 8.

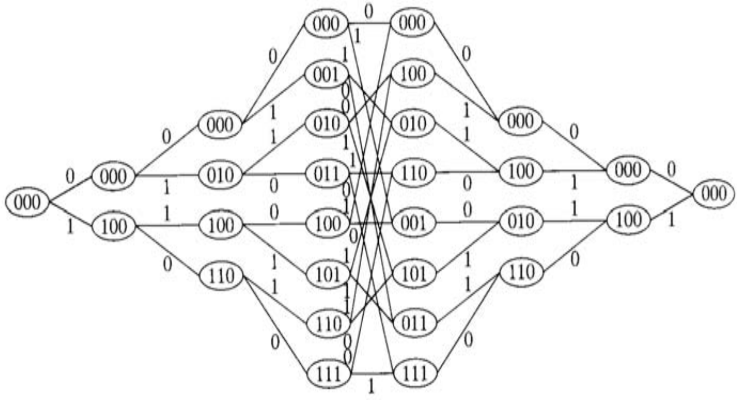


Figure 10: The 7-section trellis diagram for the $(7,4)$ Hamming code with 3-bit state label.

i	G_i^s	a^*	a^0	A_i^s	State label
0	ϕ	a_0	–	ϕ	(000)
1	$\{g_0\}$	a_1	–	$\{a_0\}$	$(a_0 00)$
2	$\{g_0, g_1\}$	a_2	–	$\{a_0, a_1\}$	$(a_0 a_1 0)$
3	$\{g_0, g_1, g_2\}$	a_3	a_0	$\{a_0, a_1, a_2\}$	$(a_0 a_1 a_2)$
4	$\{g_1, g_2, g_3\}$	–	a_1	$\{a_1, a_2, a_3\}$	$(a_1 a_2 a_3)$
5	$\{g_2, g_3\}$	–	a_2	$\{a_2, a_3\}$	$(a_2 a_3 0)$
6	$\{g_3\}$	–	a_3	$\{a_3\}$	$(a_3 00)$
7	ϕ	–	–	ϕ	(000)

Table 8: State–defining sets and state labels for the 8 section trellis for $(8,4)$ RM code.

Next, we derive a special symmetry structure for trellises of some linear block codes. This structure is quite useful for implementing a trellis-based decoder, especially in gaining decoding speed.

- Consider the TOGM of a binary (n,k) linear block code of even length n,

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix}.$$

- Suppose the TOGM G has the following symmetry property:
  - For each row g in G with bit span $\phi(g) = [a, b]$, there exists a row g' in G with bit span $\phi(g') = [n-1-b, n-1-a]$.
  - With this symmetry in G, we can readily see that for $0 \le i \le n/2$, the number of rows in G that are active at time-$(n-i)$ is equal to the number of rows in G that are active at time-i.

    This implies that

$$|\sum_{n-i}(C)| = |\sum_{i}(C)| \texttt{ or } \rho_{n-i} = \rho_i$$

for $0 \le i \le n/2$.

- We can permute the rows of G such that the resultant matrix, denoted by $G^{'}$, is in a reverse trellis-oriented form:
  1. The trailing 1 of each row appears in a column before the trailing 1 of any row below it.
  2. No two rows have their leading 1's in the same column.

- If we rotate the matrix $G^{'}$ 180° counterclockwise, we obtain a matrix $G^{"}$ in which the ith row $g_i^{"}$ is simply the $(k-1-i)$th row $g_{k-1-i}^{'}$ of $G^{'}$ in reverse order[a].
- From the foregoing, we see that $G^{"}$ and G are structurally identical in the sense that $\phi(g_i^{"}) = \phi(g_i)$ for $0 \leq i < k$.
- Consequently, the n–section trellis T for C has the following mirror symmetry:

  The last $n/2$ sections of T form the mirror image of the first $n/2$ sections of T (not including the path labels).

[a] The trailing 1 of $g_{k-1-i}^{'}$ becomes the leading 1 of $g_i^{"}$, and the leading 1 of $g_{k-1-i}^{'}$ becomes the trailing 1 of $g_i^{"}$

- **Example**:
  To consider the TOGM of the $(8,4)$ RM code

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

  – Examining the rows of $G_{\texttt{TOGM}}$, we find that $\phi(g_0) = [0, 3]$, $\phi(g_3) = [4, 7]$, and g_0 and g_3 are symmetrical with each other.
  – Row g_1 has bit span $[1, 6]$ and is symmetrical with itself. Row g_2 has bit span $[2, 5]$ and is also symmetrical with itself.

– We obtain the following matrix in reverse trellis–oriented form:

$$G' = \begin{bmatrix} g_0' \\ g_1' \\ g_2' \\ g_3' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

.

– Rotating G' 180° counterclockwise, we obtain the following matrix:

$$G" = \begin{bmatrix} g"_0 \\ g"_1 \\ g"_2 \\ g"_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

- For the case in which n is odd, if the TOGM G_{TOGM} of a binary (n, k) code C has the mirror symmetry property, then the last $(n-1)/2$ sections of the n-section trellis T for C form the mirror image of the first $(n-1)/2$ sections of T.
- The trellises of all cyclic codes have mirror-image symmetry.

Trellis sectionalization and parallel decomposition

- In a bit-level trellis diagram, every time instant in the encoding interval $\Gamma = [0, 1, 2, \ldots, n]$.
- It is possible to sectionalize a bit-level trellis with section boundary locations at selected instants in Γ.

- This sectionalization result in a trellis in which a branch may represent multiple code bits, and two adjacent states may be connected by multiple branches.
- For a positive integer $v \leq n$, let

$$\Lambda \triangleq \{t_0, t_1, t_2, \ldots, t_v\}$$

be a subset of $v + 1$ time instants in the encoding interval $\Gamma = \{0, 1, 2, \ldots, n\}$ for an (n, k) linear block code C with $0 = t_0 < t_1 < t_2 < \ldots < t_v = n$.

- A v-section trellis diagram for C with section boundaries at the locations(time instants) in Λ, denoted by $T(\Lambda)$, can be obtained from the n-section trellis T by
  1. To delete every state in Σ_t(C) for $t \in \{0, 1, ..., n\} \setminus \Lambda$ and every branch entering or leaving a deleted state.
  2. For $1 \leq j \leq v$, connecting a state $s \in \sum_{t_{j-1}}$ to a state $s \in \sum_{t_j}$ by a branch with label α if and only if there is a path with label α from state s to state s' in the n-section trellis T.

- In this v-section trellis, a branch connecting a state $s \in \sum_{t_{j-1}}$ to a state $s' \in \sum_{t_j}$ represents $(t_j - t_{j-1})$ code symbols.
- The state space $\sum_{t_{j-1}}(C)$ at time-t_{j-1}, the state space $\sum_{t_j}(C)$ at time-t_j, and all the branches between states in $\sum_{t_{j-1}}(C)$ and states in $\sum_{t_j}(C)$, form the jth section of $T(\Lambda)$.
- If the lengths of all the sections are the same, $T(\Lambda)$ is said to be uniformly sectionalized.

- If the section boundary locations $t_0, t_1, \ldots, t_v$ are chosen at the places where $\rho_{t_1}, \rho_{t_2}, \ldots, \rho_{t_{v-1}}$ are small, then the resultant v-section code trellis $T(\Lambda)$ has a small state space complexity
- However, sectionalization, in general, results in an increase in branch complexity.

It is important to properly choose the section boundary locations to provide a good trade-off between state and branch complexities.

- **Example**:
  Consider the 16–section trellis for the $(16, 11)$ RM code shown in Figure 11.
  - Suppose we choose $v = 4$ and the section boundary set $\Lambda = \{0, 4, 8, 12, 16\}$.
  - The result is a 4-section trellis as shown in Figure 12. Each section is 4 bits long.
  - The state space dimension profile for this 4–section trellis is $(0, 3, 3, 3, 0)$, and the maximum state space dimension is $\rho_{4,\max}(C) = 3$.
  - The trellis consists of two parallel and structurally identical subtrellises without cross-connections between them.

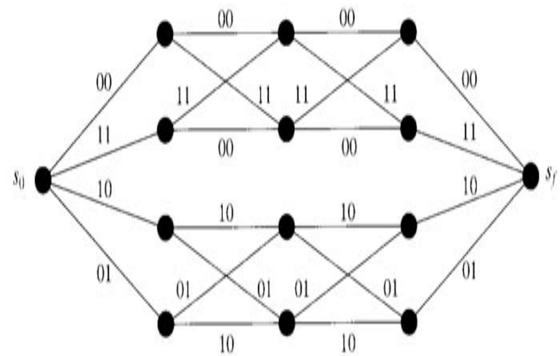


Figure 11: A 4–section trellis for the $(8, 4)$ RM code.

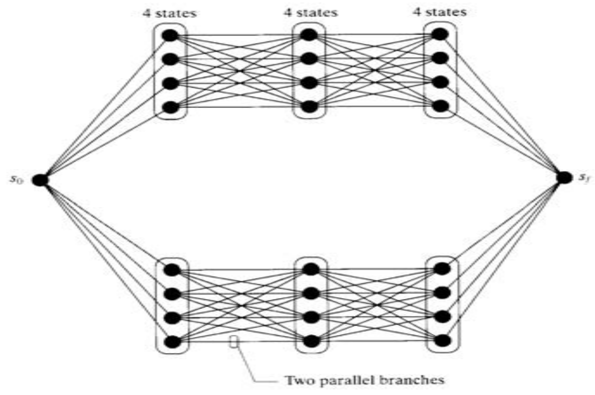


Figure 12: A 4–section minimal trellis diagram $T(\{0, 4, 8, 12, 16\})$ for the $(16, 11)$ RM code.

Parallel decomposition

- For long codes with large dimensions, it is very difficult to implement any trellis-based MLD algorithm based on a full code trellis on (an) integrated circuit(IC) chip(s) for hardware implementation.
- Parallel decomposition:

  To overcome this implementation difficulty, one possible approach is to decompose the minimal trellis diagram of a code into parallel and structurally identical subtrellises of smaller dimensions without cross connections between them so that each subtrellis with reasonable complexity can be put on a single IC chip of reasinable size.

Parallel decomposition should be done in such a way that the maximum state space dimension of the minimal trellis of a code is not exceeded.

- Advantages:
  - **Complexity**:
    Because all the subtrellis are structurally identical, we can devise identical decoders of much smaller complexity to process these subtrellises in parallel.
  - **Speed**:
    This method speeds up the decoding process.
  - **Implementation**:
    Parallel decomposition resolves wire routing problem in IC implementation and reduces internal communication.

- Suppose we choose a subcode C_1 of C by removing a row g from the $G_{\texttt{TOGM}}$ of C.

  $\Rightarrow$ The generator matrix G_1 for this subcode is $G_1 = G_{\texttt{TOGM}} \setminus \{g\}$.
- Let $\dim(C)$ and $\dim(C_1)$ denote the dimensions of C and C_1, respectively.

  $\Rightarrow \dim(C_1) = \dim(C) - 1 = k - 1$.
- The state space dimension $\rho_i(\text{C})$ is equal to the number of rows of G whose active time spans contain the time index i.

$$\Rightarrow \begin{cases} \rho_i(C_1) = \rho_i(C) - 1 & \texttt{for } i \in \tau_a(g) \\ \rho_i(C_1) = \rho_i(C) & \texttt{for } i \notin \tau_a(g) \end{cases}$$

- We define the following index set:

$$I_{\max}(C) \triangleq \{i : \rho_i(C) = \rho_{\max}(C), \ \texttt{for} \ 0 \le i \le n\}$$

- **Theorem**:
If there exists a row g in the $G_{\texttt{TOGM}}$ for an (n, k) linear code C such that $\tau_a(g) \supseteq I_{max}(C)$, then the subcode C_1 of C generated by $G_{\texttt{TOGM}} \setminus \{g\}$ has a minimal trellis T_1 with maximum state space dimension $\rho_{\max}(C_1) = \rho_{\max}(C) - 1$, and

$$I_{\max}(C_1) = I_{\max}(C) \cup \{i : \rho_i(C) = \rho_{\max}(C) - 1, \ i \notin \tau_a(g)\}.$$ [a]

[a] Theorem can be applied repeatedly until either the desired level of decomposition is achieved or no row in the generator matrix can be found to satisfy the condition in Theorem.

- Because G is in TOF, $G_1 = G \setminus \{g\}$ is also in TOF. If the condition of Theorem holds, then it follows from the foregoing that it is possible to construct a trellis for C that consists of two parallel and structurally identical subtrellises, one for C_1 and the other for its coset $C_1 \oplus g$.
- Each subtrellis is a minimal trellis and has maximum state space dimension equal to $\rho_{\max}(C)$.

- **Example**:
Consider the $(8, 4)$ RM code with TOGM

$$G_{\texttt{TOGM}} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

  - Its state space dimension profile is $(0, 1, 2, 3, 2, 3, 2, 1, 0)$ and $\rho_{\max}(C) = 3$. The index set $I_{\max}$ is $I_{\max}(C) = \{3, 5\}$.
  - By examming $G_{\texttt{TOGM}}$, we find only the second row g_1, whose active time span, $\tau_a(g_1) = [2, 6]$, contains $I_{\max}(C) = \{3, 5\}$.

- Suppose we remove g_1 from $G_{\texttt{TOGM}}$. The resulting matrix is

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

  which generates an $(8, 3)$ subcode C_1 of the $(8, 4)$ RM code C.
- From G_1 we can construct a minimal 8–section trellis T_1 for C_1, as shown in Figure 13 (the upper subtrellis).
- The state space dimension profile of T_1 is $(0, 1, 1, 2, 1, 2, 1, 1, 0)$ and $\rho_{\max}(C) = 2$. Adding $g_1 = (01011010)$ to every path in T_1, we obtain the minimal 8–section trellis T_1' for the coset $g_1 \oplus C_1$, as shown in Figure 13 (the lower subtrellis).

- The trellises T_1 and T_1' form a parallel decomposition of the minimal 8–section trellis T for the $(8,4)$ RM code. We see that the state space dimension profile of $T_1 \cup T_1'$ is $(0,2,2,3,2,3,2,2,0)$.
- Clearly, $T_1 \cup T_1'$ is not a minimal 8-section trellis for the $(8,4)$ RM code: however, its maximum state space dimension is still $\rho_{\max}(C) = 3$.
- If we sectionalize $T_1 \cup T_1'$ at locations $\Lambda = \{0,2,4,6,8\}$, we obtain the minimal 4-section trellis for the code, as shown in Figure 11.

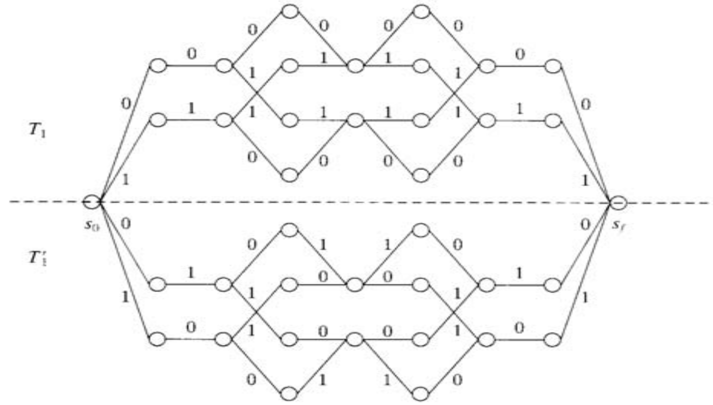


Figure 13: Parallel decomposition of the minimal 8-section trellis for the $(8,4)$ RM code.

- **Theorem**:
  Let $G_{\texttt{TOGM}}$ be the TOGM of an (n,k) linear block code C over $GF(2)$. Define the following subset of rows of $G_{\texttt{TOGM}}$:

$$R(C) \triangleq \{g \in G_{\texttt{TOGM}} : \tau_a(g) \supseteq I_{\max}(C)\}.$$

For any integer r with $1 \le r \le \mid R(C) \mid$, there exists a subcode of C_r of C such that

$$\rho_{\max}(C_r) = \rho_{\max}(C) - r \texttt{ and } \texttt{dim}(C_r) = \texttt{dim}(C) - r$$

if and only if there exists a subset $R_r \subseteq R(C)$ consisting of r rows of $R(C)$ such that for every i with $\rho_i(C) > \rho_{\max}(C_r)$, there exist at least $\rho_i(C) - \rho_{\max}(C_r)$ rows in R_r whose active time spans contains i. The subcode C_r is generated by $G_{\texttt{TOGM}} \setminus R_r$, and the set of coset representatives for C/C_r is generated by R_r.

- **Corollary**:
  In decomposing a minimal trellis diagram for a linear block code C, the maximum number of parallel isomorphic subtrellises one can obtain such that the total state space dimension at any time does not exceed $\rho_{\max}(C)$ is upper bounded by $2^{|R(C)|}$.
- **Corollary**:
  The logarithm base-2 of the number of parallel isomorphic subtrellises in a minimal v-section trellis with section boundary locations in $\Lambda = \{t_0, t_1, t_2, \ldots, t_v\}$ for a binary (n,k) linear block code C is given by the number of rows in the $G_{\texttt{TOGM}}$ whose active time spans contain the section boundary locations, t_1,t_2, ..., t_{v-1}.

- **Example**:
To consider the (16, 11) RM code. Suppose we sectionalize the bit–level trellis at locations in $\Lambda = \{0, 4, 8, 12, 16\}$. Examining the TOGM of the code, we find only row

$$g_3 = (0001111010001000)$$

whose active time span, $\tau_a(g_3) = [4, 12]$, consider $\{4, 8, 12\}$. Therefore, the 4-section trellis $T(\{0, 4, 8, 12, 16\})$ consists of two parallel isomorphic subtrellises.

Cartesian product

- Consider the interleaved code $C^\lambda = C_1 * C_2 * \ldots * C_\lambda$, which is constructed by interleaving λ linear block code, $C_1, C_2, \ldots, C_\lambda$, of length n.
- For $1 \le j \le \lambda$, let T_j be an n-section trellis for C_j. For $0 \le i \le n$, let $\sum_i(C_j)$ denote the state space of T_j at time-i.

- The Cartesian product of $T_1, T_2, \ldots, T_\lambda$, denote by $T^\lambda \triangleq T_1 \times T_2 \times \ldots \times T_\lambda$, is constructed as follows:
  1. For $0 \le i \le n$, form the Cartesian product of $\sum_i(C_1)$, $\sum_i(C_2)$, ..., $\sum_i(C_\lambda)$,

$$\begin{aligned} \textstyle\sum_i(C^\lambda) &\triangleq \textstyle\sum_i(C_1) \times \sum_i(C_2) \times \ldots \times \sum_i(C_\lambda) \\ &= \{(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(\lambda)}) : s_i^{(j)} \in \textstyle\sum_i(C_j) \texttt{ for } 1 \le j \le \lambda\}. \end{aligned}$$

Then, $\sum_i(C^\lambda)$ forms the state space of T^λ at time-i, i.e., the λ-tuple in $\sum_i(C^\lambda)$ form the nodes of T^λ at level-i.

2. A state $(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(\lambda)})$ in $\sum_i(C^\lambda)$ is adjacent to a state $(s_{i+1}^{(1)}, s_{i+1}^{(2)}, \ldots, s_{i+1}^{(\lambda)})$ in $\sum_{i+1}(C^\lambda)$ if and only if s_i^j is adjacent to s_{i+1}^j for $1 \le j \le \lambda$. Let $l_j \triangleq l(s_i^j, s_{i+1}^j)$ denote the label of the branch that connects the state s_i^j to the state s_{i+1}^j for $1 \le j \le \lambda$. We connect the state $(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(\lambda)}) \in \sum_i(C^\lambda)$ to the state $(s_{i+1}^{(1)}, s_{i+1}^{(2)}, \ldots, s_{i+1}^{(\lambda)}) \in \Sigma_{i+1}(C^\lambda)$ by a branch that is labeled by the following λ-tuple:

$$(l_1, l_2, \ldots, l_\lambda).$$

This label is simply a column of the array of

$$\begin{bmatrix} v_{1,0} & v_{1,1} & \ldots, & v_{1,n-1} \\ v_{2,0} & v_{2,1} & \ldots, & v_{2,n-1} \\ \vdots & & & \\ v_{\lambda,0} & v_{\lambda,1} & \ldots & v_{\lambda,n-1} \end{bmatrix}.$$

- The constructed Cartesian product $T^{\lambda} = T_1 \times T_2 \times \ldots \times T_{\lambda}$ is an n-section trellis for the interleaved code

$$C^{\lambda} = C_1 * C_2 * \ldots * C_{\lambda}$$

in which each section is of length λ.

- **Example**:
  Let C be the $(3, 2)$ even parity-check code whose generator matrix in trellis-oriented form is

$$G_{\mathtt{TOGM}} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

  - The 3–section bit level trellis T fot this code can easily be constructed form $G_{\mathtt{TOGM}}$ and is shown in Figure 14(a).
  - Suppose the code is interleaved to a depth of $\lambda = 2$. Then, the interleaved code $C^2 = C * C$ is a $(6, 4)$ linear code.
  - The Cartesian product $T \times T$ results in 3–section trellis T^2 for C^2, as shown in Figure 14(b).

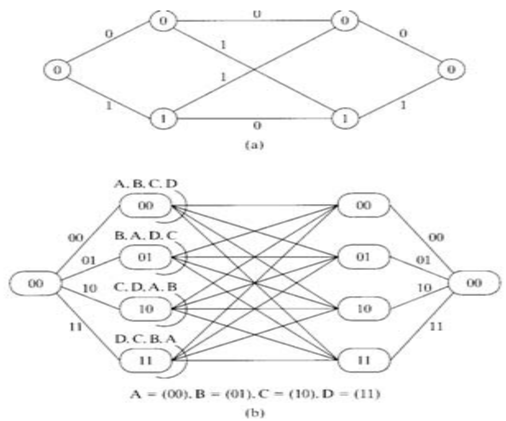


Figure 14: (a) The minimal 3–section bit level trellis for $(3, 2)$ even parity-check code, (b) A 3–section trellis for the interleaved $(3, 2)$ code of depth 2.

- Let C_1 be an (n_1, k_1) linear code, and let C_2 be an (n_2, k_2) linear code. The product $C_1 \times C_2$ is then an $(n_1 n_2, k_1 k_2)$ linear block code.
- To construct a trellis for the product code $C_1 \times C_2$, we regard the top k_2 rows of the product array shown in Figure 15 as an interleaved array with codewords from the same code C_1.
- We then construct an n_1-section trellis for the interleaved code,
$$C_1^{k_2} = \underbrace{C_1 * C_1 * \ldots * C_1}_{k_2}$$
using the Cartesian product. Each k_2–tuple branch label in the trellis is encoded into a codeword in C_2.
- The result is an n_1-section trellis for the product $C_1 \times C_2$, each section is n_2–bit in length.

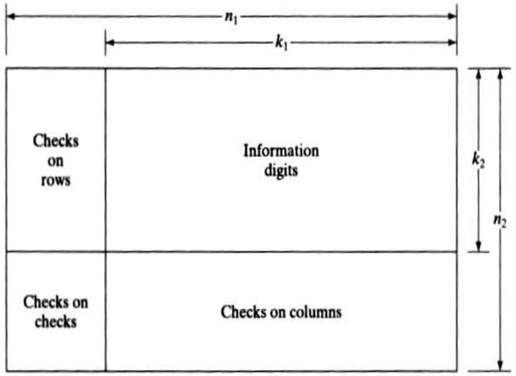


Figure 15: Code array for the product code $C_1 \times C_2$.

- **Example**:
  Let C_1 and C_2 both be the $(3, 2)$ even parity-check code. Then, the product $C_1 \times C_2$ is a $(9, 4)$ linear code with a minimum distance 4.
  - Using the Cartesian product construction method given above, we first construct the 3–section trellis for the interleaved code $C_1^2 = C_1 * C_1$, which is shown in Figure 14(b).
  - we encode each branch label in this trellis based on the $C_2 = (3, 2)$ code. The result is a 3–section trellis for the product code $C_1 \times C_2$, as shown in Figure 16.

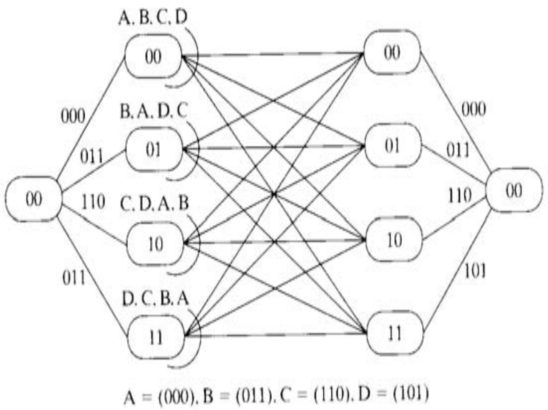


Figure 16: A 3–section trellis for the product $(3, 2) \times (3, 2)$.

- Let C_1 and C_2 be an (n, k_1, d_1) and an (n, k_2, d_2) binary linear block code with generator matrix, G_1 and G_2, respectively.
- Suppose C_1 and C_2 have only the all-zero codeword $\mathbf{0}$ in common; that is, $C_1 \cap C_2 = \{\mathbf{0}\}$.
- Their direct-sum, denoted by $C_1 \oplus C_2$, is defined as follows:

$$C \triangleq C_1 \oplus C_2 \triangleq \{u + v : u \in C_1, v \in C_2\}.$$

- Then, $C = C_1 \oplus C_2$ is an $(n, k_1 + k_2)$ code with minimum distance $d_{\min} \leq \min\{d_1, d_2\}$ and generator matrix

$$G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix}.$$

- Let T_1 and T_2 be the n-section trellis for C_1 and C_2, respectively. Then, an n-section trellis T for the direct-sum code $C = C_1 \oplus C_2$ can be constructed by taking the Cartesian product of T_1 and T_2.
- The formation of state spaces for T and the condition for state adjacency between states in T are the same as for forming the trellis for an interleaved code by taking the Cartesian product of the trellis for the component codes. The difference is branch labeling.

- For two adjacent states$(s_i^{(1)}, s_i^{(2)})$ and $(s_{i+1}^{(1)}, s_{i+1}^{(2)})$ in T at time-i and time-$(i+1)$, let $l_j \triangleq \mathrm{l}(s_i^{(j)}, s_{i+1}^{(j)})$ be the label of the branch that connects the state $s_i^{(j)}$ and the state $s_{i+1}^{(j)}$ for $1 \leq j \leq 2$.
- We connect the state $(s_i^{(1)}, s_i^{(2)})$ and the state $(s_{i+1}^{(1)}, s_{i+1}^{(2)})$ with a branch that is labeled with $l_1 + l_2 = \mathrm{l}(s_i^{(1)}, s_{i+1}^{(1)}) + \mathrm{l}(s_i^{(2)}, s_{i+1}^{(2)})$.
- The described product of two trellises is also known as the Shannon product.

- **Example**:
Let C_1 and C_2 be two linear block codes of length 8 generated by

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

and

$$G_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

respectively. It is easy to check that $C_1 \cap C_2 = \{\mathbf{0}\}$.

The direct–sum $C_1 \oplus C_2$ is generated by

$$G = \begin{bmatrix} G_1 \\ G_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix},$$

which is simply the TOGM for the $(8, 4, 4)$ RM code given in before example.

- Both G_1 and G_2 are TOF. Based on G_1 and G_2, we construct the 8–section trellises T_1 and T_2 for C_1 and C_2 as shown in Figure 17 and 18, respectively.
- Taking the Shannon product of T_1 and T_2, we obtain an 8–section trellis $T_1 \times T_2$ for the direct–sum $C_1 \oplus C_2$ as shown in Figure 19, which is simply the 8–section minimal trellis for the $(8, 4, 4)$ RM code as shown in Figure 5.

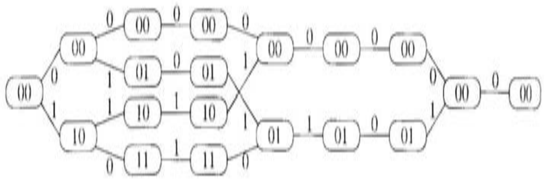


Figure 17: The 8–section minimum trellis for the code generate by $G_1 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$

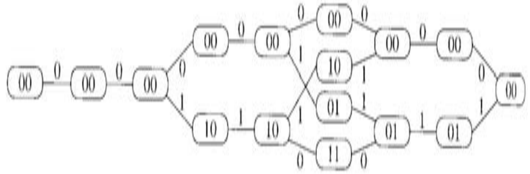


Figure 18: The 8–section minimum trellis for the code generate by $G_2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

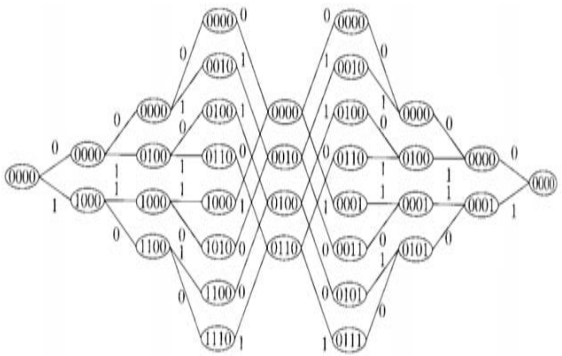


Figure 19: The Shannon product of the trellis of Figures 17 and 18.

- The Shannon product can be generalized to construct a trellis for a code that is a direct-sum of m linear block codes.
- For a positive integer $m \geqq 2$, and $1 \leqq j \leqq m$, let C_j be an (N, K_j, d_j) linear code.
- Suppose $C_1, C_2, \ldots, C_m$ satisfy the following condition: for $1 \leqq$ j, $j^{'} \leqq$ m, and j$\neq j^{'}$,
$$C_j \cap C_{j'} = \{\mathbf{0}\}.$$
- This condition simply implies that for $v_j \in C_j$ with $1 \leqq j \leqq m$,
$$v_1 + v_2 + \ldots + v_m = \mathbf{0}$$
if and only if $v_1 = v_2 = \ldots = v_m = \mathbf{0}$.

- The direct-sum of $C_1, C_2, \ldots, C_m$ is defined as

$$\begin{aligned} C &\triangleq C_1 \oplus C_2 \oplus \ldots \oplus C_m \\ &= \{v_1 + v_2 + \ldots + v_m : \; v_j \in C_j, \; 1 \le j \le m\}. \end{aligned}$$

- Then, $C = C_1 \oplus C_2 \oplus \ldots \oplus C_m$ is an (N, K, d) linear block code with

$$K = K_1 + K_2 + \ldots + K_m \text{ and } d \le \min_{1 \le j \le m} \{d_j\}.$$

- Let G_j be the generator matrix of C_j for $1 \le j \le m$. Then, $C = C_1 \oplus C_2 \oplus \ldots \oplus C_m$ is generated by the following matrix:

$$G = \begin{bmatrix} G_1 \\ G_2 \\ \vdots \\ G_m \end{bmatrix}$$

- The construction of an n-section trellis T for the direct-sum $C = C_1 \oplus C_2 \oplus \ldots \oplus C_m$ is the same as that for the direct-sum of two codes.
- Let $(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(m)})$ and $(s_{i+1}^{(1)}, s_{i+1}^{(2)}, \ldots, s_{i+1}^{(m)})$ be two adjacent states in T.
- The branch connecting $(s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(m)})$ to $(s_{i+1}^{(1)}, s_{i+1}^{(2)}, \ldots, s_{i+1}^{(m)})$ is labeled with

$$l(s_i^{(1)}, s_{i+1}^{(1)}) + l(s_i^{(2)}, s_{i+1}^{(2)}) + \ldots + l(s_i^{(m)}, s_{i+1}^{(m)}),$$

where for $1 \le j \le m$, $l(s_i^{(j)}, s_{i+1}^{(j)})$ is the label of the branch that connects the state $s_i^{(j)}$ and the state $s_{i+1}^{(j)}$ in the trellis T_j for the jth code C_j.

- **Example**:
  Again, we consider the $(8, 4, 4)$ RM code generated by the following TOGM:

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

  - For $1 \le j \le 4$, let C_j be the $(8, 1, 4)$ code generated by the jth row of G. Then, the direct–sum, $C_1 \oplus C_2 \oplus C_3 \oplus C_4$, gives the $(8, 4, 4)$ RM code.
  - The 8–section minimal trellises for the four component codes are shown in Figure 20.

- The Shannon products $T_1 \times T_2$ and $T_3 \times T_4$ generate the trellises for $C_1 \oplus C_2$ and $C_3 \oplus C_4$, respectively, as shown in figure 17 and 18.
- The Shannon product $(T_1 \times T_2) \times (T_3 \times T_4)$ results in the overall trellis for the $(8, 4, 4)$ RM code shown in Figure 19.

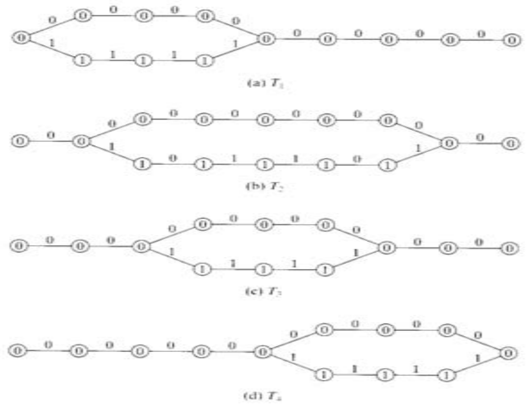


Figure 20: The 8–section minimal trellises for the four component codes.

- Let T denote the minimal n-section trellis for C, and $\rho_i(C)$ denote the state space dimension of T at time-i for $0 \leq i \leq n$. Then,
$$\rho_i(C) \leq \sum_{j=1}^{m} \rho_i(C_j).$$
- If the equality holds for $0 \leq i \leq n$, then the Shannon product $T_1 \times T_2 \times \ldots \times T_m$ is the minimal n-section trellis for the direct-sum $C = C_1 \oplus C_2 \oplus \ldots \oplus C_m$.

- For $1 \leq j \leq m$, let T_j be the minimal N-section trellis for the component code C_j.
Then, the Shannon product $T_1 \times T_2 \times \ldots \times T_m$ is the minimal N-section trellis for C if and only if the following condition holds: for $0 \leq i \leq N$, $1 \leq j, j^{'} \leq m$, and $j \neq j^{'}$,
$$p_{0,i}(C_j) \cap p_{0,i}(C_{j'}) = \{\mathbf{0}\}.$$

- **Example**:
  Suppose we sectionalize each of the two 8-section trellises of Figures 17 and 18 into 4 sections, each of length 2. The resultant 4–section trellises are shown in Figure 21, and the Shannon product of these two 4–section trellises gives a 4–section trellis, as shown in Figure 22, which is the same as the 4-section trellis for the $(8, 4, 4)$ RM code shown in Figure 11.

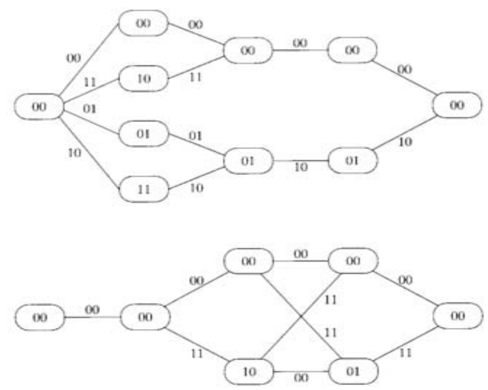


Figure 21: The 4–section trellises for the trellises of Figures 17 and 18.

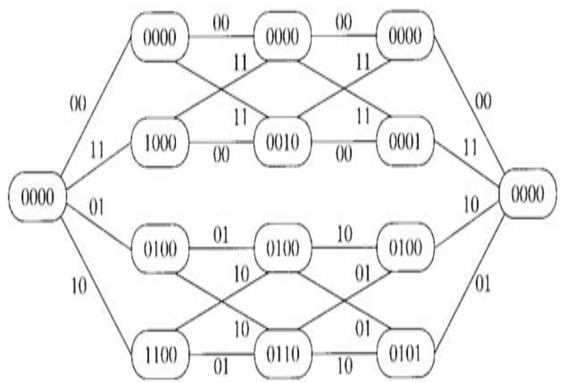


Figure 22: A 4–section trellises for the $(8, 4, 4)$ RM code.

14.1 THE VITERBI DECODING ALGORITHM

- The decoder processes the trellis from the initial state to the final state serially, section by section.
- The survivors at each level of the code trellis are extended to the next through the connecting branches between the two levels.
- The paths that enter a state at the next level are compared, and the most probable path is chosen as the survivor.
- This process continues until the end of the trellis is reached.

- The number of computations required to decode a received sequence can be enumerated easily.
- In a bit-level trellis, each branch represents one code bit, and hence a branch metric is simply a bit metric.
- Let N_a denote the total number of additions required to process the trellis T. Then,

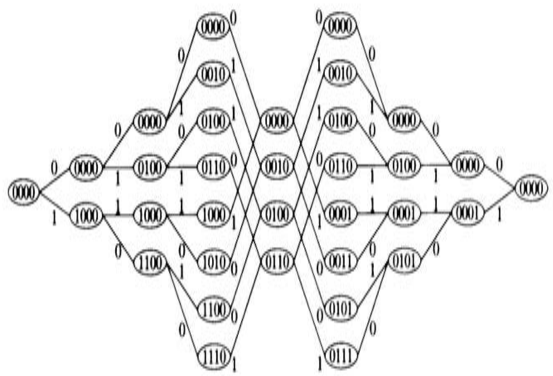
$$N_a = \sum_{i=0}^{n-1} 2^{\rho_i}.I_i(a^*) \quad (1)$$

where 2^{ρ_i} is number of states at the ith level of the code trellis T, a^* is the current input information bit at time-i, and
$I_i(a^*) = 1, \quad$ if $a^* \not\subseteq A_i^f$
$I_i(a^*) = 2, \quad$ if $a^* \subseteq A_i^f$

- If there is an oldest information bit a^0 to be shifted out from the encoder memory at time-i, then there are two branches entering each state s_{i+1} of the trellis at time-$i+1$.
- Define
$J_i(a^0) = 0, \quad$ if $a^0 \not\subseteq A_i^s$
$J_i(a^0) = 1, \quad$ if $a^0 \subseteq A_i^s$,
where A_i^s is the state-defining information set at time-i.
- Let N_c denote the total number of comparisons required to determine the survivors in the decoding process. Then,

$$N_c = \sum_{i=0}^{n-1} 2^{\rho_{i+1}}.J_i(a^0). \quad (2)$$

- The total number of computations(additions and comparisons) required to decode a received sequence based on the bit-level trellis T using the Viterbi decoding algorithm is $N_a + N_c$.



- The total number of computations(additions and comparisons) required to process a sectionalized trellis T(Λ) depends on the choice of the section boundary location set $\Lambda = [t_0, t_1, ..., t_v]$.
- A sectionalization of a code trellis that given the smallest total number of computations is called an "optimal sectionalization" for the code.

- For any two integers x and y with 0 ≤ x < y ≤ n, the section from time-x to time-y in any sectionalized trellis T(Λ) with x,y ∈ Λ and x+1, x+2, ..., y-1 ⊈ Λ is identical.
- Let φ(x,y) denote the number of computations of the Viterbi decoding algorithm to process the trellis section from time-x to time-y. Let φ_{min}(x,y) denote the smallest number of computations of the Viterbi decoding algorithm to process the trellis section from time-x to time-y.

- It follows from the definitions of φ(x,y) and φ_{min}(x,y) that

$$\varphi_{min}(0,y) = \begin{cases} min\{\varphi(0,y), min_{\{0<x<y\}}\{\varphi_{min}(0,x)+\varphi(x,y)\}\}, \\ for 1 < y \leq n \\ \varphi(0,1), \\ for y = 1. \end{cases}$$

For every $y \in$ {1, 2, ...,n}, φ_{min}(0,y) can be computed as follows.

- Example: Consider the second-order RM code of length 64 that is a (64,22) code with minimum Hamming distance 16. We find that the boundary location ser Λ = {0, 8, 16, 32, 48, 56, 61, 63, 64} results in an optimum sectionalization. With this optimum sectionalization, φ_{min}(0,64) is 101786.
With the section boundary location set Λ = {0, 8, 16, 24, 32, 40, 48, 56, 64}, requires a total of 119935 computations.

14.4.1 The MAP Algorithm Based on a Bit-level Trellis

- Assume BPSK transmission. Let v = (v_0, v_1, ..., v_{n-1}) be a codeword and c = (c_0, c_1, ..., c_{n-1}) be its corresponding bipolar signal sequence, where for $0 \leq i < n, c_i = 2v_i - 1 = \pm 1$. Let r = ($r_0$, r_1, ..., r_{n-1}) be the soft-decision received sequence.
- Log-likelihood ratio(LLR), which is defined as

$$L(v_i) \triangleq \log \frac{p(v_i = 1|r)}{p(v_i = 0|r)}, \quad (3)$$

where $p(v_i|r)$ is the a posteriori probability of v_i given the received sequence r.

- The estimated code bit v_i is then given by the sign of its LLR as follows:

$$v_i = \begin{cases} 1, & if L(v_i) > 0, \\ 0, & if L(v_i) \leq 0. \end{cases} \quad (4)$$

the larger the $|L(v_i)|$, the more reliable the hard decision of v_i. Therefore, $L(v_i)$ represents the soft information associated with the decision on v_i.

- In the n-section bit-level trellis T for C, let B_i(C) denote the set of all branches (s_i, s_{i+1}) that connect the state in state space Σ_i(C) at time-i and the states in state space Σ_{i+1}(C) at time-$(i+1)$.
- Let B_i^0(C) and B_i^1(C) denote the two disjoint subsets of B_i(C) that correspond to code v_i=0 and v_i=1, respectively. Clearly,

$$B_i(C) = B_i^0(C) \cup B_i^1(C) \quad (5)$$

for $0 \leq i < n$. Based on the structure of linear codes, $|B_i^0(C)| = |B_i^1(C)|$.

- For $(s^{'}, s) \in B_i(C)$, we define the joint probabilities

$$\lambda_i(s^{'}, s) \triangleq p(s_i = s^{'}, s_{i+1} = s, r) \quad (6)$$

for $0 \leq i < n$. Then, the joint probabilities $p(v_i, r)$ for $v_i = 0$ and $v_i = 1$ are given by

$$p(v_i = 0, r) = \sum_{(s^{'}, s) \in B_i^0(C)} \lambda_i(s^{'}, s), \quad (7)$$

$$p(v_i = 1, r) = \sum_{(s^{'}, s) \in B_i^1(C)} \lambda_i(s^{'}, s). \quad (8)$$

- In fact, the LLR of v_i can be computed directly from the joint probabilities $p(v_i = 0, r)$ and $p(v_i = 1, r)$, as follows:

$$L(v_i) \triangleq \log \frac{p(v_i = 1, r)}{p(v_i = 0, r)}. \tag{9}$$

for $0 \leq j \leq l \leq n$. let $r_{j,l}$ denote the following section of the received sequence r:

$$r_{j,l} \triangleq (r_j, r_{j+1}, ..., r_{l-1}) \tag{10}$$

- For any state $s \in \sum_i(C)$, we define the probabilities

$$\alpha_i(s) \triangleq p(s_i = s, r_{0,i}) \tag{11}$$

$$\beta_i(s) \triangleq p(r_{i,n}|s_i = s). \tag{12}$$

- It follows from the definitions of $\alpha_i(s)$ and $\beta_i(s)$ that

$$\alpha_0(s_0) = \beta_n(s_f) = 1. \tag{13}$$

For any two adjacent states $s^{'}$ and s with $s^{'} \in \sum_{i+1}(\mathrm{C})$, we define the probability

$$\begin{aligned} \gamma_i(s^{'}, s) &\triangleq p(s_{i+1} = s, r_i|s_i = s^{'}) \\ &= p(s_{i+1} = s|s_i = s^{'})p(r_i|(s_i, s_{i+1}) = (s^{'}, s)). \end{aligned} \tag{14}$$

For a memoryless channel, it follows from the definitions of $\lambda_i(s', s), \alpha_i(s), \beta_i(s)$, and $\gamma_i(s^{'}, s)$ that for $0 \leq i < n$,

$$\lambda_i(s', s) = \alpha_i(s^{'})\gamma_i(s^{'}, s)\beta_{i+1}(s). \tag{15}$$

- Then, it follows from (7), (8), and (15) that we can express (9) as follows:

$$L(v_i) \triangleq \log \frac{\sum_{(s^{'},s)\in B_i^1(C)} \alpha_i(s^{'})\gamma_i(s^{'}, s)\beta_{i+1}(s)}{\sum_{(s^{'},s)\in B_i^0(C)} \alpha_i(s^{'})\gamma_i(s^{'}, s)\beta_{i+1}(s)}. \tag{16}$$

- For a state $s \in \sum_i(C)$, let $\Omega_{i-1}^{(c)}(s)$ and $\Omega_{i+1}^{(d)}(s)$ denote the sets of states in $\sum_{i-1}(C)$ and in $\sum_{i+1}(C)$, respectively, that are adjacent to s, as shown in Figure 14.9.

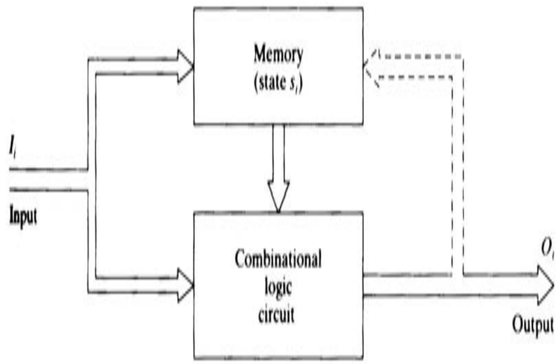
- Then $\alpha_i(s)$ and $\beta_i(s)$ can be expressed as follows:

1. For $0 \leq i \leq n$,

$$\begin{aligned} \alpha_i(s) &= \sum_{s^{'}\in\Omega_{i-1}^{(c)}(s)} p(s_{i-1} = s^{'}, s_i = s, r_{i-1}, r_{0,i-1}) \\ &= \sum_{s^{'}\in\Omega_{i-1}^{(c)}(s)} \alpha_{i-1}(s^{'})\gamma_{i-1}(s^{'}, s); \end{aligned} \tag{17}$$

2. For $0 \leq i \leq n$,

$$\begin{aligned} \beta_i(s) &= \sum_{s^{'}\in\Omega_{i+1}^{(d)}(s)} p(r_i, r_{i-1,n}, s_{i+1} = s^{'}|s_i = s) \\ &= \sum_{s^{'}\in\Omega_{i+1}^{(d)}(s)} \gamma_i(s, s^{'})\beta_{i+1}(s^{'}). \end{aligned} \tag{18}$$



- The state transition probability $\gamma_i(s^{'}, s)$ depends on the probability distribution of the information bits and the channel. Assume that each information bit is equally likely to be 0 or 1. Then, all the states in $\sum_i$(C) are equiprobable, and the transition probability

$$p(s_{i+1} = s | s_i = s^{'}) = \frac{1}{\Omega_{i+1}^{(d)}(s')} \quad (19)$$

For an AWGN channel with zero mean and two-sided power spectral density $\frac{N_0}{2}$, the conditional probability

$$p(r_i|(s_i, s_{i+1}) = (s^{'}, s)) = \frac{1}{\sqrt{\pi N_0}} exp\{\frac{-(r_i - c_i)^2}{N_0}\}, \quad (20)$$

where $c_i = 2v_i$ - 1, and v_i is the code bit on the branch($s^{'}$, s).

- We can use

$$w_i(s^{'}, s) \triangleq exp\{\frac{-(r_i - c_i)^2}{N_0}\} \quad (21)$$

to replace $\gamma_i(s^{'}, s)$ in computing $\alpha_i(s), \beta_i(s), \lambda_i(s^{'}, s), p(v_i = 1, r)$, and $p(v_i = 0, r)$. We call $w_i(s^{'}, s)$ the *weight* of the branch ($s^{'}$,s).

- Consequently, For each state $s \in \sum_{i+1}(C)$ compute and store

$$\alpha_{i+1}(s) = \sum_{s^{'} \in \Omega_i^{(c)}(s)} \alpha_i(s^{'})\omega_i(s^{'}, s). \quad (22)$$

$$\beta_i(s) = \sum_{s^{'} \in \Omega_{i+1}^{(d)}(s)} \omega_i(s, s^{'})\beta_{i+1}(s^{'}). \quad (23)$$

- Then, to carry out the MAP decoding algorithm, use the following three steps:
1. Perform the forward recursion process to compute the forward state probabilities, $\alpha_i(s)$'s, for $0 \leq i \leq n$.
2. Perform the backward recursion process to compute the backward state probabilities, $\beta_i(s)$'s, for $0 \leq i \leq n$.
3. Decoding is also performed during the backward recursion. As soon as the probabilities $\beta_{i+1}(s)$'s for all state $s \in \sum_{i+1}(C)$ have been computed, evaluate the probabilities $\lambda_i(s^{'}, s)$'s for all branches $(s^{'}, s) \in B_i(C)$.

- From (7) and (8) compute the joint probabilities $p(v_i = 0, r)$ and $p(v_i = 1, r)$. Then, compute the LLR of v_i from (9) and decode v_i based on the decision rule given by (4).

Factor Graphs

Coding and Communication Laboratory

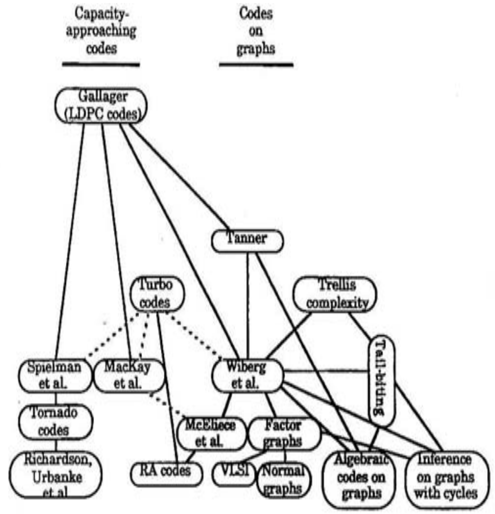
Dept. of Electrical Engineering,
National Chung Hsing University

- **Chapter 9: Factor Graphs**
  1. Introduction
  2. Factor graphs
  3. An example
  4. Sum product algorithm
  5. Code realization: behavior and probability modeling
  6. Trellis decoding for trellis-based realization
  7. Iterative decoding for LDPC, turbo, and RA codes

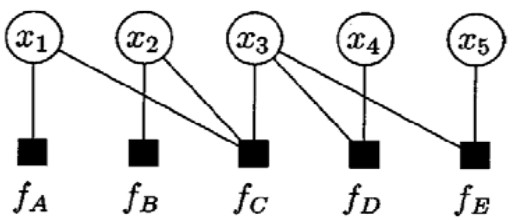
Reference

1. **Kschischang, Factor graphs and sum-product algorithm**
2. **Kschischang, Codes defined on graphs**
3. Wiberg, Codes and iterative decoding on general graphs
4. Wiberg, Codes and decoding on general graphs
5. **Forney, Codes and graphs: normal realizations**
6. Forney, Codes on Graphs: News and Views
7. Tanner, A recursive approach to low complexity codes
8. Loeliger, An introduction to factor graphs
9. Schlegel, Trellis and turbo coding: chapter 8

Introduction

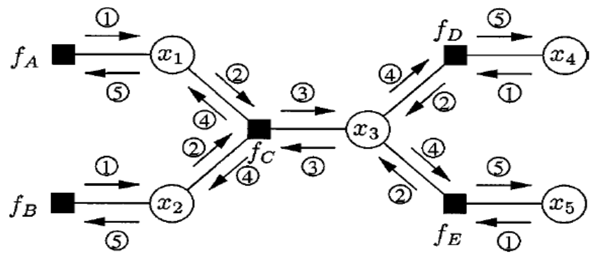


1. A **factor graph** is a graphical representation of any function, in particular for a complicated global function which is a product of some simple local functions.
   In other words, a factor graph expresses how a global function of many variables factors into a product of local functions of a subset of some variables.



$f(x_1, x_2, x_3, x_4, x_5)$
$= f_A(x_1) \cdot f_B(x_2) \cdot f_c(x_1, x_2, x_3) \cdot f_D(x_3, x_4) \cdot f_E(x_3, x_5).$

2. The **sum product algorithm** operated on the factor graph attempts to computer (1) exact marginal functions associated with the global function when the factor graph is cycle free or (2) almost exact marginal function with low complexity iterative processing when the factor graph has cycles.



3. For coding, **factor graphs** provide a nice graphical description of any realization of codes and the **sum product** algorithms represent the optimal or near optimal decoding of codes.
4. For any linear code C (block or convolutional codes), we can associate C with at least two realizations and its corresponding factor graphs:
   - one based on the parity check matrix of C, (check-based realization: Tanner factor graph with cycles)
   - the other based on the trellis of C. (trellis-based realization: Wiberg factor graph without cycles).
5. We can then apply the sum product decoding algorithms on these two factor graphs.

check realization and its factor graph

The binary linear code $C[6,3,3] = \{x \in F_2^6 : H \cdot x = 0\}$ with

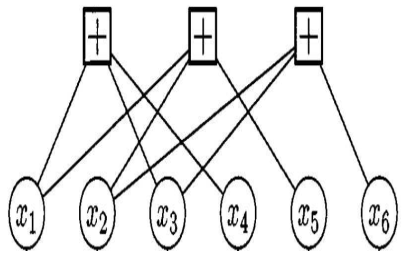
$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$[(x_1, x_2, \ldots, x_6) \in C]$ (a complicated function)

$= [x_1 \oplus x_3 \oplus x_4 = 0] \cdot [x_1 \oplus x_2 \oplus x_5 = 0] \cdot [x_2 \oplus x_3 \oplus x_6 = 0]$

$[(x_1, x_2, \ldots, x_6) \in C]$ (a complicated function)

$= [x_1 \oplus x_3 \oplus x_4 = 0] \cdot [x_1 \oplus x_2 \oplus x_5 = 0] \cdot [x_2 \oplus x_3 \oplus x_6 = 0]$



This factor graph is also called a Tanner graph.

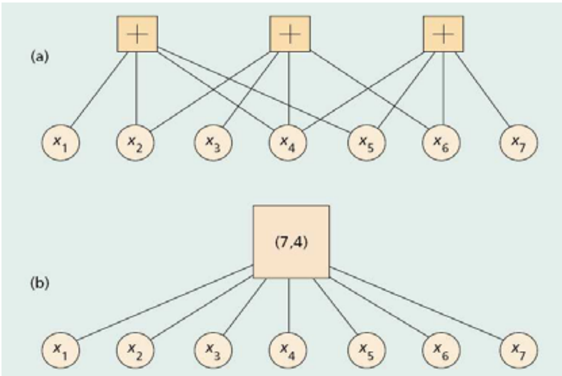
The factor graph for a given code based on a $r \times n$ H has r check nodes and n symbol nodes.

For a binary code, we have simple parity check nodes and binary-valued symbol nodes (bit nodes).

The factor graph is not unique since a linear code has many parity check matrices associated with it.

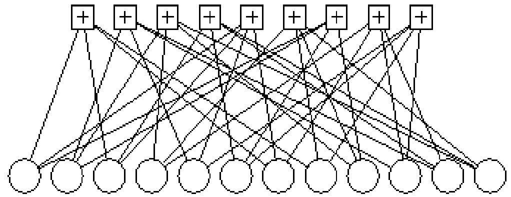
The factor graph with parity check matrices for $C[6,3,3]$ is in general not cycle free and supports suboptimal iterative decoding with low complexity.

We can combined these three **simple** check nodes into a single complex global $(6,3)$ check node whose factor graph is now cycle free and support optimal noniterative decoding with high complexity.



- (a) A factor graph with cycles: three simple $(4, 3, 2)$ single parity check with iterative decoding
- (b) A cycle-free factor graph: a complicated global $(7, 4, 3)$ Hamming check with optimum decoding

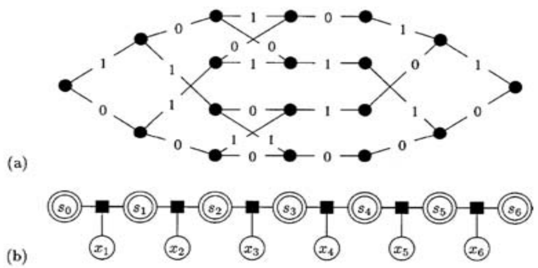
A $r \times n$ parity check matrix with constant $\rho \ll n$ row weights and constant $\gamma \ll r$ column weights, i.e., a factor graph in which every check node has degree ρ and every symbol node has degree γ, is called a regular (γ, ρ) LDPC code.



A $(3, 4)$ regular LDPC code with $n = 12$ and $r = 9$.

trellis realization and its factor graph

The same $C[6, 3, 3]$ linear code with trellis and factor graph:



This factor graph is also called a Wiberg graph.

$[(x_1, x_2, \ldots, x_6) \in C, (s_0, s_1, \ldots, s_6) \in S] = [(s_0, x_1, s_1) \in T_1] \cdot$
$[(s_1, x_2, s_2) \in T_2] \cdot [(s_2, x_3, s_3) \in T_3] \cdot [(s_3, x_4, s_4) \in T_4] \cdot$
$[(s_4, x_5, s_5) \in T_5] \cdot [(s_5, x_6, s_6) \in T_6] \cdot [(s_6, x_7, s_7) \in T_7]$
E.g., $T_2 = \{(0, 0, 00), (0, 1, 10), (1, 0, 11), (1, 1, 01)\}$

Besides the n symbol nodes and n trellis-section check nodes, we also have $n + 1$ state nodes.

The state nodes are not binary-valued. For a good code, the number of states is prohibitively large.

The factor graph based on the trellis of C is cycle free but also not unique since there are many trellises realizations of a code even though the ordering of the codeword bits is fixed.

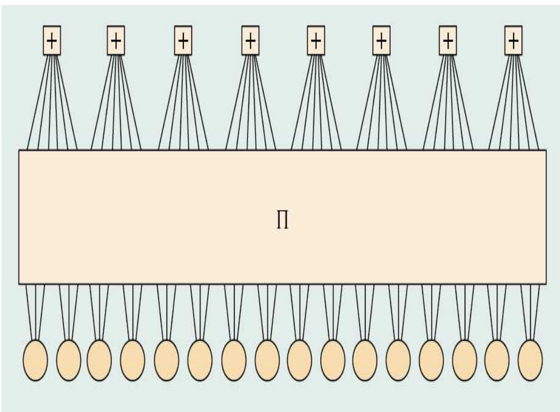
Since the minimal trellis of C always exists for fixed ordering, the corresponding factor graph for the minimal trellis is unique.

The problem of finding an optimal minimal trellis among all permutations of bit positions is extremely difficult. Right now, only Reed Muller codes of natural ordering with optimal minimal trellis have been found.

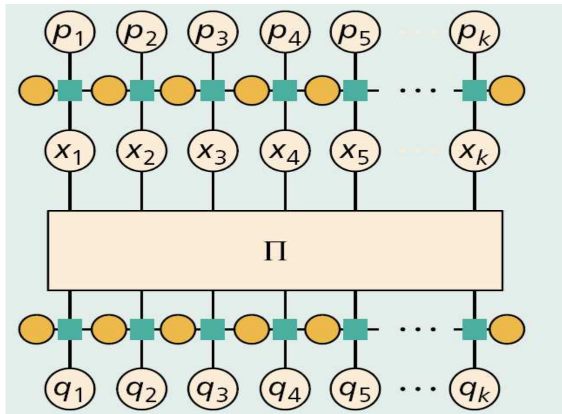
We can also consider the tailbiting trellises for C whose factor graph is with one cycle.

Finding a good tailbiting trellis for C is a very active research.

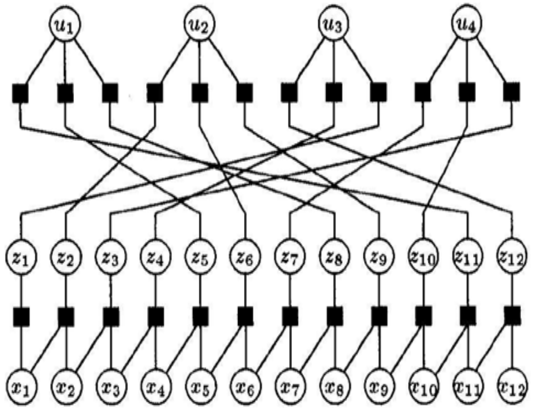
6. Many Shannon capacity approaching codes, such as turbo codes, LDPC codes, RA codes, are all well understood as codes defined on **factor graph**:
   (a) Turbo codes (1993): Berrou
   (b) LDPC codes (1963): Gallager
   (c) RA codes (1998): McEliece



A factor graph of LDPC codes with random permutation Π



A factor graph of turbo codes with random permutation Π



A factor graph of RA codes with random permutation

7. Many decoding algorithms, such as BCJR, SOVA, Viterbi, and iterative decoding for turbo codes and LDPC codes, can all be viewed as the specific instances of **sum product** algorithm.
8. Beside coding, many other known algorithms in AI and signal processing can also be viewed as instances of sum product algorithm that operates by passing messages in the factor graph.

9. Factor graphs are bipartite graphs:
   The nodes (vertices) of factor graphs are divided into two groups such that there are no edges (branches) connections inside each group, but only edges connected between nodes in different groups.
10. In the case of factor graphs for coding, these groups are
    (a) variable nodes (symbol nodes, visible nodes, bit nodes)
    (b) auxiliary nodes (state nodes, hidden nodes, latent nodes)
    (c) function nodes (constraints nodes, factor nodes, check nodes)
    Each node can be thought of a processor doing the calculation of the messages and each edge is a channel of passing the messages in two directions.

Algebraic vs. Graphic coding

- Algebraic coding $\Longrightarrow$ coding theory
- Graphic coding $\Longrightarrow$ coding practice
- Algebraic coding and Graphic coding are two important disciplines in error correcting codes.
- Coding theory and coding practice are indispensable subject in communication engineering.
- More connection and twists between algebraic coding (coding theory) and graphic coding (coding practice) are expected and need to be explored.

History

- In 1981, Tanner introduced bipartite graphs to describe any linear block code with $r \times n$ parity check matrix H, in particular for regular low density parity check codes with constant column and row weight.
- The bipartite graph of codes has n codeword bit nodes as one group and r parity check nodes as the other group, representing parity check constraints. The edge is connected from the i check node to the j bit node iff $h_{i,j} = 1$.
- Tanner also presented the fundamentals of two iterative decoding on graphs, i.e., the min-sum and sum-product algorithms.

- In 1995, Wiberg et al. introduced analogous graphs for any codes with trellis structure by adding hidden state variables to Tanner's graphs.
- In 1999, Aji and McEliece present an equivalent definition: generalized distributive law (GDL).
- In 2001, Kschischang et al. introduced factor graph notations.
- In 2001, Forney introduced the normal graphs.

- The artificial intelligence community also developed graphical methods of solving probabilistic inference problems, termed probability propagation.
- Pearl presented an algorithm in 1986 under the name **belief propagation** in Bayesian Networks, for use in acyclic or cycle-free graphs.
- **Fact:** many algorithms used in digital communications and signal processing are all special instances of a more general message-passing algorithm, the sum-product algorithm, operating on factor graphs.

Factor vs. Normal graphs

- A realization of a code/system is a mathematical characterization of the code/system.
- Given a code, we can have at least two different realizations of the code, i.e., the check realization and the trellis realization.
- For each realization, we can associate it with different graphs, e.g., the factor graphs and the normal graphs.
- For coding, factor graphs by Kschischang and normal graphs by Forney are two different graphical representations of any realization of a code C.

- The normal graph replaces the symbol nodes by an edge of degree 1 and the state nodes by an edge of degree 2.
- We can easily transform a factor graph into a normal graph and vice versa.
- For a beginner, we suggest him to understand the factor graphs first.
- For an expert, you might learn the factor and normal graphs simultaneously.
- It seems that the normal graphs is more natural than factor graph in coding society.

Factor graphs

Factor graphs

- Let $x = (x_1, x_2, \ldots, x_n)$ be a collection of variables.
  - Each x_i takes on values in A_i, $1 \le i \le n$, usually $|A_i| < \infty$.
  - x_i could be visible or hidden nodes.
  - For check-based factor graph, $A_i = F_2$, $1 \le i \le n$.
  - For trellis-based factor graph, besides $A_i = F_2$ for visible bit node x_i, we could have a state space $A_s = F_2^{k_s}$, where k_s is the state dimension of the s-th state space for hidden state node x_s.

- Let $g(x) = g(x_1, \ldots, x_n)$ be a real-valued function of these variables.
  - I.e., $g(x)$ a function with domain

$$S = A_1 \times A_2 \times \ldots \times A_n$$

    and codomain R.
  - The domain S of g is also called the **configuration space** and each element of S is a particular configuration of g.
  - We can compute n marginal functions $g_i(x)$, i.e., for each $a \in A_i$ the value of $g_i(a)$ is obtained by summing the value of global function $g(x_1, \ldots, x_n)$ over all configurations of the variables which have $x_i = a$.

- Let g be a function of three variables x_1, x_2, and x_3, then the "summary for x_2" is denoted by

$$\sum_{\sim\{x_2\}} g(x_1, x_2, x_3) := \sum_{x_1 \in A_1} \sum_{x_3 \in A_3} g(x_1, x_2, x_3).$$

- Therefore, the i-th marginal function is denoted by

$$g_i(x_i) := \sum_{\sim\{x_i\}} g(x_1, \ldots, x_n).$$

- Assuming that $g(x)$ can be factored into some local functions and with the help of distributive law of product operation over sum operation, we are seeking an efficient algorithm operate on the factor graph of $g(x)$ to compute $g_i(x_i)$.

- A complicated global function $g(x_1, \ldots, x_n)$ can be factored into a product of several simple local functions, each having some subset of $\{x_1, \ldots, x_n\}$ as arguments:

$$g(x_1, \ldots, x_n) = \prod_{j \in J} f_j(X_j)$$

where X_j is a subset of $\{x_1, \ldots, x_n\}$, and $f_j(X_j)$ is the j-th function having the elements of X_j as arguments.

Definition. A factor graph is a bipartite graph that expresses the structure of the factorization $g(x_1, \ldots, x_n) = \prod_{j \in J} f_j(X_j)$.

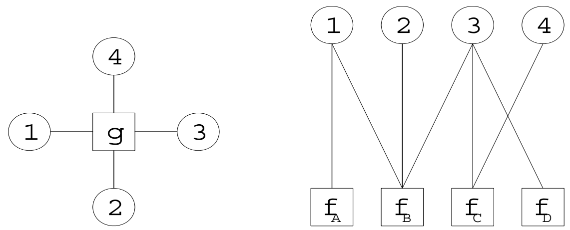
A factor graph has a variable node for each variable x_i, a factor node for each local function f_j, and an edge-connecting variable node x_i to factor node f_j if and only if x_i is an argument of f_j.

- More precisely, given a global function $g(x) = g(x_1, \ldots, x_n)$ factored into some local functions $f_E(x_E)$, $E \in Q$

$$g(x_1, \ldots, x_n) = \prod_{E \in Q} f_E(x_E),$$

a factor graph of $g(x)$ is a bipartite graph with vertex set $S \cup Q$ and edge set $\{\{i, E\} : i \in S, E \in Q, i \in E\}$, where S={variable nodes} and Q={function nodes}.

Example: $g(x_1, x_2, x_3, x_4) = f_A(x_1) f_B(x_1, x_2, x_3) f_C(x_3, x_4) f_D(x_3)$



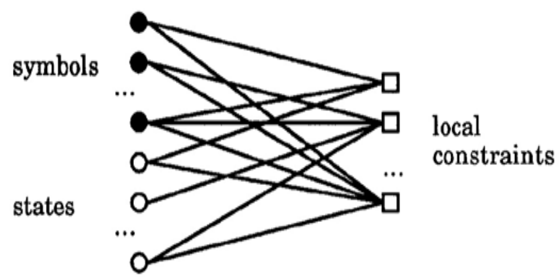
$S = \{x_1, x_2, x_3, x_4\}$ and $Q = \{f_A, f_B, f_C, f_D\}$

Notations

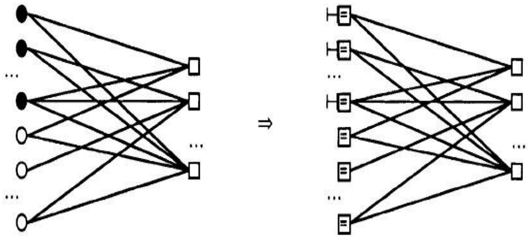
- In coding, it is preferred to separate the variable nodes into symbol and state nodes and call the function nodes as constraint or check nodes.
- In other words, we have three nodes: symbol, state, and check nodes in coding application.
- The edge is connected from the i-th check node to the k-th symbol node or the j-th state node if the i-th constraint node is involved with the k-th symbol and the j-th state nodes.

- Then the extended code (full behavior) is the set of all symbol/state configurations satisfying all local constraints.
- The code (partial behavior) is the symbol configurations (codewords) that appears as part of symbol/state configurations in the extended code.

- In factor graphs, symbol, state, and check nodes are three different nodes and can be thought as processors doing the calculation of the message.



- In normal graphs, we only have check nodes which purely do the computation; symbol edges are purely for input/output and state edges are purely for message passing.



An example

Example 1 (A Simple Factor Graph)

Let $g(x)$ be a function of five variables, and suppose that $g(x)$ can be expressed as a product

$$g(x_1, x_2, x_3, x_4, x_5) = f_A(x_1) f_B(x_2) f_C(x_1, x_2, x_3) \cdot f_D(x_3, x_4) f_E(x_3, x_5),$$

where $J = \{A, B, C, D, E\}$, $X_A = \{x_1\}$, $X_B = \{x_2\}$, $X_C = \{x_1, x_2, x_3\}$, $X_D = \{x_3, x_4\}$, and $X_E = \{x_3, x_5\}$.

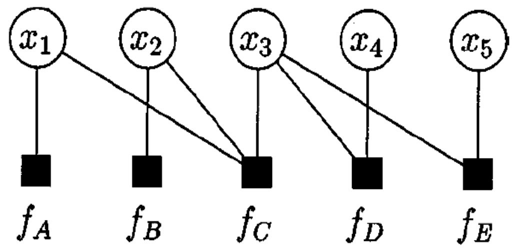


Figure 1: A factor graph for $g(x) = f_A(x_1) \cdot f_B(x_2) \cdot f_C(x_1, x_2, x_3) \cdot f_D(x_3, x_4) \cdot f_E(x_3, x_5)$.

Expression trees

- Now, we can compute the first marginal function $g_1(x_1)$ as

$$g_1(x_1) = f_A(x_1)\left(\sum_{x_2} f_B(x_2)\left(\sum_{x_3} f_C(x_1,x_2,x_3)\right.\right.$$
$$\left.\left.\cdot\left(\sum_{x_4} f_D(x_3,x_4)\right)\left(\sum_{x_5} f_E(x_3,x_5)\right)\right)\right)$$

or, in summary notation

$$g_1(x_1) = f_A(x_1)\times\sum_{\sim\{x_1\}}\left(f_B(x_2)f_C(x_1,x_2,x_3)\times\left(\sum_{\sim\{x_3\}} f_D(x_3,x_4)\right)\right.$$
$$\left.\times\left(\sum_{\sim\{x_3\}} f_E(x_3,x_5)\right)\right).$$

- Similarly, we find that

$$g_3(x_3) = \left(\sum_{\sim\{x_3\}} f_A(x_1)f_B(x_2)f_C(x_1,x_2,x_3)\right)$$
$$\times\left(\sum_{\sim\{x_3\}} f_D(x_3,x_4)\right)\times\left(\sum_{\sim\{x_3\}} f_E(x_3,x_5)\right).$$

- In computer science, arithmetic expressions like the right-hand sides of $g_1(x_1)$ and $g_3(x_3)$ are often represented by ordered rooted trees, here called expression trees.
- The operators shown in these figures are the function product and the summary, having various local functions as their arguments.

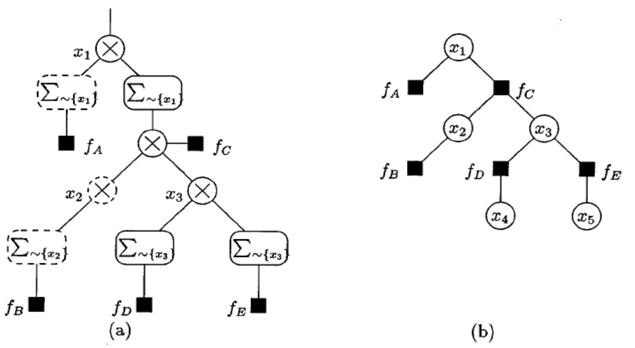


Figure 3: (a) A tree representation for $g_1(x_1)$.
(b) The factor graph with x_1 as root.

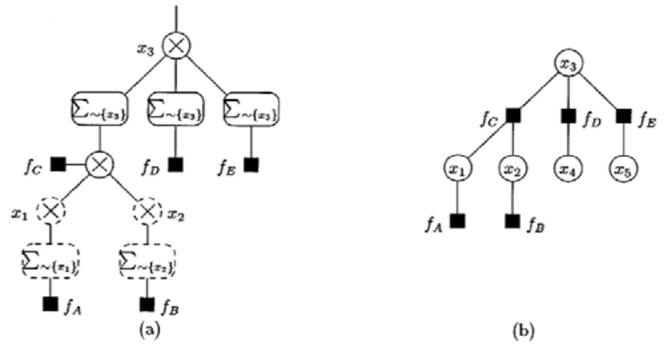
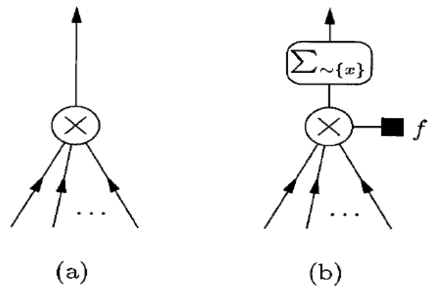


Figure 4: (a) A tree representation for $g_3(x_3)$.
(b) The factor graph with x_3 as root.

- When a factor graph is cycle-free, the factor graph not only encodes in its structure the factorization of the global function, but also provides an efficient way to compute the marginal functions.
- To find $g_i(x_i)$, we can arrange x_i as the root of the expression tree. Then every node v in the factor graph has a clearly defined parent node, namely, the neighboring node through which the unique path from v to x_i must pass.



- In (a), replace each variable node in the factor graph with a product operator.
- In (b), replace each factor node in the factor graph with a "form product and multiply by f"operator, and between a factor node f and its parent x, insert a $\sum_{\sim\{x\}}$ summary operator.

Computing a single marginal function

- Every expression tree represents an algorithm for computing the corresponding expression.
- To compute $g_i(x_i)$, use a "**bottom-up**" procedure:
  - To start at the leaves of the tree, with each operator vertex combining its operands and passing on the result as an operand for its parent.

- To best understand such algorithms, it helps to imagine that

$$\begin{cases} \text{processor} & \Leftrightarrow \quad \text{each vertex of the factor graph} \\ \text{channel} & \Leftrightarrow \quad \text{the factor-graph edge} \end{cases}$$

- Messages sent along the edges (channels) between the nodes (processor) are some appropriate description of some marginal function.

- The computation starts from the leaves.
- Each leaf variable node sends an identity function message to its parent
- Each leaf function node sends the description of f to its parent.
- After receiving the messages from all its children, the vertex then computes them and sends the updated messages to its parent.
  - For a variable node, it sends the product of all messages from its children to its parent.
  - For a function node with parent node x, it operates the summary of x of the product of all messages form its children and sends to its parent.
- The process terminates at node x_i and $g_i(x_i)$ is the product of all messages from the children of x.

Computing all marginal functions

- Computation of $g_i(x_i)$ for all i simultaneously can be efficiently accomplished by essentially “overlaying” on a single factor graph all possible instances of the single-i algorithm.
- At variable node x_i, the product of all incoming messages is the marginal function $g_i(x_i)$, just as in the single-i algorithm.
- Since this algorithm operates by computing various sums and products, we refer to it as the **sum-product algorithm**.
- The operations of sum and product in $R(+,\cdot)$ need to satisfy the distributive law

$$x \cdot (y+z) = x \cdot y + x \cdot z$$

Sum product algorithm

The sum-product algorithm

- The sum-product algorithm operates according to the following simple rule:

The sum-product update rule:
The message sent from a node v on an edge e is the product of the local function at v (or the unit function if v is a variable node) with all messages received at v on edges other than e, summarized for the variable associated with e.

- Let

$$\begin{cases} \mu_{x \to f}(x)\text{: The message sent from node } x \text{ to node } f. \\ \mu_{f \to x}(x)\text{: The message sent from node } f \text{ to node } x. \\ n(v)\text{: The set of neighbors of a given node } v \text{ in a factor graph.} \end{cases}$$

- The message computations performed by the sum-product algorithm may be expressed as follows:
  - ► variable to local function:

$$\mu_{x \to f}(x) = \prod_{h \in n(x) \setminus \{f\}} \mu_{h \to x}(x)$$

  - ► local function to variable:

$$\mu_{f \to x}(x) = \sum_{\sim\{x\}} \left(f(X) \prod_{y \in n(f) \setminus \{x\}} \mu_{y \to f}(y) \right)$$

where $X = n(f)$ is the set of arguments of the function f .

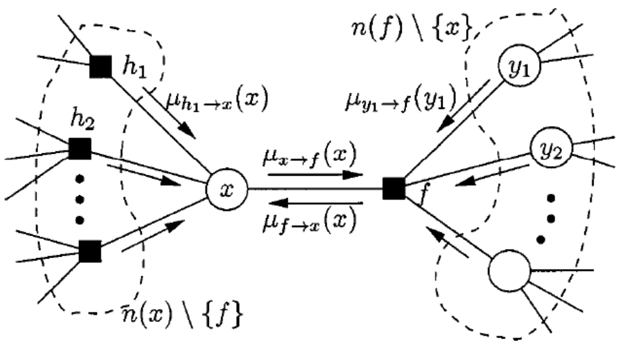


Figure 6: message passing in an edge between f and x.

► A detailed Example:
Fig. 7 shows the flow of messages that would be generated by the sum-product algorithm applied to the factor graph of Fig. 1.

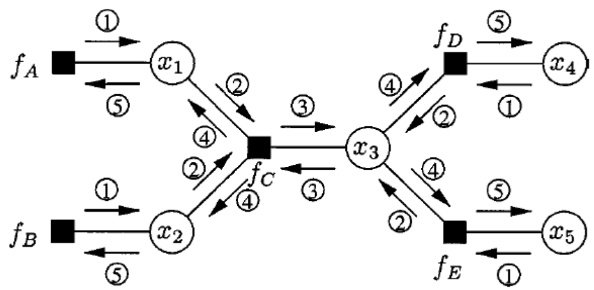


Figure 7: sum-product algorithm for a cycle-free graph .

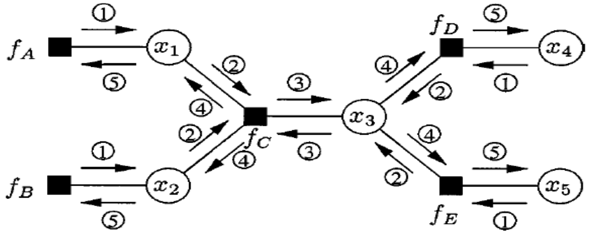
1.

$$\mu_{f_A \to x_1}(x_1) = \sum_{\sim\{x_1\}} f_A(x_1) = f_A(x_1)$$

$$\mu_{f_B \to x_2}(x_2) = \sum_{\sim\{x_2\}} f_B(x_2) = f_B(x_1)$$

$$\mu_{x_4 \to f_D}(x_4) = 1$$

$$\mu_{x_5 \to f_E}(x_5) = 1$$



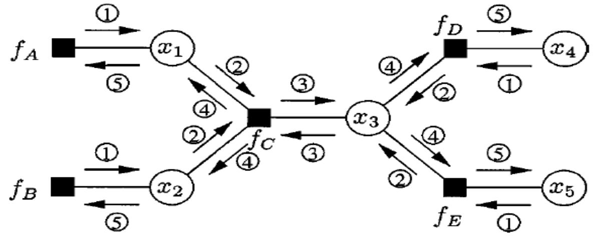
2.

$$\mu_{x_1 \to f_C}(x_1) = \mu_{f_A \to x_1}(x_1)$$

$$\mu_{x_2 \to f_C}(x_2) = \mu_{f_B \to x_2}(x_2)$$

$$\mu_{f_D \to x_3}(x_3) = \sum_{\sim\{x_3\}} \mu_{x_4 \to f_D}(x_4) f_D(x_3, x_4) = \sum_{\sim\{x_3\}} f_D(x_3, x_4)$$

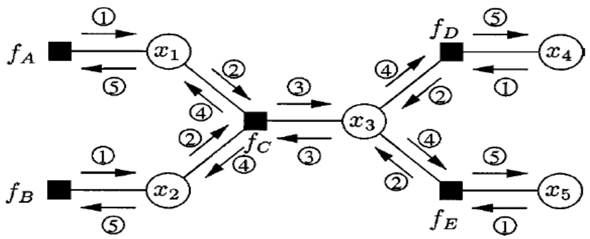
$$\mu_{f_E \to x_3}(x_3) = \sum_{\sim\{x_3\}} \mu_{x_5 \to f_E}(x_5) f_E(x_3, x_5) = \sum_{\sim\{x_3\}} f_E(x_3, x_5)$$



3.

$$\mu_{f_C \to x_3}(x_3) = \sum_{\sim\{x_3\}} \mu_{x_1 \to f_C}(x_1)\mu_{x_2 \to f_C}(x_2) f_C(x_1, x_2, x_3)$$

$$\mu_{x_3 \to f_C}(x_3) = \mu_{f_D \to x_3}(x_3)\mu_{f_E \to x_3}(x_3)$$



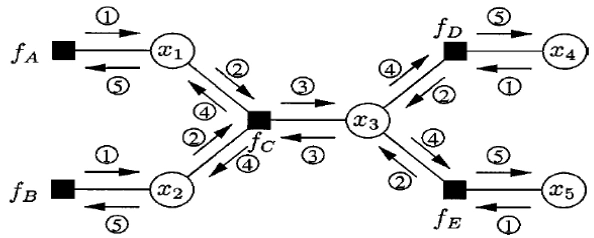
4.

$$\mu_{f_C \to x_1}(x_1) = \sum_{\sim\{x_1\}} \mu_{x_3 \to f_C}(x_3)\mu_{x_2 \to f_C}(x_2) f_C(x_1, x_2, x_3)$$

$$\mu_{f_C \to x_2}(x_2) = \sum_{\sim\{x_2\}} \mu_{x_3 \to f_C}(x_3)\mu_{x_1 \to f_C}(x_1) f_C(x_1, x_2, x_3)$$

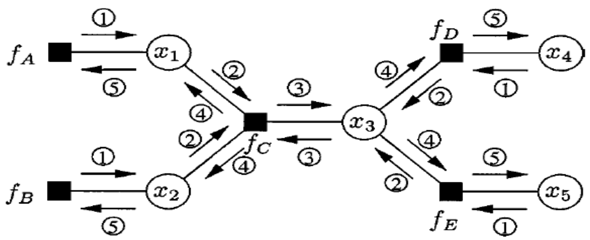
$$\mu_{x_3 \to f_D}(x_3) = \mu_{f_C \to x_3}(x_3)\mu_{f_E \to x_3}(x_3)$$

$$\mu_{x_3 \to f_E}(x_3) = \mu_{f_C \to x_3}(x_3)\mu_{f_D \to x_3}(x_3)$$



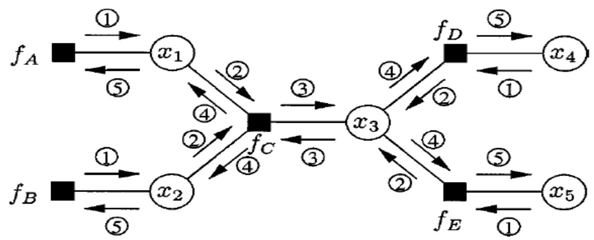
5.

$$
\begin{aligned}
\mu_{x_1\to f_A}(x_1) &= \mu_{f_C\to x_1}(x_1)\\
\mu_{x_2\to f_B}(x_2) &= \mu_{f_C\to x_2}(x_2)\\
\mu_{f_D\to x_4}(x_4) &= \sum_{\sim\{x_4\}} \mu_{x_3\to f_D}(x_3) f_D(x_3,x_4)\\
\mu_{f_E\to x_5}(x_5) &= \sum_{\sim\{x_5\}} \mu_{x_3\to f_E}(x_3) f_E(x_3,x_5)
\end{aligned}
$$



6. Termination:

$$
\begin{aligned}
g_1(x_1) &= \mu_{f_A\to x_1}(x_1)\mu_{f_C\to x_1}(x_1)\\
g_2(x_2) &= \mu_{f_B\to x_2}(x_2)\mu_{f_C\to x_2}(x_2)\\
g_3(x_3) &= \mu_{f_C\to x_3}(x_3)\mu_{f_D\to x_3}(x_3)\mu_{f_E\to x_3}(x_3)\\
g_4(x_4) &= \mu_{f_D\to x_4}(x_4)\\
g_5(x_5) &= \mu_{f_E\to x_5}(x_5)
\end{aligned}
$$



- Equivalently, since the message passed on any given edge is equal to the product of all but one of these messages, we may compute $g_i(x_i)$ as the product of the two messages that were passed (in opposite directions) over any single edge incident on x_i.
- Thus, for example, we may compute $g_3(x_3)$ in three other ways as follows:

$$
\begin{aligned}
g_3(x_3) &= \mu_{f_C\to x_3}(x_3)\mu_{x_3\to f_C}(x_3)\\
&= \mu_{f_D\to x_3}(x_3)\mu_{x_3\to f_D}(x_3)\\
&= \mu_{f_E\to x_3}(x_3)\mu_{x_3\to f_E}(x_3)
\end{aligned}
$$

Code realization: behavior/probability modeling

Code realization and factor graphs

- We now apply the factor graph description to coding realization.
- Forney makes a clear distinction between the realization of a code and the graphic model of the realization.
- In other words, a code can have many realizations and for each realization, we can associated it with different graphic modeling.
- We will describe various ways in which factor graphs may be used to model the code/systems.
- We will use "the realization of a code" and "the modeling of a code" interchangeably.

- Given a code, we will talk about two realizations for codes and one realization for decoding:
  1. behavior modeling: check-based and trellis-based realization
  2. probability modeling
- For each realization, we will only consider the factor graph associated with it and leave the normal graph to the readers.

- If P is a predicate (Boolean proposition) involving some set of variables, then $[P]$ is the $\{0,1\}$-valued function that indicates the truth of P, i.e.

$$[P] = \begin{cases} 1, & \texttt{if } P \texttt{ is true} \\ 0, & \texttt{otherwise}. \end{cases}$$

- For example, $f(x,y,z) = [x+y+z=0]$ is the function that takes a value of 1 if (x,y,z) has even weight, and 0 otherwise.

- If we let $\wedge$ denote the logical conjunction or "AND" operator, then an important property of Iverson's convention is that

$$[P = P_1 \wedge P_2 \wedge \ldots \wedge P_n] = [P_1]\,[P_2]\ldots[P_n]\,.$$

Thus, P is true iff P_i is true for all i; then the complicated global indicator $[P]$ can be factored into n simple indicators P_i product. Hence we can represent P by a factor graph.

- In the "behavioral" modeling of codes/systems:
  Use the characteristic (i.e., indicator) function for the given code/behavior; then the factor graph for the characteristic function is the graphic description for the code/beharior.
  - Factorizations of this characteristic function can give important structural information about the model.

- In probabilistic modeling of systems:
  A factor graph can be used to represent the joint probability mass function of the variables that comprise the system.
- Factorizations of this function can give important information about statistical dependencies among these variables.
- For example, in channel coding, we model both the valid behavior (i.e., the set of codewords) and the a posteriori joint probability mass function over the variables that define the codewords given the received output of a channel.

check-based modeling

- Let $x_1, x_2 \ldots, x_n$ be a collection of variables with configuration space $S = A_1 \times A_2 \times \ldots \times A_n$.
- Any element of S is called a configuration (word).
- By a code/behavior in S, we mean any subset B of S.
  - The elements of B are called the valid configurations (codewords).
  - Since a system is specified via its behavior B, this approach is known as **behavioral modeling**.

- Behavioral modeling is natural for codes.
  - If the domain of each variable is some finite alphabet A, so that the configuration space is the n-fold Cartesian product $S = A^n$, then a behavior $C \subset S$ is called a block code of length n over A, and the valid configurations are called codewords.
- The characteristic (or set membership indicator) function for a behavior B is defined as

$$\chi_B(x_1, \ldots, x_n) := [(x_1, \ldots, x_n) \in B]$$

Obviously, specifying χ_B is equivalent to specifying B.

Example 2 (Tanner Graphs for Linear Codes)

The characteristic function for any linear code defined by an $r \times n$ parity check matrix H can be represented by a factor graph having n variable nodes and r factor nodes.

- Let C be the binary linear code with parity-check matrix

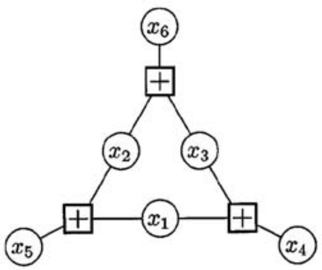
$$H = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

- C is the set of all binary 6-tuples $\mathbf{x} \triangleq (x_1, x_2, \ldots, x_6)$ that satisfy three simultaneous equations: $C = \{x : H\mathbf{x}^T = 0\}$.
- The factor graph of C has 6 symbol nodes and 3 check nodes.

– The indication function of C is

$$\begin{aligned} \chi_C(x_1, x_2, \ldots, x_6) &= [(x_1, x_2, \ldots, x_6) \in C] \\ &= [x_1 \oplus x_2 \oplus x_5 = 0]\,[x_2 \oplus x_3 \oplus x_6 = 0] \\ &\quad \cdot [x_1 \oplus x_3 \oplus x_4 = 0] \end{aligned}$$

– The factor graph (Tanner graph) for the code is:



– By this example, a Tanner graph for any $[n, k]$ linear block code may be obtained from a parity-check matrix $H = [h_{ij}]$ for the code.

– Such a parity-check matrix has n columns and at least $n - k$ rows.

– Variable (symbol) nodes correspond to the columns of H and factor (check) nodes to the rows of H, with an edge connecting factor node i to variable node j if and only if $h_{ij} \neq 0$.

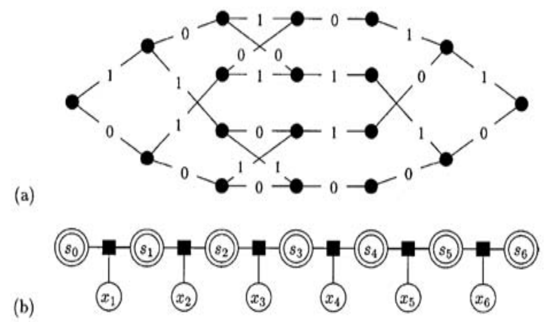
► Of course, since there are, in general, many parity check matrices that represent a given code, there are, in general, many Tanner graph representations for the code.

trellis-based modeling

- Often, a description of a system is simplified by introducing hidden (sometimes called auxiliary, latent, or state) variables. Nonhidden variables are called visible.
- A particular behavior B with both auxiliary and visible variables is said to represent a given (visible) behavior C if the projection of the elements of B on the visible variables is equal to C.
- Any factor graph for B is then considered to be also a factor graph for C.

- Such graphs were introduced by Wiberg et al. and may be called Wiberg-type graphs.
- ► As in Wiberg, hidden variable nodes are in general indicated by a double circle.
- An important class of models with hidden variables are the trellis representations.

- A trellis for a block code C is an edge labelled directed graph with one left root and one right goal vertices.
- Each sequence of edge labels encountered in any directed path from the root vertex to the goal vertex is a codeword in C.
- The collection of all pathes forms the code.
- All paths from the root to any given vertex should have the same fixed length d, called the depth of the given vertex.
- The root vertex has depth 0, and the goal vertex has depth n. The set of depth d vertices can be viewed as the d-th state space.



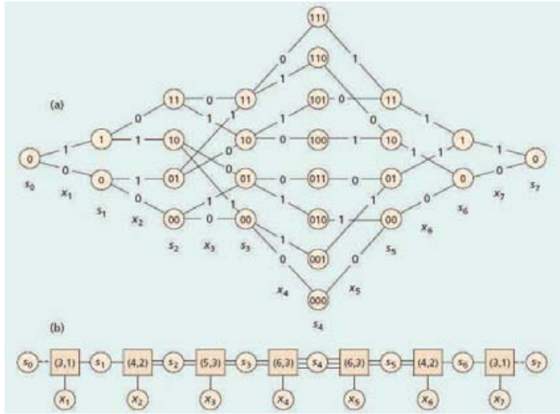
- (a) A trellis and its (b) Wibger graph for a $[6, 3, 4]$ code.
- In addition to the visible variable nodes $x_1, x_2, \ldots, x_6$, there are also hidden (state) variable nodes $s_0, s_1, \ldots, s_6$.
- Each local check corresponds to one section of the trellis.

- In this example, the local behavior T_2 corresponding to the second trellis section from the left in Fig. 9 consists of the following triples (s_1, x_2, s_2):

$$T_2 = \{(0,0,00),\ (0,1,10),\ (1,1,01),\ (1,0,11)\}$$

  where the domains of the state variables s_1 and s_2 are taken to be $\{0,1\}$ and $\{00,01,10,11\}$, respectively, numbered from bottom to top.
- Each element of the local behavior corresponds to one trellis edge.
- The corresponding factor node in the Wiberg-type graph is the indicator function $f(s_1, x_2, s_2) = [(s_1, x_2, s_2) \in T_2]$.

- (a) A trellis and its (b) Wibger graph for a $[7,4,3]$ code.



- A trellis divides naturally into n sections, where the ith trellis section T_i is the subgraph of the trellis induced by the vertices at depth $i-1$ and depth i.
- The set of edge labels in T_i may be viewed as the domain of a (visible) variable x_i.
- In effect, each trellis section T_i defines a "local behavior" that constrains the possible combinations of s_{i-1}, x_i, and s_i.

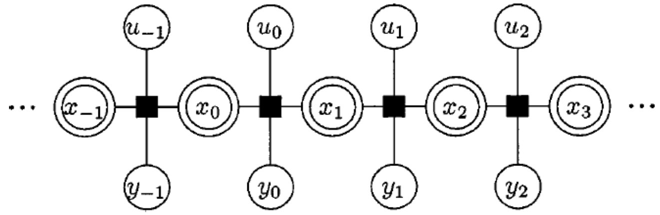
- It is important to note that a factor graph corresponding to a trellis is cycle-free.
- Since every code has a trellis representation, it follows that every code can be represented by a cycle-free factor graph.

► Unfortunately, it often turns out that the state-space sizes (the sizes of domains of the state variables) can easily become too large to be practical.

  For example, trellis representations of the overall turbo codes have enormous state spaces. But the trellis of the two component codes in turbo codes is relatively small.

Other behavior modeling

- Trellises are basically conventional state-space system models, and the generic factor graph shown below can represent any state-space model of a time-invariant or time-varying system.



Here, we allow a trellis edge to have both an input label and an output label.

Example 4 (State-Space Models)

- The classical linear time-invariant state-space model is given by the equations

$$\begin{aligned} x(j+1) &= Ax(j) + Bu(j) \\ y(j) &= Cx(j) + Du(j) \end{aligned}$$

where $j \in \mathbb{Z}$ is the discrete time index

- $u(j) = (u_1(j), \ldots, u_k(j))$ are the time–j input variables, $y(j) = (y_1(j), \ldots, y_n(j))$ are the time–j output variables, and $x(j) = (x_1(j), \ldots, x_m(j))$ are the time–j state variables.
- A, B, C, D are $m \times m$, $m \times k$, $n \times m$, and $n \times k$, matrices respectively.

- The time-j check function

$$f\left(x(j), u(j), y(j), x(j+1)\right) : F^m \times F^k \times F^n \times F^m \to \{0, 1\}$$

is

$$\begin{aligned} f\left(x(j), u(j), y(j), x(j+1)\right) = \;& [x(j+1) = Ax(j) + Bu(j)] \\ & \cdot [y(j) = Cx(j) + Du(j)]\,. \end{aligned}$$

Summary of behavioral realization

- The behavioral/code realization of a code is characterized by three sets of elements:
  1. A symbol configuration space
  $$A = \prod_{k \in I_A} A_k,$$
  2. A state configuration space
  $$S = \prod_{j \in I_S} S_j,$$
  3. some local constraints
  $$\{C_i : i \in I_C\},$$

  where I_A, I_S, and I_C are discrete index sets.

- A check-based realization of a code $C \subset A$ is defined by some local constraints (behaviors). Each local behavior C_i involves a subset of some symbol variables, indexed by $I_A(i) \subset I_A$:

$$C_i \subset T_i = \prod_{k \in I_A(i)} A_k$$

- The code $C \subset A$ is the set of all valid symbol configuration satisfying **all** local constraints:

$$C = \{a \in A | a_{I_A(i)} \in C_i, \forall i \in I_C\},$$

where $a_{|I_A(i)} = \{a_k | k \in I_A(i)\} \in T_i$ is the projection of a onto T_i.

- The factor graph associated this realization is also called a Tanner graph.

- A trellis-based realization of a code $C \subset A$ is defined by some local constraints (behaviors). Each local behavior C_i involves a subset of some symbol variables, indexed by $I_A(i) \subset I_A$ and some state variables, indexed by $I_S(i) \subset I_S$:

$$C_i \subset W_i = \prod_{k \in I_A(i)} A_k \times \prod_{j \in I_S(i)} S_j$$

- The full behavior $B \subset A \times S$ is the set of all valid symbol/state configuration satisfying **all** local constraints:

$$B = \{(a, s) \in A \times S | (a_{|I_A(i)}, s_{|I_S(i)}) \in C_i, \forall i \in I_C\},$$

where $(a_{|I_A(i)}, s_{|I_S(i)}) = \{\{a_k, k \in I_A(i)\}, \{s_j, j \in I_S(i)\}\} \in W_i$ is the projection of a onto W_i.

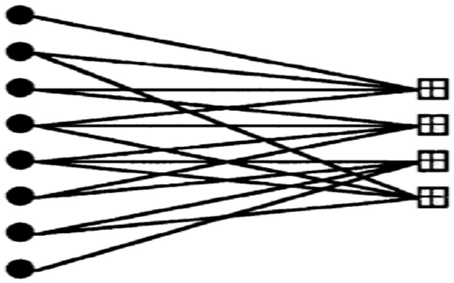
- The code C is then the projection of B onto the symbol configuration space, i.e., $C = B_{|A}$.
- The factor graph associated this realization is also called a Wiber graph.

- In coding, there are three possible behavior/code realizations
  1. code modeling by a parity check matrix (with cycles)
  2. code modeling by conventional trellis (cycle-free)
  3. code modeling by tailbiting trellis (with one cycle)

Code modeling by a parity check matrix (with many cycles)

$I_A = [1, n]$ and $I_C = [1, r]$

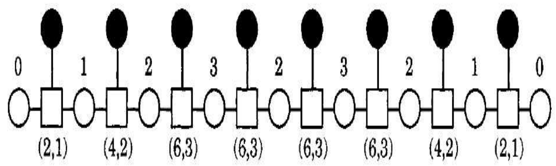
$C_i = [n_i, n_i - 1, 2]$ single parity check code, where $n_i = |I_A(i)|$.



Code modeling by conventional trellis (cycle-free)

$I_A = [0, n-1] = Z_n$, $I_S = [0, n]$, and $I_C = [0, n-1] = Z_n$

C_i is the i-th trellis section specifies the valid local configuration: $(s_i, a_i, s_{i+1}) \in S_i \times A_i \times S_{i+1}$

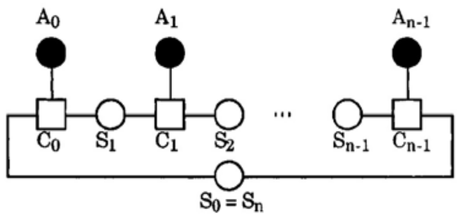


Code modeling by tailbiting trellis (with one cycle)

$I_A = Z_n$, $I_S = Z_n$, and $I_C = Z_n$

Same i-th trellis section as conventional trellis except the $n-1$-th

$$(s_{n-1}, a_{n-1}, s_0) \in S_{n-1} \times A_{n-1} \times S_0$$



Probability Modeling for codes

- In decoding of DMC with input $x \in X$ and output $y \in Y$, we are given the prior probability $p(x)$, usually uniform distributed, and the conditional probability $p(y|x)$.
- In BSC, $X = Y = F_2^n$ and

$$\begin{aligned} p(y|x) &= p^{d(x,y)}(1-p)^{n-d(x,y)} \\ &= \prod_{i=1}^{n} p^{d(x_i,y_i)}(1-p)^{1-d(x_i,y_i)} \end{aligned}$$

- In AWGN, $X = F_2^n$ and $Y = R^n$ and

$$p(y|x) = \prod_{i=1}^{n} \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y_i - x_i)^2}{N_0}}$$

- We can find the posterior probability $p(x|y) = p(x)p(y|x)$.
- Then we may do the MAP decoding for minimizing block or bit error probability:

$$\max_{x \in C} p(x|y) = \max_{x \in C} p(x)p(y|x)$$

$$p(x_i|y) = \sum_{\sim\{x_i\}} p(x|y)$$

- Since conditional and unconditional independence of random variables is expressed in terms of a factorization of their joint probability mass or density function, factor graphs for probability distributions arise in many situations.

Example 5 (APP Distributions)

Consider the standard coding model in which a codeword $x = (x_1, \ldots, x_n)$ is selected from a code C of length n and transmitted over a memoryless channel with corresponding output sequence $y = (y_1, \ldots, y_n)$.

- For each fixed observation y, the joint a posteriori probability (APP) distribution for the components of x (i.e., $p(x|y)$) is proportional to the function $g(x) = f(y|x)p(x)$, where $p(x)$ is the a priori distribution of x and $f(y|x)$ is the conditional pdf for y when x is transmitted.

- Assuming that the a priori distribution for the transmitted vectors is uniform over codewords, we have

$$p(x) = \chi_C(x)/|C|,$$

where $\chi_C(x)$ is the characteristic function for C and $|C|$ is the number of codewords in C.
If the channel is memoryless, then $f(y|x)$ factors as

$$f(y_1, \ldots, y_n|x_1, \ldots, x_n) = \prod_{i=1}^{n} f(y_i|x_i).$$

- Under these assumptions, we have

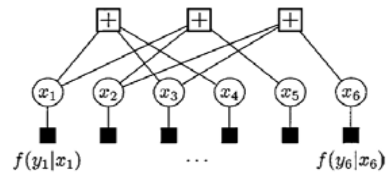
$$g(x_1, \ldots, x_n) = \frac{1}{|C|}\chi_C(x_1, \ldots, x_n)\prod_{i=1}^{n} f(y_i|x_i).$$

  Now the characteristic function χ_C itself may factor into a product of local characteristic functions.
- Given a factor graph F for $\chi_C(x)$, we obtain a factor graph for (a scaled version of) the APP distribution over simply by augmenting F with factor nodes corresponding to the different $f(y_i|x_i)$ factors.
- The ith such factor has only one argument, namely x_i, since y_i is regarded as a parameter. Thus, the corresponding factor nodes appear as pendant vertices ("dongles") in the factor graph.

- Consider a $C[6,3,3]$ binary linear code with $p(x|y)$

$$\begin{aligned} g(x_1, \ldots, x_6) &= [x_1 \oplus x_2 \oplus x_5 = 0] \cdot [x_2 \oplus x_3 \oplus x_6 = 0] \\ &\quad \cdot [x_1 \oplus x_3 \oplus x_4 = 0] \cdot \prod_{i=1}^{6} f(y_i|x_i) \end{aligned}$$

  whose factor graph is shown in Fig. 11.



- With this factor graph, we can run the iterative sum product decoding.

Other probability modeling

Example 6 (Markov Chains, Hidden Markov Models)

In general, let $f(x_1, \ldots, x_n)$ denote the joint probability mass function of a collection of random variables. By the chain rule of conditional probability, we may always express this function as

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{n} f(x_i|x_1, \ldots, x_{i-1}).$$

- For example, if $n = 4$, then

$$f(x_1, \ldots, x_4) = f(x_1)f(x_2|x_1)f(x_3|x_1x_2)f(x_4|x_1, x_2, x_3)$$

  which has the factor graph representation shown in Fig. 12(b).

- In general, since all variables appear as arguments of $f(x_n|x_1, \ldots, x_{n-1})$, the factor graph of Fig. 12(b) has no advantage over the trivial factor graph shown in Fig. 12(a).
- Suppose that random variables X_1, X_2, …, X_n (in that order) form a Markov chain. We then obtain the nontrivial factorization

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{n} f(x_i|x_{i-1})$$

  whose factor graph is shown in Fig. 12(c) for $n = 4$.

- Continuing this Markov chain example, if we cannot observe each X_i directly, but instead can observe only Y_i, the output of a memoryless channel with X_i as input, then we obtain a so-called "hidden Markov model."
- The joint probability mass or density function for these random variables then factors as

$$f(x_1, \ldots, x_n, y_1, \ldots, y_n) = \prod_{i=1}^{n} f(x_i|x_{i-1}) f(y_i|x_i)$$

whose factor graph is shown in Fig. 12(d) for $n = 4$.

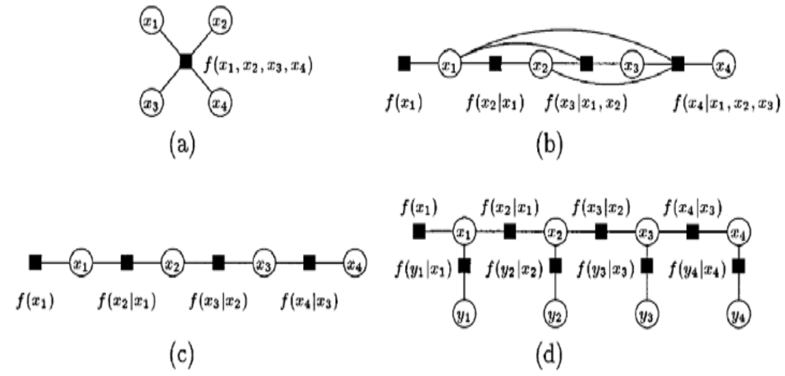


Figure 12: Factor graphs for probability distributions. (a) The trivial factor graph. (b) The chain-rule factorization. (c) A Markov chain. (d) A hidden Markov model.

Trellis decoding for trellis-based realization

Trellis processing

- An important family of cycle free factor graphs for a code is the one with the chain graphs that represent trellises or Markov models for it.
- Apply the sum-product algorithm to the trellises; it is obvious that a variety of well-known algorithms:
  1. the forward/backward algorithm (BCJR, MAP, APP)
  2. the Viterbi algorithm, SOVA

  may be viewed as special cases of the sum-product algorithm.

The forward/backward algorithm

- We start with the forward/backward algorithm, sometimes referred to in coding theory as the BCJR, APP, or MAP algorithm.
- The factor graph of Fig. 13 models the most general situation, which involves a combination of behavioral and probabilistic modeling.
- We have vectors $u = (u_1, u_2, \ldots, u_k)$, $x = (x_1, x_2, \ldots, x_n)$, and $s = (s_0, \ldots, s_n)$ that represent, respectively, input variables, output variables, and state variables in a Markov model, where each variable is assumed to take on values in a finite domain.

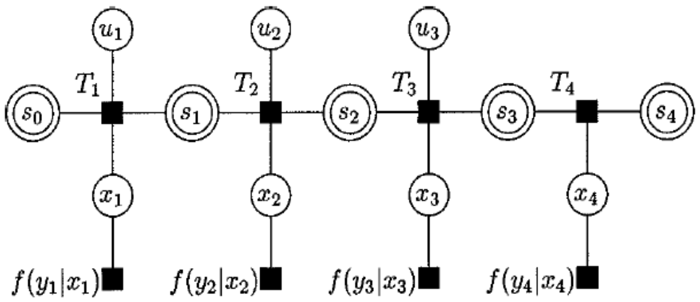


Figure 13: The factor graph in which the forward/backward algorithm operates: the s_i are state variables, the u_i are input variables, the x_i are output variables, and each y_i is the output of a memoryless channel with input x_i.

- This model is a "hidden" Markov model in which we cannot observe the output symbols directly.
- The a posteriori joint probability mass function for u, s, and x given the observation y is proportional to

$$g_y(u, s, x) := \prod_{i=1}^{n} T_i(s_{i-1}, x_i, u_i, s_i) \prod_{i=1}^{n} f(y_i|x_i)$$

where y is again regarded as a parameter of g. The factor graph of Fig. 13 represents this factorization of g.

- Given y, we would like to find the APPs $p(u_i|y)$ for each i. These marginal probabilities are proportional to the following marginal functions associated with g:

$$p(u_i|y) \propto \sum_{\sim\{u_i\}} g_y(u, s, x).$$

Since the factor graph of Fig. 13 is cycle-free, these marginal functions may be computed by applying the sum-product algorithm to the factor graph of Fig. 13.

Initialization

- This is a cycle-free factor graph for the code and the sum product algorithm starts at the leaf variable nodes, i.e.,

$$s_0, s_n, \text{ and } u_i, 1 \le i \le k,$$

and the leaf functional nodes, i.e.,

$$f(y_i|x_i), 1 \le i \le n.$$

- Trivial messages are sent by the input variable nodes and the endmost state variable nodes, i.e.,

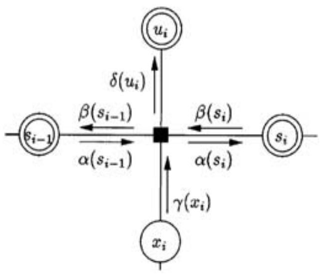
$$\mu_{u_i \to T_i}(u_i = 0) = \mu_{u_i \to T_i}(u_i = 1) = 1, 1 \le i \le k,$$

$$\mu_{s_0 \to T_1}(s_0 = 0) = \mu_{s_n \to T_n}(s_n = 0) = 1,$$

$$\mu_{s_0 \to T_1}(s_0 \ne 0) = \mu_{s_n \to T_n}(s_n \ne 0) = 1$$

- Each pendant factor node, $f(y_i|x_i)$, $1 \le i \le n$, sends a message $(f(y_i|x_i = 0), f(y_i|x_i = 1))$ to the corresponding output variable node.
- Since the output variable nodes, x_i, have degree two, no computation is performed; instead, incoming messages received on one edge are simply transferred to the other edge and sent to the corresponding trellis check node T_i.

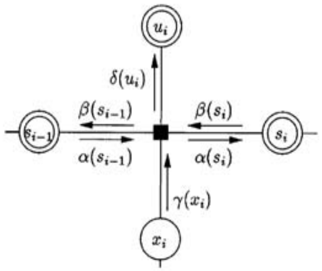
- In the literature on the forward/backward algorithm:
  - The message $\mu_{x_i \to T_i}(x_i)$ is denoted as $\gamma(x_i)$
  - The message $\mu_{s_i \to T_{i+1}}(s_i)$ is denoted as $\alpha(s_i)$
  - The message $\mu_{s_i \to T_i}(s_i)$ is denoted as $\beta(s_i)$.
  - The message $\mu_{T_i \to u_i}(u_i)$ is denoted as $\delta(u_i)$.



- The operation of the sum-product algorithm creates two natural recursions:
  one to compute $\alpha(s_i)$ as a function of $\alpha(s_{i-1})$ and $\gamma(x_i)$ and the other to compute $\beta(s_{i-1})$ as a function of $\beta(s_i)$ and $\gamma(x_i)$.
- These two recursions are called the forward and backward recursions, respectively, according to the direction of message flow in the trellis.

The forward and backward recursions do not interact, so they could be computed in parallel.

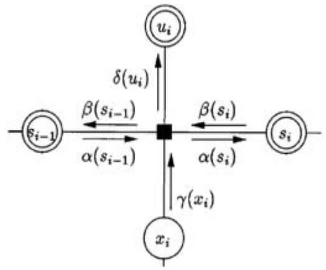
- Fig. 14 gives a detailed view of the message flow for a single trellis section. The local function in this figure represents the trellis check $T_i(s_{i-1}, u_i, x_i, s_i)$.



The Forward/Backward Recursions

$$\alpha(s_i) = \sum_{\sim\{s_i\}} T_i(s_{i-1}, u_i, x_i, s_i)\alpha(s_{i-1})\gamma(x_i)$$

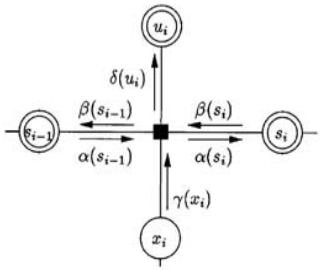
$$\beta(s_{i-1}) = \sum_{\sim\{s_{i-1}\}} T_i(s_{i-1}, u_i, x_i, s_i)\beta(s_i)\gamma(x_i)$$



Termination

The algorithm terminates with the computation of the $\delta(u_i)$.

$$\delta(u_i) = \sum_{\sim\{u_i\}} T_i(s_{i-1}, u_i, x_i, s_i)\alpha(s_{i-1})\beta(s_i)\gamma(x_i).$$

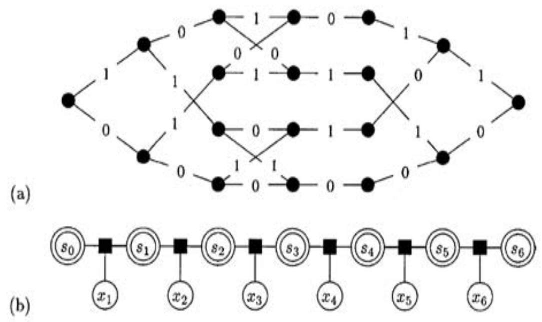


- These sums can be viewed as being defined over valid trellis edges $e = (s_{i-1}, u_i, x_i, s_i)$ such that $T_i(e) = 1$.
- For each edge e, we let $\alpha(e) = \alpha(s_{i-1})$, $\beta(e) = \beta(s_i)$, and $\gamma(e) = \gamma(x_i)$.
- Denoting by $E_i(s)$ the set of edges incident on a state s in the ith trellis section, the α and β update equations may be rewritten as

$$\alpha(s_i) = \sum_{e \in E_i(s_i)} \alpha(e)\gamma(e)$$

$$\beta(s_{i-1}) = \sum_{e \in E_i(s_{i-1})} \beta(e)\gamma(e)$$

The basic operations in the forward and backward recursions are therefore "sum of products."



The min-sum and max-product semirings

- The codomain R of the global function g represented by a factor graph may in general be any semiring with two operations "+" and "·" that satisfy the distributive law

$$(\forall\ x,\ y,\ z\ \in\ R) \qquad x\cdot(y+z)=(x\cdot y)+(x\cdot z)$$

- For nonnegative real-valued quantities x, y, and z, "·" distributes over "max"

$$x(\max(y,z)) = \max(xy, xz).$$

- With maximization as a summary operator, the maximum value of a nonnegative real-valued function $g(x_1,\ldots,x_n)$ is viewed as the "complete summary" of g; i.e.

$$\begin{aligned}\max g(x_1,\ldots,x_n) &= \max_{x_1}(\max_{x_2}(\ldots(\max_{x_n} g(x_1,\ldots,x_n)))) \\ &= \sum_{\sim\{\}} g(x_1,\ldots,x_n)\end{aligned}$$

- MAP decoding for minimizing the codeword error probability

$$\max_{x\in C} p(x|y) = \max_{x\in C} p(x)p(y|x)$$

.

- MAP decoding is equivalent to ML decoding if $p(x)=\frac{1}{M}$.

► For the MAP or ML decoding problem, we are interested in finding a valid configuration x that achieves this maximum.

- The sum product decoding becomes max. product decoding.

- In practice, MLSD is most often carried out in the negative log-likelihood domain.
- Here, the "product" operation becomes a "sum" and the "max" operation becomes a "min" operation, so that we deal with the "min-sum" semiring.

▶ For real-valued quantities x, y, z, "+" distributes over "min"

$$x + \min(y, z) = \min(x + y, x + z).$$

- Applying the min-sum algorithm in this context yields the same message flow as in the forward/backward algorithm.
- As in the forward/backward algorithm, we may write an update equation for the various messages.

▶ For example, the basic update equation corresponding to $\alpha(s_i) = \sum_{e \in E_i(s_i)} \alpha(e)\gamma(e)$ is

$$\alpha(s_i) = \min_{e \in E_i(s_i)} (\alpha(e) + \gamma(e))$$

so that the basic operation is a "minimum of sums" instead of a "sum of products."

- A similar recursion may be used in the backward direction, and from the results of the two recursions the most likely sequence may be determined. The result is a "bidirectional" Viterbi algorithm.

Iterative decoding for LDPC, turbo, and RA codes

- We now apply the sum product algorithm to the factor graphs with cycles
- We will discuss three Iterative decoding for:
  1. Turbo codes
  2. LDPC codes
  3. RA codes

Iterative decoding of turbo code

- A "turbo code" ("parallel concatenated convolutional code") has the encoder structure shown in Figure 15(a).

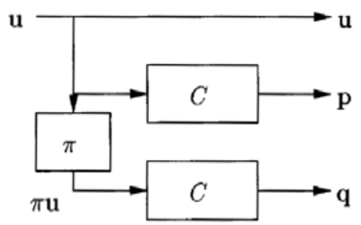
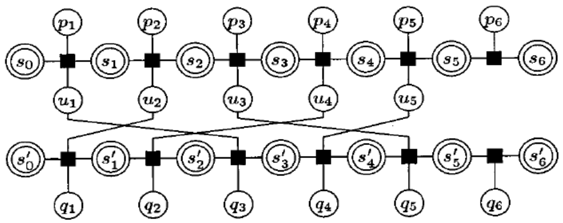


Figure 15: Turbo code. (a) Encoder block diagram.

A block u of data to be transmitted enters a systematic encoder which produces u, and two parity-check sequences p and q at its output.

- The first parity-check sequence p is generated via a standard recursive convolutional encoder; viewed together, u and p would form the output of a standard rate $\frac{1}{2}$ convolutional code.
- The second parity-check sequence q is generated by applying a permutation π to the input stream, and applying the permuted stream to a second convolutional encoder.

- A factor graph representation for a (very) short turbo code is shown in Fig. 15(b).
- Included in the figure are the state variables for the two constituent encoders, as well as a terminating trellis section in which no data is absorbed, but outputs are generated.



- Iterative decoding of turbo codes is usually accomplished via a message-passing schedule that involves a forward/backward computation over the portion of the graph representing one constituent code, followed by propagation of messages between encoders (resulting in the so-called extrinsic information in the turbo-coding literature).
- This is then followed by another forward/backward computation over the other constituent code, and propagation of messages back to the first encoder.

LDPC codes

- LDPC codes were introduced by Gallager in the early 1960s.
- LDPC codes are defined in terms of a regular bipartite graph.
- In a (j, k) LDPC code, left nodes, representing codeword symbols, all have degree j, while right nodes, representing checks, all have degree k.

- For example, Fig. 16 illustrates the factor graph for a short $(2, 4)$ LDPC code.

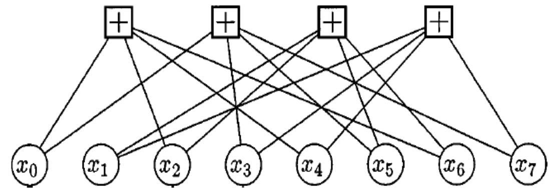


Figure 16: A factor graph for a LDPC code.

RA codes

- RA codes are a special, low-complexity class of turbo codes introduced by Divsalar, McEliece, and others, who initially devised these codes because their ensemble weight distributions are relatively easy to derive.
- An encoder for an RA code operates on k input bits $u_1, \ldots, u_k$, repeating each bit Q times, and permuting the result to arrive at a sequence $z_1, \ldots, z_{kQ}$.
- An output sequence $x_1, \ldots, x_{kQ}$ is formed via an accumulator that satisfies $x_1 = z_1$ and $x_i = x_{i-1} + z_i$ for $i > 1$.

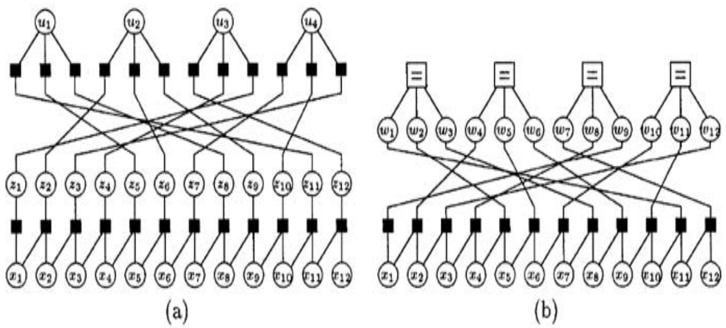


Figure 17: Two Equivalent factor graphs for an RA code.

Iterative decoding for LDPC and RA codes

- We treat here only the important case where all variables are binary (Bernoulli) and all functions except single-variable functions are parity checks.
- The probability mass function for a binary random variable may be represented by the vector (p_0, p_1), where $p_0 + p_1 = 1$.
- According to the generic updating rules, when messages (p_0, p_1) and (q_0, q_1) arrive at a variable node of degree three, the resulting (normalized) output message should be

$$\texttt{VAR}(p_0, p_1, q_0, q_1) = \left(\frac{p_0 q_0}{p_0 q_0 + p_1 q_1}, \frac{p_1 q_1}{p_0 q_0 + p_1 q_1} \right).$$

- Similarly, at a check node representing the function

$$f(x, y, z) = [x \oplus y \oplus z = 0]$$

(where "$\oplus$" represents modulo-2 addition), we have

$$\texttt{CHK}(p_0, P_1, q_0, q_1) = (p_0 q_0 + p_1 q_1, p_0 q_1 + p_1 q_0).$$

- Since $p_0 + p_1$, binary probability mass functions can be parametrized by a single value.

- Depending on the parametrization, various probability gate implementations arise. We give four different parametrizations, and derive the `VAR` and `CHK` functions for each.
  1. Likelihood ratio (LR)
  2. Log-likelihood ratio (LLR)
  3. Likelihood difference (LD)
  4. Signed log-likelihood difference (SLLD)

◄ Likelihood ratio (LR) ►

Definition: $\lambda(p_0, p_1) = p_0/p_1$.

$$\texttt{VAR}(\lambda_1, \lambda_2) = \lambda_1 \lambda_2$$
$$\texttt{CHK}(\lambda_1, \lambda_2) = \tfrac{1+\lambda_1\lambda_2}{\lambda_1+\lambda_2}$$

◄ Log-likelihood ratio (LLR) ►

Definition: $\Lambda(p_0, p_1) = \ln(p_0/p_1)$.

$$\begin{aligned} \texttt{VAR}(\Lambda_1, \Lambda_2) = &\ \Lambda_1 + \Lambda_2 \\ \texttt{CHK}(\Lambda_1, \Lambda_2) = &\ \ln(\cosh((\Lambda_1 + \Lambda_2)/2)) \\ &- \ln(\cosh((\Lambda_1 - \Lambda_2)/2)) \\ = &\ 2\tanh^{-1}(\tanh(\Lambda_1/2)\tanh(\Lambda_2/2)). \end{aligned}$$

◄ **Likelihood difference (LD)** ►

Definition: $\delta(p_0, p_1) = p_0 - p_1$.

$$\mathtt{VAR}\,(\delta_1, \delta_2) = \frac{\delta_1+\delta_2}{1+\delta_1\delta_2}$$
$$\mathtt{CHK}\,(\delta_1, \delta_2) = \delta_1\delta_2.$$

◄ **Signed log-likelihood difference (SLLD)** ►

Definition: $\Delta(p_0, p_1) = \mathtt{sqn}(p_1 - p_0)\ln|p_1 - p_0|$.

$$\mathtt{VAR}\,(\Delta_1, \Delta_2) = \begin{cases} s\ln\left(\frac{\cosh((|\Delta_1|+|\Delta_2|)/2)}{\cosh((|\Delta_1|-|\Delta_2|)/2)}\right), \\ \quad \mathtt{if}\ \mathtt{sgn}(\Delta_1) = \mathtt{sgn}(\Delta_2) = s. \\ s\cdot\mathtt{sgn}(|\Delta_1| - |\Delta_2|)\ln\left(\frac{\sinh((|\Delta_1|+|\Delta_2|)/2)}{\sinh((|\Delta_1|-|\Delta_2|)/2)}\right), \\ \quad \mathtt{if}\ \mathtt{sgn}(\Delta_1) = -\mathtt{sgn}(\Delta_2) = -s. \end{cases}$$
$$\mathtt{CHK}\,(\Delta_1, \Delta_2) = \mathtt{sgn}(\Delta_1)\mathtt{sgn}(\Delta_2)(|\Delta_1| + |\Delta_2|).$$

- In the LLR domain, we observe that for $x \gg 1$
$$\ln(\cosh(x)) \approx |x| - \ln(2).$$
Thus, an approximation to the $\mathtt{CHK}$ function is
$$\begin{aligned}\mathtt{CHK}(\Lambda_1, \Lambda_2) &\approx |(\Lambda_1 + \Lambda_2)/2| - |(\Lambda_1 - \Lambda_2)/2| \\ &= \mathtt{sgn}(\Lambda_1)\mathtt{sgn}(\Lambda_2)\min(|\Lambda_1|, |\Lambda_2|)\end{aligned}$$
which turns out to be precisely the min-sum update rule.
- By applying the equivalence between factor graphs illustrated in Figure 18, it is easy to extend these formulas to cases where variable nodes or check nodes have degree larger than three.

- In particular, we may extend the $\mathtt{VAR}$ and $\mathtt{CHK}$ functions to more than two arguments via the relations
$$\mathtt{VAR}(x_1, x_2, \ldots, x_n) = \mathtt{VAR}(x_1, \mathtt{VAR}(x_2, \ldots, x_n))$$
$$\mathtt{CHK}(x_1, x_2, \ldots, x_n) = \mathtt{CHK}(x_1, \mathtt{CHK}(x_2, \ldots, x_n))$$
Of course, there are other alternatives, corresponding to the various binary trees with n leaf vertices.
- For example, when $n = 4$ we may compute $\mathtt{VAR}(x_1, x_2, x_3, x_4)$ as
$$\mathtt{VAR}(x_1, x_2, x_3, x_4) = \mathtt{VAR}(\mathtt{VAR}(x_1, x_2), \mathtt{VAR}(x_3, x_4))$$
which would have better time complexity in a parallel implementation than a computation based on above equations.

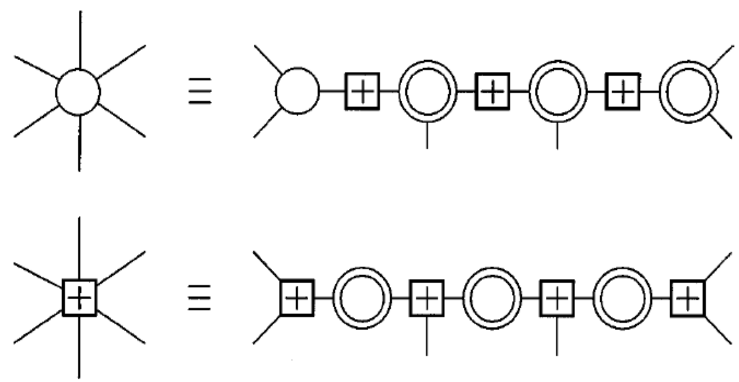


Figure 18: Transforming variable and check nodes of high degree to multiple nodes of degree three.

Iterative probability propagation

- Let us now study the more complex factor graph of the $[8,4]$ extended Hamming code. This code has the systematic parity-check matrix

$$H_{[8,4]} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The code has 4 information bits, which we associate with variable nodes $U_1, \ldots, U_4$; it also has 4 parity bits, which are associated with the variable nodes $X_5, \ldots, X_8$.

- There are also 8 observed output symbols, nodes $Y_1, \ldots, Y_8$, which are the noisy received variables $U_1, \ldots, X_8$.
- The function nodes $V_1, \ldots, V_4$ are the four parity-check equations described by above matrix, given as boolean truth functions.
- The function nodes $V_1, \ldots, V_4$ are the four parity-check equations, given as boolean truth functions.
  For example,

$$\begin{aligned} V_1 &: \quad U_1 \oplus U_2 \oplus U_3 \oplus X_5 = 0 \\ V_2 &: \quad U_1 \oplus U_2 \oplus U_4 \oplus X_6 = 0 \\ V_3 &: \quad U_1 \oplus U_3 \oplus U_4 \oplus X_7 = 0 \\ V_4 &: \quad U_2 \oplus U_3 \oplus U_4 \oplus X_8 = 0 \end{aligned}$$

- The complete factor graph of the parity-check matrix representation of this code is shown in Figure 7(a).

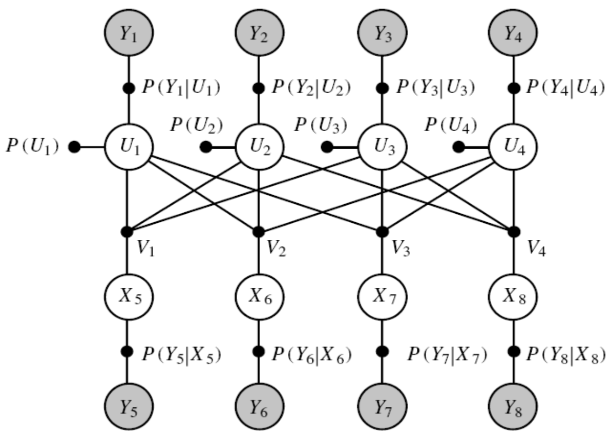


Figure 7(a): [8,4] Hamming code factor graph representation.

- This network has loops (*e.g.*, $U_1 - V_1 - U_2 - V_2 - U_1$) and thus the sum-product algorithm has no well-defined forward-backward schedule, but many possible schedules of passing messages between the nodes.
- A sensible message passing schedule will involve all nodes which are not observation nodes in a single sweep, called an iteration.
- Such iterations can then be repeated until a desired result is achieved. The messages are passed exactly according to the rules of the sum-product algorithm.

Turbo Codes

Coding and Communication Laboratory

Dept. of Electrical Engineering,
National Chung Hsing University

- **Chapter 10: Turbo Codes**
  1. Introduction
  2. Turbo code encoder
  3. Iterative decoding of turbo codes

Reference

1. Lin, Error Control Coding
   - chapter 16

Introduction

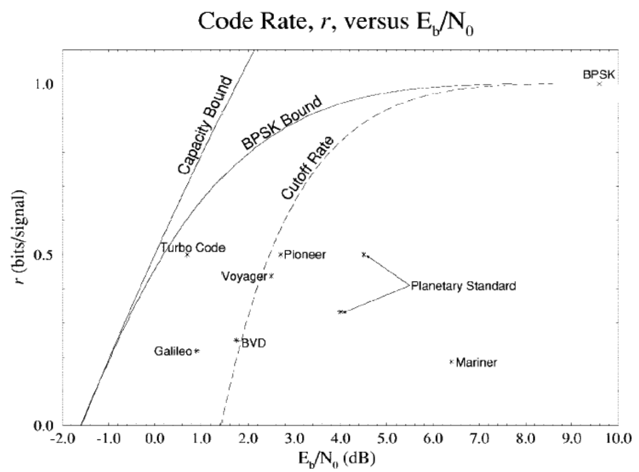
What are turbo codes?

- Turbo codes, which a class of error correcting codes is, are introduced by Berrou and Glavieux in ICC93.
- Turbo codes can come closer to approaching Shannon's limit than any other class of error correcting codes.
- Turbo codes achieve their remarkable performance with relatively low complexity encoding and decoding algorithms.

- A firm understanding of convolutional codes is an important prerequisite to the understanding of turbo codes.
- Tow fundamental ideas of Turbo code:

```
Encoder:  It produce a codeword with randomlike
          properties.
Decoder:  It make use of soft-output values and
          iterative decoding.
```


Power efficiency of existing standards



Turbo code encoder

Turbo code encoder

- The fundamental turbo code encoder:
  Two identical recursive systematic convolutional (RSC) codes with parallel concatenation.
- An RSC encoder[a] is termed a component encoder.
- The two component encoders are separated by an interleaver.

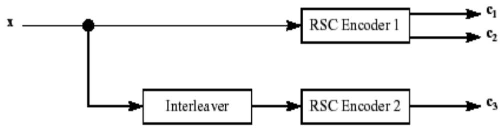


Figure A: Fundamental Turbo Code Encoder ($r = \frac{1}{3}$)

[a] In general, an RSC encoder is typically $r = \frac{1}{2}$.

- To achieve performance close to the Shannon limit, the information block length (interleaver size) is chosen to be very large, usually at least several thousand bits.
- RSC codes, generated by systematic feedback encoders, give much better performance than nonrecursive systematic convolutional codes, that is, feedforward encoders.
- Because only the ordering of the bits changed by the interleaver, the sequence that enters the second RSC encoder has the same weight as the sequence x that enters the first encoder.

Turbo codes suffer from two disadvantages:

1. A large decoding delay, owing to the large block lengths and many iterations of decoding required for near–capacity performance.
2. It significantly weakened performance at BERs below 10^{-5} owing to the fact that the codes have a relatively poor minimum distance, which manifests itself at very low BERs.

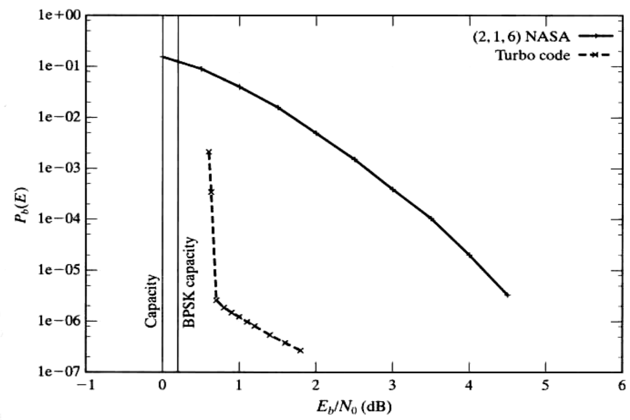


Figure A-1: Performance comparison of convolutional codes and turbo codes.

Recursive systematic convolutional (RSC) encoder

- The RSC encoder:
  The conventional convolutional encoder by feeding back one of its encoded outputs to its input.
- **Example**. Consider the conventional convolutional encoder in which

Generator sequence	$g_1 = [111]\,, g_2 = [101]$
Compact form	$G = [g_1, g_2]$

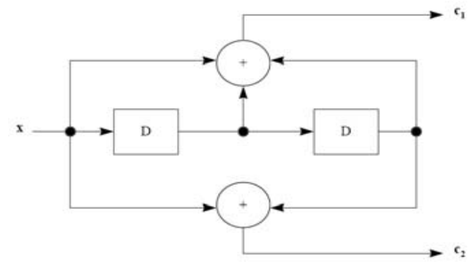


Figure B: Conventional convolutional encoder with $r = \frac{1}{2}$ and $K = 3$

- The RSC encoder of the conventional convolutional encoder:

$$G = [1, g_2/g_1]$$

  where the first output is fed back to the input.
- In the above representation:

1	the systematic output
g_2	the feed forward output
g_1	the feedback to the input of the RSC encoder

- Figure C shows the resulting RSC encoder.

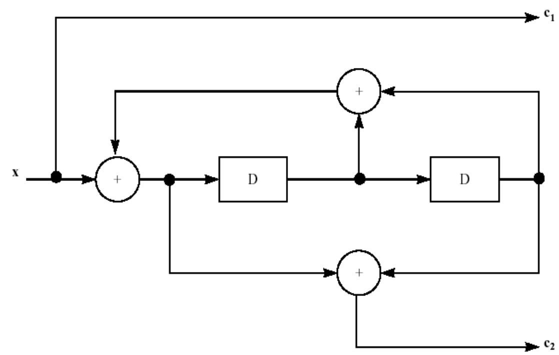


Figure C: The RSC encoder obtained from figure B with $r = \frac{1}{2}$ and $K = 3$.

Trellis termination

- For the conventional encoder, the trellis is terminated by inserting $m = K - 1$ additional zero bits after the input sequence. These additional bits drive the conventional convolutional encoder to the all-zero state (Trellis termination). However, this strategy is not possible for the RSC encoder due to the feedback.
- Convolutional encoder are time-invariant, and it is this property that accounts for the relatively large numbers of low-weight codewords in terminated convolutional codes.
- Figure D shows a simple strategy that has been developed in [a] which overcomes this problem.

[a] Divsalar, D. and Pollara, F., "Turbo Codes for Deep-Space Communications, " JPL TDA Progress Report 42-120, Feb. 15, 1995.

For encoding the input sequence, the switch is turned on to position A and for terminating the trellis, the switch is turned on to position B.

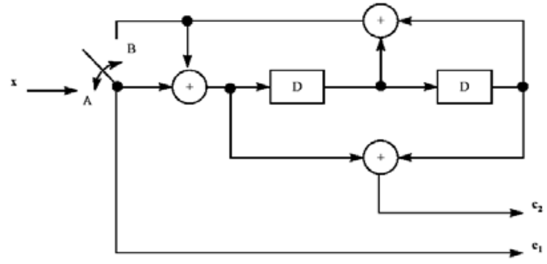


Figure D: Trellis termination strategy for RSC encoder

Recursive and nonrecursive convolutional encoders

- **Example**. Figure E shows a simple nonrecursive convolution encoder with generator sequence $g1 = [11]$ and $g2 = [10]$.

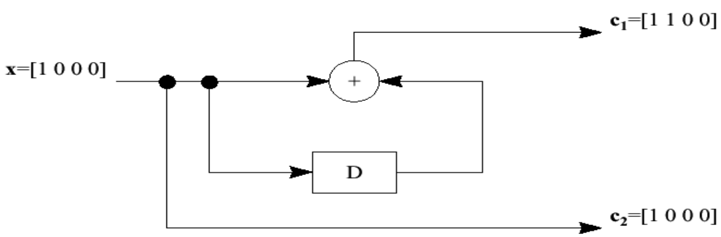


Figure E: Nonrecursive $r = \frac{1}{2}$ and $K = 2$ convolutional encoder with input and output sequences.

- **Example**. Figure F shows the equivalent recursive convolutional encoder of Figure E with $G = \left[1, \frac{g_2}{g_1}\right]$.

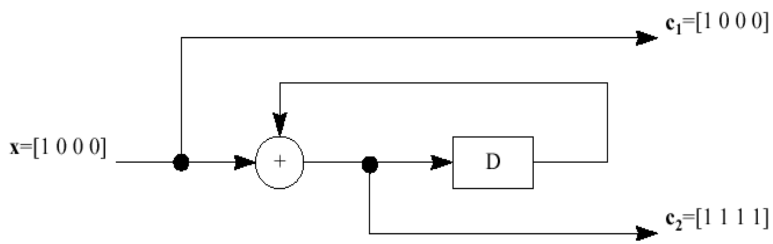


Figure F: Recursive $r = \frac{1}{2}$ and $K = 2$ convolutional encoder of Figure E with input and output sequences.

- Compare Figure E with Figure F:

The nonrecursive encoder	output codeword with weight of 3
The recursive encoder	output codeword with weight of 5

- State diagram:

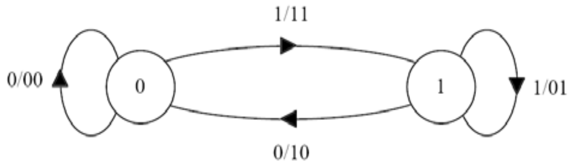


Figure G-1: State diagram of the nonrecursive encoder in Figure E.

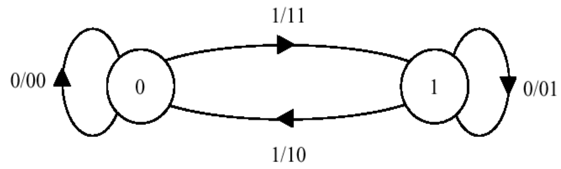


Figure G-2: State diagram of the recursive encoder in Figure F.

- A recursive convolutional encoder tends to produce codewords with increased weight relative to a nonrecursive encoder. This results in fewer codewords with lower weights and this leads to better error performance.
- For turbo codes, the main purpose of implementing RSC encoders as component encoders is to utilize the recursive nature of the encoders and not the fact that the encoders are systematic.

- Clearly, the state diagrams of the encoders are very similar.
- The transfer function of Figure G-1 and Figure G-2 :

$$T(D) = \frac{D^3}{1-D}$$ `Where N and J are neglected.`

- The two codes have the same minimum free distance and can be described by the same trellis structure.
- These two codes have different bit error rates. This is due to the fact that BER depends on the input–output correspondence of the encoders.[b]
- It has been shown that the BER for a recursive convolutional code is lower than that of the corresponding nonrecursive convolutional code at low signal-to-noise ratios E_b/N_0.[c]

---

[b] Benedetto, S., and Montorsi, G., "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes," IEEE Transactions on Information Theory, Vol. 42, No. 2, pp. 409-428, March 1996.

[c] Berrou, C., and Glavieux, A., "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," IEEE Transactions on Communications, Vol. 44, No. 10, pp. 1261-1271, Oct. 10, 1996.

Concatenation of codes

- A concatenated code is composed of two separate codes that combined to form a large code.
- There are two types of concatenation:
  - Serial concatenation
  - Parallel concatenation

- The total code rate for serial concatenation is

$$r_{\texttt{tot}} = \frac{k_1 k_2}{n_1 n_2}$$

which is equal to the product of the two code rates.

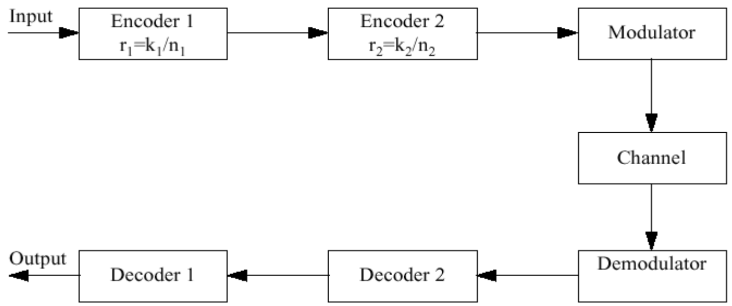


Figure H: Serial concatenated code

- The total code rate for parallel concatenation is:

$$r_{\texttt{tot}} = \frac{k}{n_1 + n_2}$$

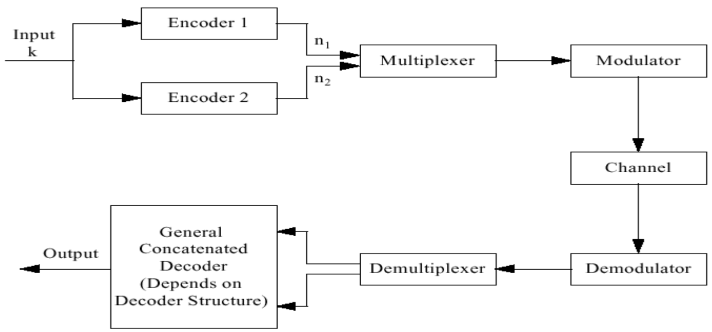


Figure I: Parallel concatenated code.

- For serial and parallel concatenation schemes:
  An interleaver is often between the encoders to improve burst error correction capacity or to increase the randomness of the code.
- Turbo codes use the parallel concatenated encoding scheme. However, the turbo code decoder is based on the serial concatenated decoding scheme.
- The serial concatenated decoders are used because they perform better than the parallel concatenated decoding scheme due to the fact that the serial concatenation scheme has the ability to share information between the concatenated decoders whereas the decoders for the parallel concatenation scheme are primarily decoding independently.

Interleaver design

- The interleaver is used to provide randomness to the input sequences.
- Also, it is used to increase the weights of the codewords as shown in Figure J.

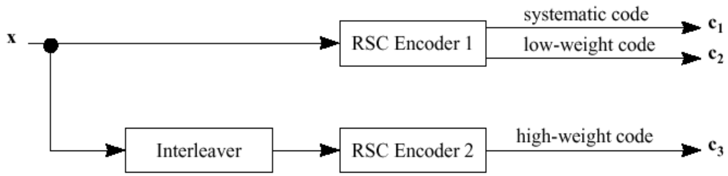


Figure J: The interleaver increases the code weight for RSC Encoder 2 as compared to RSC Encoder 1.

- Form Figure K, the input sequence x_i produces output sequences c_{1i} and c_{2i} respectively. The input sequence x_1 and x_2 are different permuted sequences of x_0.

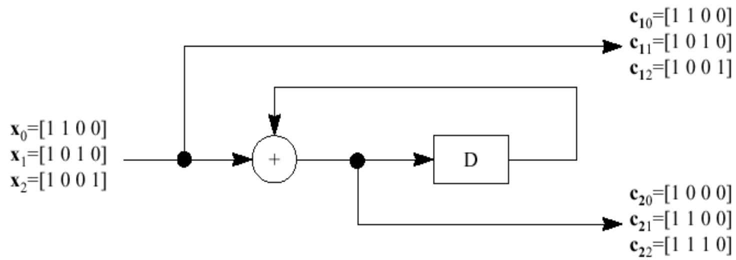


Figure K: An illustrative example of an interleaver's capability.

Table: Input an Output Sequences for Encoder in Figure K.

	Input Sequence x_i	Output Sequence c_{1i}	Output Sequence c_{2i}	Codeword Weight i
$i=0$	1100	1100	1000	3
$i=1$	1010	1010	1100	4
$i=2$	1001	1001	1110	5

- The interlever affect the performance of turbo codes because it directly affects the distance properties of the code.

Block interleaver

- The block interleaver is the most commonly used interleaver in communication system.
- It writes in column wise from top to bottom and left to right and reads out row wise from left to right and top to bottom.

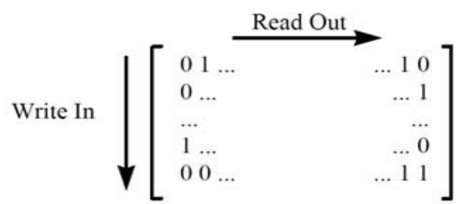


Figure L: Block interleaver.

Random (Pseudo-Random) interleaver

- The random interleaver uses a fixed random permutation and maps the input sequence according to the permutation order.
- The length of the input sequence is assumed to be L.
- The best interleaver reorder the bits in a pseudo-random manner. Conventional block (row-column) interleavers do not perform well in turbo codes, except at relatively short block lengths.

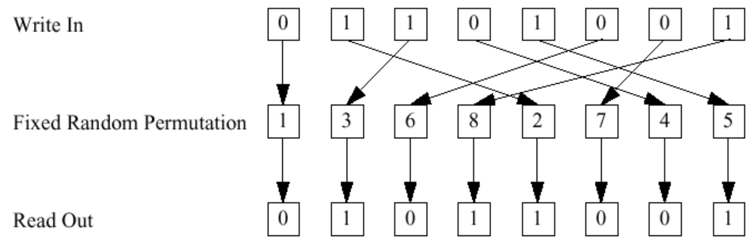


Figure M: A random (pseudo-random) interleaver with $L = 8$.

Circular-Shifting interleaver

- The permutation $\mathbf{p}$ of the circular-shifting interleaver is defined by

$$p(i) = (ai + s) \mod L$$

satisfying $a < L$, a is relatively prime to L, and $s < L$ where i is the index, a is the step size, and s is the offset.

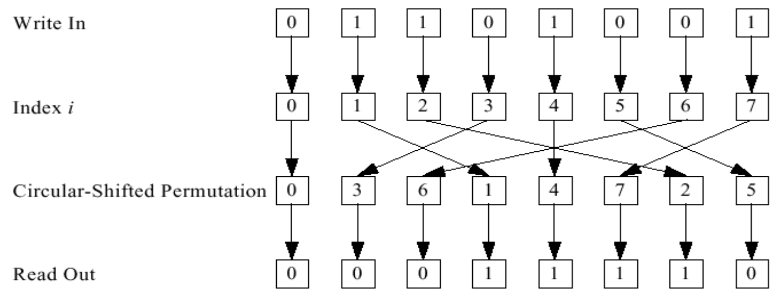
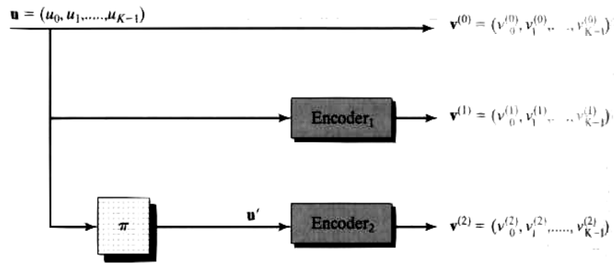


Figure N: A circular-shifting interleaver with $L = 8$, $a = 3$, $s = 0$.

Iterative decoding of turbo codes

The notation of turbo code enocder

- The information sequence (including termination bits) is considered to a block of length $K = K^* + v$ and is represented by the vector $\mathbf{u} = (u_0, u_1, \ldots, u_{K-1})$.



- Because encoding is systematic the information sequence $\mathbf{u}$ is the first transmitted sequence; that is

$$\mathbf{u} = \mathbf{v}^{(0)} = (v_0^{(0)}, v_1^{(0)}, \ldots, v_{K-1}^{(0)}).$$

- The first encoder generates the parity sequence

$$\mathbf{v}^{(1)} = (v_0^{(1)}, v_1^{(1)}, \ldots, v_{K-1}^{(1)}).$$

- The parity sequence by the second encoder [a] is represented as

$$\mathbf{v}^{(2)} = (v_0^{(2)}, v_1^{(2)}, \ldots, v_{K-1}^{(2)}).$$

- The final transmitted sequence (codeword) is given by the vector

$$v = (v_0^{(0)} v_0^{(1)} v_0^{(2)}, v_1^{(0)} v_1^{(1)} v_1^{(2)}, \ldots, v_{K-1}^{(0)} v_{K-1}^{(1)} v_{K-1}^{(2)})$$

[a] The second encoder may or may not be transmitted

The basic structure of an iterative turbo decoder

- The basic structure of an iterative turbo decoder is shown in Figure O. (We assume here a rate $R = 1/3$ parallel concatenated code without puncturing.)

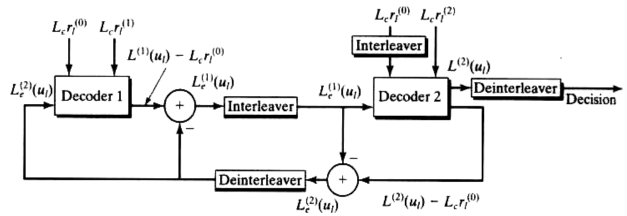


Figure O: Basic structure of an iterative turbo decoder.

- At each time unit l, three output values are received from the channel, one for the information bit $u_l = v_l^{(0)}$, denoted by $r_l^{(0)}$, and two for the parity bits $v_l^{(1)}$ and $v_l^{(2)}$, denote by $r_l^{(1)}$ and $r_l^{(2)}$, and the $3K$-dimensional received vector is denoted by

$$r = (r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, \ldots, r_{K-1}^{(0)} r_{K-1}^{(1)} r_{K-1}^{(2)})$$

- Let each transmitted bit represented using the mapping

$$0 \rightarrow -1 \texttt{ and } 1 \rightarrow +1.$$

- For an AWGN channel with unquantized (soft) outputs, we define the log-likelihood ratio (L-value) $L(v_l^{(0)}|r_l^{(0)}) = L(u_l|r_l^{(0)})$ (before decoding) of a transmitted information bit u_l given the received value $r_l^{(0)}$ as

$$\begin{aligned} L(u_l|r_l^{(0)}) &= \ln \frac{P(u_l=+1|r_l^{(0)})}{P(u_l=-1|r_l^{(0)})} \\ &= \ln \frac{P(r_l^{(0)}|u_l=+1)P(u_l=+1)}{P(r_l^{(0)}|u_l=-1)P(u_l=-1)} \\ &= \ln \frac{P(r_l^{(0)}|u_l=+1)}{P(r_l^{(0)}|u_l=-1)} + \ln \frac{P(u_l=+1)}{P(u_l=-1)} \\ &= \ln \frac{e^{-(E_s/N_0)(r_l^{(0)}-1)^2}}{e^{-(E_s/N_0)(r_l^{(0)}+1)^2}} + \ln \frac{P(u_l=+1)}{P(u_l=-1)} \end{aligned}$$

where E_s/N_0 is the channel SNR, and u_l and $r_l^{(0)}$ have both been normalized by a factor of $\sqrt{E_s}$.

- This equation simplifies to

$$\begin{aligned} L(u_l|r_l^{(0)}) &= -\frac{E_s}{N_0}\left\{(r_l^{(0)}-1)^2 - (r_l^{(0)}+1)^2\right\} + \ln \frac{P(u_l=+1)}{P(u_l=-1)} \\ &= \frac{E_s}{N_0} r_l^{(0)} + \ln \frac{P(u_l=+1)}{P(u_l=-1)} \\ &= L_c r_l^{(0)} + L_a(u_l), \end{aligned}$$

where $L_c = 4(E_s/N_0)$ is the channel reliability factor, and $L_a(u_l)$ is the a priori L-value of the bit u_l.

- In the case of a transmitted parity bit $v_l^{(j)}$, given the received value $r_l^{(j)}, j = 1, 2$, the L-value (before decoding) is given by

$$L(v_l^{(j)}|r_l^{(j)}) = L_c r_l^{(j)} + L_a(v_l^{(j)}) = L_c r_l^{(j)},\ j = 1, 2,$$

- In a linear block code with equally likely information bits, the parity bits are also equally likely to be $+1$ or -1, and thus the a priori L–values of the parity bits are 0; that is,

$$L_a(v_l^{(j)} = \ln \frac{P(v_l^{(j)})}{P(v_l^{(j)} = -1)} = 0,\ j = 1, 2.$$

- The received soft channel L-valued $L_c r_l^{(0)}$ for u_l and $L_c r_l^{(1)}$ for $v_l^{(1)}$ enter decoder 1, and the (properly interleaved) received soft channel L–valued $L_c r_l^{(2)}$ for $v_l^{(2)}$ enter decoder 2.

The output of decoder 1 contains two terms:

1. $L^{(1)}(u_l) = \ln\left[P(u_l = +1/\mathbf{r}_1, \mathbf{L}_a)/P(u_l = -1|\mathbf{r}_1, \mathbf{L}_a)\right]$, the a posteriori L-value (after decoding) of each information bit produced by decoder 1 given the (partial) received vector $\mathbf{r}_1 \triangleq \left[r_0^{(0)}r_0^{(1)}, r_1^{(0)}r_1^{(1)}, \ldots, r_{K-1}^{(0)}r_{K-1}^{(1)}\right]$ and the a priori input vector $\mathbf{L}_a^{(1)} \triangleq \left[L_a^{(1)}(u_0), L_a^{(1)}(u_1), \ldots, L_a^{(1)}(u_{K-1})\right]$ for decoder 1, and
2. $L_e^{(1)}(u_l) = L^{(1)}(u_l) - \left[L_c r_l^{(0)} + L_e^{(2)}(u_l)\right]$, the extrinsic a posteriori L-value (after decoding) associated with each information bit produced by decoder 1, which, after interleaving, is passed to the input of decoder 2 as the a priori value $L_a^{(2)}(u_l)$.

- Subtracting the term in brackets, namely, $L_c r_l^{(0)} + L_e^{(2)}(u_l)$, removes the effect of the current information bit u_l from $L^{(1)}(u_l)$, leaving only the effect of the parity constraint, thus providing an independent estimate of the information bit u_l to decoder 2 in addition to the received soft channel L-values at time l.

Similarly, the output of decoder 2 contains two terms:

1. $L^{(2)}(u_l) = \ln\left[P(u_l = +1|\mathbf{r}_2, \mathbf{L}_a^{(2)})/P(u_l = -1|r_2, \mathbf{L}_a^{(2)})\right]$, where $\mathbf{r}_2$ is the (partial) received vector and $L_a^{(2)}$ the a priori input vector for decoder 2, and
2. $L_e^{(2)}(u_l) = L^{(2)}(u_l) - \left[L_c r_l^{(0)} + L_e^{(1)}\right]$, and the extrinsic a posteriori L–values $L_e^{(2)}(u_l)$ produced by decoder 2, after deinterleaving, are passed back to the input of decoder 1 as the a priori values $L_a^{(1)}(u_l)$.

Iterative decoding using the log-MAP algorithm

- **Example.** Consider the parallel concatenated convolutional code (PCCC) formed by using the 2–state $(2, 1, 1)$ systematic recursive convolutional code (SRCC) with generator matrix

$$G(D) = [1 \ \frac{1}{1+D}]$$

as the constituent code. A block diagram of the encoder is shown in Figure P(a).

  – Also consider an input sequence of length $K = 4$, including one termination bit, along with a 2×2 block (row–column) interleaver, resulting in a $(12, 3)$ PCCC with overall rate $R = 1/4$.

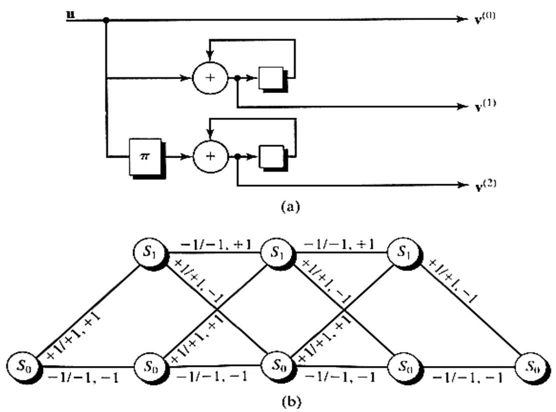


Figure P: (a) A 2-state turbo encoder and (b) the decoding trellis for $(2, 1, 1)$ constituent code with $K = 4$.

- The length $K = 4$ decoding trellis for the component code is show in Figure P(b), where the branches are labeled using the mapping $0 \to -1$ and $1 \to +1$.
- The input block is given by the vector $\mathbf{u} = [u_0, u_1, u_2, u_3]$, the interleaved input block is $u' = [[u'_0, u'_1, u'_2, u'_3] = [u_0, u_1, u_2, u_3]$, the parity vector for the first component code is given by $\mathbf{p}^{(1)} = \left[p_0^{(1)}, p_1^{(1)}, p_2^{(1)}, p_3^{(1)}\right]$, and the parity vector for the second component code is $\mathbf{p}^{(2)} = \left[p_0^{(2)}, p_1^{(2)}, p_2^{(2)}, p_3^{(2)}\right]$.
- We can represent the 12 transmitted bits in a rectangular array, as shown in Figure R(a), where the input vector $\mathbf{u}$ determines the parity vector $\mathbf{p}^{(1)}$ in the first two rows, and the interleaved input vector $\mathbf{u}'$ determines the parity vector $\mathbf{p}^{(2)}$ in the first two columns.

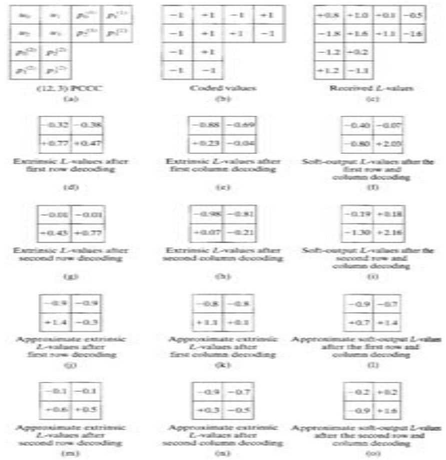


Figure R: Iterative decoding example for a (12,3) PCCC.

- For purposes of illustration, we assume the particular bit values shown in Figure R(b).
- We also assume a channel SNR of $E_s/N_0 = 1/4$ $(-6.02\texttt{dB})$, so that the received channel L-values corresponding to the received vector

$$\mathbf{r} = \left[r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, r_2^{(0)} r_2^{(1)} r_2^{(2)}, r_3^{(0)} r_3^{(1)} r_3^{(2)}\right]$$

are given by

$$L_c r_l^{(j)} = 4(\frac{E_s}{N_0}) r_l^{(j)} = r_l^{(j)},\ l = 0,1,2,3,\ j = 0,1,2.$$

Again for purposes of illustration, a set of particular received channel L-values is given in Figure R(c).

- In the first iteration of decoder 1 (row decoding), the log–MAP algorithm is applied to the trellis of the 2-state $(2, 1, 1)$ code shown in Figure P(b) to compute the a posteriori L-values $L^{(1)}(u_l)$ for each of the four input bits and the corresponding extrinsic a posteriori L-values $L_e^{(1)}(u_l)$ to pass to decoder 2 (the column decoder).
- Similarly, in the first iteration of decoder 2, the log-MAP algorithm uses the extrinsic posteriori L-values $L_e^{(1)}(u_l)$ received from decoder 1 as the a priori L-values, $L_a^{(2)}(u_l)$ to compute the a posteriori L-values $L^{(2)}(u_l)$ for each of the four input bits and the corresponding extrinsic a posteriori L-values $L_e^{(2)}(u_l)$ to pass back to decoder 1.

  Further decoding proceeds iteratively in this fashion.

- To simplify notation, we denote the transmitted vector as $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$, where $\mathbf{v}_l = (u_l, p_l)$, $l = 0, 1, 2, 3$, u_l is an input bit, and p_l is a parity bit.
- Similarly, the received vector is denoted as $\mathbf{r} = (\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)$, where $\mathbf{r}_l = (r_{u_l}, r_{p_l})$, $l = 0, 1, 2, 3$, r_{u_l} is the received symbol corresponding to the transmitted input bit u_l, and r_{p_l} is the received symbol corresponding to the transmitted parity bit p_l.

- An input bit a posteriori L-value is given by

$$\begin{aligned} L(u_l) &= \ln \frac{P(u_l=+1|)\mathbf{r}}{P(u_l=-1|)\mathbf{r}} \\ &= \ln \frac{\sum_{(s',s)\in\sum_l^+} p(s',s,\mathbf{r})}{\sum_{(s',s)\in\sum_l^-} p(s',s,\mathbf{r})} \end{aligned}$$

where

- s' represents a state at time l (denote by $s' \in \sigma_l$).
- s represents a state at time $l+1$ (denoted by $s \in \sigma_{l+1}$).
- The sums are over all state pairs (s', s) for which $u_l = +1$ or -1, respectively.

- We can write the joint probabilities $p(s', s, \mathbf{r})$ as

$$p(s', s, \mathbf{r}) = e^{\alpha_l^*(s') + \gamma_l^*(s',s) + \beta_{l+1}^*(s)},$$

where $\alpha_l^*(s')$, $\gamma_l^*(s', s)$, and $\beta_{l+1}^*(s)$ are the familiar log-domain $\alpha's$, $\gamma's$ and $\beta's$ of the MAP algorithm.

- For a continuous-output AWGN channel with an SNR of E_s/N_0, we can write the MAP decoding equations as

Branch metric: $r_l^*(s', s) = \frac{u_l L_a(u_l)}{2} + \frac{L_c}{2}\mathbf{r}_l \cdot \mathbf{v}_l$, $l = 0, 1, 2, 3$,

Forward metric: $\alpha_{l+1}^*(s) = \overset{*}{\max}_{s' \in \alpha_l} \left[\gamma_l^*(s', s) + \alpha_l^*(s')\right]$, $l = 0, 1, 2, 3$,

Backward metric: $\beta_l^*(s') = \overset{*}{\max}_{s \in \sigma_{l+1}} \left[\gamma_l^*(s', s) + \beta_{l+1}^*(s)\right]$

where the $\overset{*}{\max}$ function is defined in $\overset{*}{\max}(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x+y|})$ and the initial conditions are $\alpha_0^*(S_0) = \beta_4^*(S_0) = 0$, and $\alpha_0^*(S_1) = \beta_4^*(S_1) = -\infty$.

- Further simplifying the branch metric, we obtain

$$\begin{aligned} r_l^*(s', s) &= \frac{u_l L_a(u_l)}{2} + \frac{L_c}{2}(u_l r_{u_l} + p_l r_{p_l}) \\ &= \frac{u_l}{2}\left[L_a(u_l) + L_c r_{u_l}\right] + \frac{p_l}{2} L_c r_{p_l}, \ l = 0, 1, 2, 3. \end{aligned}$$

- We can express the a posteriori L-value of u_0 as

$$\begin{aligned} L(u_0) &= \ln p(s' = S_0, s = S_1, \mathbf{r}) - \ln p(s' = S_0, s = S_0, \mathbf{r}) \\ &= \left[\alpha_0^*(S_0) + \gamma_0^*(s' = S_0, s = S_1) + \beta_1^*(S_1)\right] - \\ &\quad \left[\alpha_0^*(S_0) + \gamma_0^*(s' = S_0, s = S_0) + \beta_1^*(S_1)\right] \\ &= \left\{+\tfrac{1}{2}\left[L_a(u_0) + L_c r_{u0}\right] + \tfrac{1}{2} L_c r_{p0} + \beta_1^*(S_1)\right\} - \\ &\quad \left\{-\tfrac{1}{2}\left[L_a(u_0) + L_c r_{u0}\right] + \tfrac{1}{2} L_c r_{p0} + \beta_1^*(S_0)\right\} \\ &= \left\{+\tfrac{1}{2}\left[L_a(u_0) + L_c r_{u0}\right]\right\} - \left\{-\tfrac{1}{2}\left[L_a(u_0) + L_c r_{u0}\right]\right\} + \\ &\quad \left\{+\tfrac{1}{2} L_c r_{p0} + \beta_1^*(S_1) + \tfrac{1}{2} L_c r_{p0} - \beta_1^*(S_0)\right\} \\ &= L_c r_{u0} + L_a(u_0) + L_e(u_0), \end{aligned}$$

where $L_e(u_0) \equiv L_c r_{p0} + \beta_1^*(S_1) - \beta_1^*(S_0)$ represents the extrinsic a posterior (output) L-value of u_0.

The final form of above equation illustrate clearly the three components of the a posteriori L-value of u_0 computed at the output of a log-MAP decoder:

- $L_c r_{u0}$: the received channel L-value corresponding to bit u_0, which was part of the decoder input.
- $L_a(u_0)$: the a priori L-value of u_0, which was also part of the decoder input. Expect for the first iteration of decoder 1, this term equals the extrinsic a posteriori L-value of u_0 received from the output of the other decoder.
- $L_e(u_0)$: the extrinsic part of the a posteriori L-value of u_0, which dose not depend on $L_c r_{u0}$ or $L_a(u_0)$. This term is then sent to the other decoder as its a priori input.

- We now proceed in a similar manner to compute the a posteriori L-value of bit u_1.
- We see from Figure P(b) that in this case there are two terms in each of the sums in $L(u_l) = \ln \frac{\sum_{(s',s)\in\Sigma_l^+} p(s',s,\mathbf{r})}{\sum_{(s',s)\in\Sigma_l^-} p(s',s,\mathbf{r})}$, because at this time there are two $+1$ and two -1 transitions in the trellis diagram.

$$
\begin{aligned}
L(u_1) &= \ln\left[p(s'=S_0, s=S_1, \mathbf{r}) + p(s'=S_1, s=S_0, \mathbf{r})\right] - \\
&\quad \ln\left[p(s'=S_0, s=S_0, \mathbf{r}) + p(s'=S_1, s=S_1, \mathbf{r})\right] \\
&= \overset{*}{\max}\left\{\left[\alpha_1^*(S_0) + \gamma_1^*(s'=S_0, s=S_1) + \beta_2^*(S_1)\right],\right. \\
&\quad \left.\left[\alpha_1^*(S_1) + \gamma_1^*(s'=S_1, s=S_0) + \beta_2^*(S_0)\right]\right\} \\
&\quad - \overset{*}{\max}\left\{\left[\alpha_1^*(S_0) + \gamma_1^*(s'=S_0, s=S_0) + \beta_2^*(S_0)\right],\right. \\
&\quad \left.\left[\alpha_1^*(S_1) + \gamma_1^*(s'=S_1, s=S_1) + \beta_2^*(S_1)\right]\right\} \\
&= \overset{*}{\max}\left\{\left(+\frac{1}{2}\left[L_a(u_1) + L_c r_{u1}\right] + \frac{1}{2}L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_1)\right),\right. \\
&\quad \left.\left(+\tfrac{1}{2}\left[L_a(u_1) + L_c r_{u1}\right] - \tfrac{1}{2}L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_0)\right)\right\} \\
&\quad - \overset{*}{\max}\left\{\left(+\frac{1}{2}\left[L_a(u_1) + L_c r_{u1}\right] + \frac{1}{2}L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_0)\right),\right. \\
&\quad \left.\left(+\tfrac{1}{2}\left[L_a(u_1) + L_c r_{u1}\right] - \tfrac{1}{2}L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_1)\right)\right\} \\
&= \left\{+\tfrac{1}{2}\left[L_a(u_1) + L_c r_{u1}\right]\right\} - \left\{-\tfrac{1}{2}\left[L_a(u_1) + L_c r_{u1}\right]\right\} \\
&\quad + \overset{*}{\max}\left\{\left[+\frac{1}{2}L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_1)\right], \left[-\frac{1}{2}L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_0)\right]\right\} \\
&\quad - \overset{*}{\max}\left\{\left[+\frac{1}{2}L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_0)\right], \left[-\frac{1}{2}L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_1)\right]\right\} \\
&= L_c r_{u1} + L_a(u_1) + L_e(u_1)
\end{aligned}
$$

$$\overset{*}{\max}(w+x, w+y) \equiv w + \overset{*}{\max}(x, y)$$

- Continuing, we can use the same procedure to compute the a posteriori L-values of bits u_2 and u_3 as

$$L(u_2) = L_c r_{u2} + L_a(u_2) + L_e(u_2),$$

where

$$
\Rightarrow \left\{
\begin{aligned}
L_e(u_2) = \overset{*}{\max}&\left\{\left[+\frac{1}{2}L_c r_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_1)\right], \left[-\frac{1}{2}L_c r_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_0)\right]\right\} \\
- \overset{*}{\max}&\left\{\left[+\frac{1}{2}L_c r_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_0)\right], \left[-\frac{1}{2}L_c r_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_1)\right]\right\}
\end{aligned}
\right.
$$

and

$$L(u_3) = L_c r_{u3} + L_a(u_3) + L_e(u_3),$$

where

$$
\begin{aligned}
L(u_3) &= \left[-\tfrac{1}{2}L_c r_{p3} + \alpha_3^*(S_1) + \beta_4^*(S_0)\right] - \left[-\tfrac{1}{2}L_c r_{p3} + \alpha_3^*(S_0) + \beta_4^*(S_0)\right] \\
&= \alpha_3^*(S_1) - \alpha_3^*(S_0)
\end{aligned}
$$

- We now need expressions for the terms $\alpha_1^*(S_0), \alpha_1^*(S_1), \alpha_2^*(S_0)$, $\alpha_2^*(S_1), \alpha_3^*(S_0), \alpha_3^*(S_1), \beta_1^*(S_0), \beta_1^*(S_1), \beta_2^*(S_0), \beta_2^*(S_1), \beta_3^*(S_0)$, and $\beta_3^*(S_1)$ that are used to calculate the extrinsic a posteriori L-values $L_e(u_l)$, $l = 0, 1, 2, 3$.

$$\begin{aligned}
\alpha_1^*(S_0) &= \tfrac{1}{2}(L_{u0} + L_{p0}) \\
\alpha_1^*(S_1) &= -\tfrac{1}{2}(L_{u0} + L_{p0}) \\
\alpha_2^*(S_0) &= \max^*\left\{\left[-\frac{1}{2}(L_{u1} + L_{p1}) + \alpha_1^*(S_0)\right], \left[+\frac{1}{2}(L_{u1} - L_{p1}) + \alpha_1^*(S_1)\right]\right\} \\
\alpha_2^*(S_1) &= \max^*\left\{\left[+\frac{1}{2}(L_{u1} + L_{p1}) + \alpha_1^*(S_0)\right], \left[-\frac{1}{2}(L_{u1} - L_{p1}) + \alpha_1^*(S_1)\right]\right\} \\
\alpha_3^*(S_0) &= \max^*\left\{\left[-\frac{1}{2}(L_{u2} + L_{p2}) + \alpha_2^*(S_0)\right], \left[+\frac{1}{2}(L_{u2} - L_{p2}) + \alpha_2^*(S_1)\right]\right\} \\
\alpha_3^*(S_1) &= \max^*\left\{\left[+\frac{1}{2}(L_{u2} + L_{p2}) + \alpha_2^*(S_0)\right], \left[-\frac{1}{2}(L_{u2} - L_{p2}) + \alpha_2^*(S_1)\right]\right\}
\end{aligned}$$

$$\begin{aligned}
\beta_3^*(S_0) &= -\tfrac{1}{2}(L_{u3} + L_{p3}) \\
\beta_1^*(S_1) &= +\tfrac{1}{2}(L_{u3} - L_{p3}) \\
\beta_2^*(S_0) &= \max^*\left\{\left[-\frac{1}{2}(L_{u2} + L_{p2}) + \beta_3^*(S_0)\right], \left[+\frac{1}{2}(L_{u2} - L_{p2}) + \beta_3^*(S_1)\right]\right\} \\
\beta_2^*(S_1) &= \max^*\left\{\left[+\frac{1}{2}(L_{u2} + L_{p2}) + \beta_3^*(S_0)\right], \left[-\frac{1}{2}(L_{u2} - L_{p2}) + \beta_3^*(S_1)\right]\right\} \\
\beta_1^*(S_0) &= \max^*\left\{\left[-\frac{1}{2}(L_{u1} + L_{p1}) + \beta_2^*(S_0)\right], \left[+\frac{1}{2}(L_{u1} - L_{p1}) + \beta_2^*(S_1)\right]\right\} \\
\beta_1^*(S_1) &= \max^*\left\{\left[+\frac{1}{2}(L_{u1} + L_{p1}) + \beta_2^*(S_0)\right], \left[-\frac{1}{2}(L_{u1} - L_{p1}) + \beta_2^*(S_1)\right]\right\}
\end{aligned}$$

- We note here that the a priori L-value of a parity bit $L_a(p_l) = 0$ for all l, since for a linear code with equally likely.

- We can write the extrinsic a posteriori L-values in terms of L_{u2} and L_{p2} as

$$\begin{aligned}
L_e(u_0) = {} & L_{p0} + \beta_1^*(S_1) - \beta_1^*(S_0), \\
L_e(u_1) = {} & \max^*\left\{\left[+\frac{1}{2}L_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_1)\right], \left[-\frac{1}{2}L_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_0)\right]\right\} - \\
& \max^*\left\{\left[-\frac{1}{2}L_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_0)\right], \left[+\frac{1}{2}L_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_1)\right]\right\} \\
L_e(u_2) = {} & \max^*\left\{\left[+\frac{1}{2}L_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_1)\right], \left[-\frac{1}{2}L_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_0)\right]\right\} - \\
& \max^*\left\{\left[-\frac{1}{2}L_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_0)\right], \left[+\frac{1}{2}L_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_1)\right]\right\}
\end{aligned}$$

and

$$L_e(u_3) = \alpha_3^*(S_1) - \alpha_3^*(S_0).$$

- The extrinsic L-value of bit u_l does not depend directly on either the received or a priori L-values of u_l.

Iterative decoding using the max-log-MAP algorithm

- **Example.** When the approximation $\max^*(x, y) \approx \max(x, y)$ is applied to the forward and backward recursions, we obtain for the first iteration of decoder 1

$$\begin{aligned}
\alpha_2^*(S_0) &\approx \max\{-0.70, 1.20\} = 1.20 \\
\alpha_2^*(S_1) &\approx \max\{-0.20, -0.30\} = -0.20 \\
\alpha_3^*(S_0) &\approx \max\left\{\left[-\frac{1}{2}(-1.8 + 1.1) + 1.20\right], \left[+\frac{1}{2}(-1.8 - 1.1) - 0.20\right]\right\} \\
&= \max\{1.55, -1.65\} = 1.55 \\
\alpha_3^*(S_1) &\approx \max\left\{\left[+\frac{1}{2}(-1.8 + 1.1) + 1.20\right], \left[-\frac{1}{2}(-1.8 - 1.1) - 0.20\right]\right\} \\
&= \max\{0.85, 1.25\} = 1.25
\end{aligned}$$

$$
\begin{aligned}
\beta_2^*(S_0) &\approx \max\{0.35, 1.25\} = 1.25 \\
\beta_2^*(S_1) &\approx \max\{-1.45, -3.05\} = -3.05 \\
\beta_1^*(S_0) &\approx \max\left\{\left[-\frac{1}{2}(1.0-0.5)+1.25\right], \left[+\frac{1}{2}(1.0-0.5)+3.05\right]\right\} \\
&= \max\{1.00, 3.30\} = 3.30 \\
\beta_1^*(S_1) &\approx \max\left\{\left[+\frac{1}{2}(1.0+0.5)+1.25\right], \left[-\frac{1}{2}(1.0+0.5)+3.05\right]\right\} \\
&= \max\{2.00, 2.30\} = 2.30 \\
L_e^{(1)}(u_0) &\approx 0.1 + 2.30 - 3.30 = -0.90 \\
L_e^{(1)}(u_0) &\approx \max\{[-0.25-0.45+3.05], [0.25, +0.45+1.25]\} \\
&\quad - \max\{[0.25-0.45+1.25], [-0.25+0.45+3.05]\} \\
&= \max\{2.35, 1.95\} - \max\{1.05, 3.25\} = 2.35 - 3.25 = -0.90,
\end{aligned}
$$

and, using similar calculations, we have

$$L_e^{(1)}(u_2) \approx +1.4 \text{ and } L_e^{(1)}(u_3) \approx -0.3$$

.

– Using these approximate extrinsic a posteriori L-values as a posteriori L-value as a priori L-values for decoder 2, and recalling that the roles of u_1 and u_2 are reversed for decoder 2, we obtain

$$
\begin{aligned}
\alpha_1^*(S_0) &= -\tfrac{1}{2}(0.8-0.9-1.2) = 0.65 \\
\alpha_1^*(S_1) &= +\tfrac{1}{2}(0.8-0.9-1.2) = -0.65 \\
\alpha_2^*(S_0) &\approx \max\left\{\left[-\tfrac{1}{2}(-1.8+1.4+1.2)+0.65\right], \left[+\tfrac{1}{2}(-1.8+1.4-1.2)-0.65\right]\right\} \\
&= \max\{0.25, -1.45\} = 0.25 \\
\alpha_2^*(S_1) &\approx \max\left\{\left[+\tfrac{1}{2}(-1.8+1.4+1.2)+0.65\right], \left[-\tfrac{1}{2}(-1.8+1.4-1.2)-0.65\right]\right\} \\
&= \max\{1.05, 0.15\} = 1.05 \\
\alpha_3^*(S_0) &\approx \max\left\{\left[-\tfrac{1}{2}(1.0-0.9+0.2)+0.25\right], \left[+\tfrac{1}{2}(1.0-0.9-0.2)+1.05\right]\right\} \\
&= \max\{0.10, 1.00\} = 1.00 \\
\alpha_3^*(S_1) &\approx \max\left\{\left[+\tfrac{1}{2}(1.0-0.9+0.2)+0.25\right], \left[-\tfrac{1}{2}(1.0-0.9-0.2)+1.05\right]\right\} \\
&= \max\{0.40, 1.10\} = 1.10
\end{aligned}
$$

$$
\begin{aligned}
\beta_3^*(S_0) &= -\tfrac{1}{2}(1.6-0.3-1.1) = -0.10 \\
\beta_3^*(S_1) &= +\tfrac{1}{2}(1.6-0.3+1.1) = 1.20 \\
\beta_2^*(S_0) &\approx \max\left\{\left[-\tfrac{1}{2}(1.0-0.9+0.2)-0.10\right], \left[+\tfrac{1}{2}(1.0-0.9+0.2)+1.20\right]\right\} \\
&= \max\{-0.25, 1.35\} = 1.35 \\
\beta_2^*(S_1) &\approx \max\left\{\left[+\tfrac{1}{2}(1.0-0.9-0.2)-0.10\right], \left[-\tfrac{1}{2}(1.0-0.9-0.2)+1.20\right]\right\} \\
&= \max\{-0.15, 1.25\} = 1.25 \\
\beta_1^*(S_0) &\approx \max\left\{\left[-\tfrac{1}{2}(-1.8+1.4+1.2)+1.35\right], \left[+\tfrac{1}{2}(-1.8+1.4+1.2)+1.25\right]\right\} \\
&= \max\{0.95, 1.65\} = 1.65 \\
\beta_1^*(S_1) &\approx \max\left\{\left[+\tfrac{1}{2}(-1.8+1.4-1.2)+1.35\right], \left[-\tfrac{1}{2}(-1.8+1.4-1.2)+1.25\right]\right\} \\
&= \max\{0.55, 2.05\} = 2.05 \\
L_e^{(2)}(u_0) &\approx -1.2 + 2.05 - 1.65 = -0.80 \\
L_e^{(2)}(u_2) &\approx \max\{[0.6+0.65+1.25], [-0.6-0.65+1.35]\} \\
&\quad - \max\{[-0.6+0.65+1.35], [0.6-0.65+1.25]\} \\
&\quad = \max\{2.5, 0.1\} - \max\{1.4, 1.2\} = 2.5 - 1.4 = 1.10
\end{aligned}
$$

and, using similar calculations, we have

$$L_e^{(2)}(u_1) \approx -0.8 \text{ and } L_e^{(2)}(u_3) \approx +0.1.$$

– We calculate the approximate a posteriori L-value of information bit u_0 after the first complete iteration of decoding as

$$L^{(2)}(u_0) = L_c r_{u_0} + L_a^{(2)}(u_0) + Le(u_0) \approx 0.8 - 0.9 - 0.8 = -0.9,$$

and we similar obtain the remaining approximate a posteriori L-values as $L^{(2)}(u_2) \approx +0.7$, $L^{(2)}(u_1) \approx -0.7$, and $L^{(2)}(u_3) \approx +1.4$.

Fundamental principle of turbo decoding

- We now summarize our discussion of iterative decoding using the log-MAP and Max-log-MAP algorithm:
  - The extrinsic a posteriori L-values are no longer strictly independent of the other terms after the first iteration of decoding, which causes the performance improvement from successive iterations to diminish over time.
  - The concept of iterative decoding is similar to negative feedback in control theory, in the sense that the extrinsic information from the output that is fed back to the input has the effect of amplifying the SNR at the input, leading to a stable system output.

- Decoding speed can be improved by a factor of 2 by allowing the two decoders to work in parallel. In this case, the a priori L-values for the first iteration of decoder 2 will be the same as for decoder 1 (normally equal to 0), and the extrinsic a posteriori L-values will then be exchanged at the same time prior to each succeeding iteration.
- After a sufficient number of iterations, the final decoding decision can be taken from the a posteriori L-values of either decoder, or form the average of these values, without noticeably affect performance.

- As noted earlier, the L-values of the parity bits remain constant throughout decoding. In serially concatenated iterative decoding systems, however, parity bits from the outer decoder enter the inner decoder, and thus the L-values of these parity bits must be updated during the iterations.
- The forgoing approach to iterative decoding is ineffective for nonsystematic constituent codes, since channel L-values for the information bits are not available as inputs to decoder 2; however, the iterative decoder of Figure O can be modified to decode PCCCs with nonsystematic component codes.

- As noted previously, better performance is normally achieved with pseudorandom interleavers, particularly for large block lengths, and the iterative decoding procedure remains the same.
- It is possible, however, particularly on very noisy channels, for the decoder to converge to the correct decision and then diverge again, or even to “oscillate” between correct and incorrect decision.

- Iterations can be stopped after some fixed number, typically in the range $10-20$ for most turbo codes, or stopping rules based on reliability statistics can be used to halt decoding.
- The Max-log-MAP algorithm is simpler to implement than the log-MAP algorithm; however, it typically suffers a performance degradation of about 0.5 dB.
- It can be shown that MAX-log-MAP algorithm is equivalent to the SOVA algorithm.

The stopping rules for iterative decoding

1. One method is based on the cross-entropy (CE) of the APP distributions at the outputs of the two decoders.
   - The cross-entropy $D(P||Q)$ of two joint probability distributions $P(\mathbf{u})$ and $Q(\mathbf{u})$, assume statistical independence of the bits in the vector $\mathbf{u}=[u_0,u_1,\ldots,u_{K-1}]$, is defined as

$$D(P||Q)=E_p\left\{\log\frac{P(\mathbf{u})}{Q(\mathbf{u})}\right\}=\sum_{l=0}^{K-1}E_p\left\{\log\frac{P(u_l)}{Q(u_l)}\right\}.$$

   where $E_p\{\cdot\}$ denote expectation with respect to the probability distribution $P(u_l)$.

- $D(P||Q)$ is a measure of the closeness of two distributions, and

  $D(P||Q)=0$ iff $P(u_l)=Q(u_l),\ u_l=\pm1,\ l=0,1,\ldots,K-1.$
- The CE stopping rule is based on the different between the a posteriori L-values after successive iterations at the outputs of the two decoders. For example, let

$$L^{(1)}_{(i)}(u_l)=L_c r_{u_l}+L^{(1)}_{a(i)}(u_l)+L^{(1)}_{e(i)}(u_l)$$

  represent the a posteriori L-value at the output decoder 1 after iteration i, and let

$$L^{(2)}_{(i)}(u_l)=L_c r_{u_l}+L^{(2)}_{a(i)}(u_l)+L^{(2)}_{e(i)}(u_l)$$

  represent the a posteriori L-value at the output decoder 2 after iteration i.

- Now, using the facts that $L^{(1)}_{a(i)}(u_l)=L^{(2)}_{e(i-1)}(u_l)$ and $L^{(2)}_{a(i)}(u_l)=L^{(1)}_{e(i)}(u_l)$, and letting $Q(u_l)$ and $P(u_l)$ represent the a posteriori probability distributions at the outputs of decoders 1 and 2, respectively, we can write

$$L^{(Q)}_{(i)}(u_l)=L_c r_{u_l}+L^{(P)}_{e(i-1)}(u_l)+L^{(Q)}_{e(i)}(u_l)$$

  and

$$L^{(P)}_{(i)}(u_l)=L_c r_{u_l}+L^{(Q)}_{e(i)}(u_l)+L^{(P)}_{e(i)}(u_l).$$

- We can write the difference in the two soft outputs as

$$L^{(P)}_{(i)}(u_l)-L^{(Q)}_{(i)}(u_l)=L^{(P)}_{e(i)}(u_l)-L^{(P)}_{e(i-1)}(u_l)\triangleq\Delta L^{(P)}_{e(i)}(u_l);$$

  that is, $\Delta L^{(P)}_{e(i)}(u_l)$ represents the difference in the extrinsic a posteriori L-values of decoder 2 in two successive iterations.

- We now compute the CE of the a posteriori probability distributions $P(u_l)$ and $Q(u_l)$ as follows:

$$E_p\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} = P(u_l=+1)\log\frac{P(u_l=+1)}{Q(u_l=+1)} + P(u_l=-1)\log\frac{P(u_l=-1)}{Q(u_l=-1)}$$

$$= \frac{e^{L_{(i)}^{(P)}(u_l)}}{1+e^{L_{(i)}^{(P)}(u_l)}}\log\frac{e^{L_{(i)}^{(P)}(u_l)}}{1+e^{L_{(i)}^{(P)}(u_l)}}\cdot\frac{1+e^{L_{(i)}^{(Q)}(u_l)}}{e^{L_{(i)}^{(Q)}(u_l)}} + \frac{e^{-L_{(i)}^{(P)}(u_l)}}{1+e^{-L_{(i)}^{(P)}(u_l)}}\log\frac{e^{-L_{(i)}^{(P)}(u_l)}}{1+e^{-L_{(i)}^{(P)}(u_l)}}\cdot\frac{1+e^{-L_{(i)}^{(Q)}(u_l)}}{e^{-L_{(i)}^{(Q)}(u_l)}},$$

where we have used expressions for the a posteriori distributions $P(u_l=\pm1)$ and $Q(u_l=\pm1)$ analogous to those given in $P(u_l=\pm1)=\frac{e^{\pm L_a(u_l)}}{\{1+e^{\pm L_a(u_l)}\}}$.

- The above equation can simplify as $E_p\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} = -\frac{\triangle L_{e(i)}^{(P)}(u_l)}{1+e^{L_{(i)}^{(P)}(u_l)}} + \log\frac{1+e^{-L_{(i)}^{(Q)}(u_l)}}{1+e^{-L_{(i)}^{(P)}(u_l)}}$.
- The hard decisions after iteration i, $\hat{u}_l^{(i)}$, satisfy

$$u_l^{(i)} = \mathtt{sgn}\left[L_{(i)}^{(P)}(u_l)\right] = \mathtt{sgn}\left[L_{(i)}^{(Q)}(u_l)\right].$$

- Using above equation and noting that

$$|L_{(i)}^{(P)}(u_l)| = \mathtt{sgn}\left[L_{(i)}^{(P)}(u_l)\right]L_{(i)}^{(P)}(u_l) = \hat{u}_l^{(i)}L_{(i)}^{(P)}(u_l)$$

and

$$|L_{(i)}^{(Q)}(u_l)| = \mathtt{sgn}\left[L_{(i)}^{(Q)}(u_l)\right]L_{(i)}^{(Q)}(u_l) = \hat{u}_l^{(i)}L_{(i)}^{(Q)}(u_l),$$

we can show that $E_p\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} = -\frac{\triangle L_{e(i)}^{(P)}(u_l)}{1+e^{L_{(i)}^{(P)}(u_l)}} + \log\frac{1+e^{-L_{(i)}^{(Q)}(u_l)}}{1+e^{-L_{(i)}^{(P)}(u_l)}}$ simplifies further to $E_p\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} \approx \frac{-\hat{u}_l^{(i)}\triangle L_{e(i)}^{(P)}(u_l)}{1+e^{|L_{(i)}^{(P)}(u_l)|}} + \log\frac{1+e^{-|L_{(i)}^{(Q)}(u_l)|}}{1+e^{-|L_{(i)}^{(P)}(u_l)|}}$.

- We now use the facts that once decoding has converged, the magnitudes of the a posteriori L-values are large; that is,

$$\left|L_{(i)}^{(P)}(u_l)\right| \gg 0 \;\mathtt{and}\; \left|L_{(i)}^{(Q)}(u_l)\right| \gg 0,$$

and that when x is large, e^{-x} is small, and

$$1+e^{-x}\approx 1 \;\mathtt{and}\; \log(1+e^{-x})\approx e^{-x}$$

- Applying these approximations to $E_p\left\{\log\frac{P(u_l)}{Q(u_l)}\right\}$, we can show that

$$E_p\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} \approx e^{-\left|L_{(i)}^{(Q)}(u_l)\right|}\left(1-e^{-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)}\cdot(1+\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l))\right)$$

- Noting that the magnitude of $\Delta L_{e(i)}^{(P)}(u_l)$ will be smaller than 1 when decoding converges, we can approximate the term $e^{-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)}$ using the first two terms of its series expansion as follows:

$$e^{-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)} \approx 1 - \hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l),$$

which leads to the simplified expression

$$\begin{aligned} E_p\left\{\log \frac{P(u_l)}{Q(u_l)}\right\} &\approx e^{-\left|L_{(i)}^{(Q)}(u_l)\right|}\left[\left(1-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)\right)\left(1+\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)\right)\right] \\ &= e^{-\left|L_{(i)}^{(Q)}(u_l)\right|}\left[\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)\right]^2 = \frac{\left|\Delta L_{e(i)}^{(P)}(u_l)\right|^2}{e^{\left|L_{(i)}^{(Q)}(u_l)\right|}} \end{aligned}$$

– We can write the CE of the probability distributions $P(\mathbf{u})$ and $Q(\mathbf{u})$ at iteration i as

$$\begin{aligned} D_{(i)}(P\|Q) &\triangleq E_p\left\{\log \frac{P(\mathbf{u})}{Q(\mathbf{u})}\right\} \\ &\approx \sum_{l=0}^{K-1} \frac{\left|\Delta L_{e(i)}^{(P)}(u_l)\right|^2}{e^{\left|L_{(i)}^{(Q)}(u_l)\right|}}, \end{aligned}$$

where we note that the statistical independence assumption does not hold exactly as the iterations proceed.

– We next define

$$T(i) \triangleq \frac{\left|\Delta L_{e(i)}^{(P)}(u_l)\right|^2}{e^{\left|L_{(i)}^{(Q)}(u_l)\right|}}$$

as the approximate value of the CE at iteration i. $T(i)$ can be computed after each iteration.

– Experience with computer simulations has shown that once convergence is achieved, $T(i)$ drops by a factor of $10^{-}2$ to 10^{-4} compared with its initial value, and thus it is reasonable to use

$$T(i) < 10^{-3}T(1)$$

as a stopping rule for iterative decoding.

2. Another approach to stopping the iterations in turbo decoding is to concatenate a high-rate outer cyclic code with an inner turbo code.

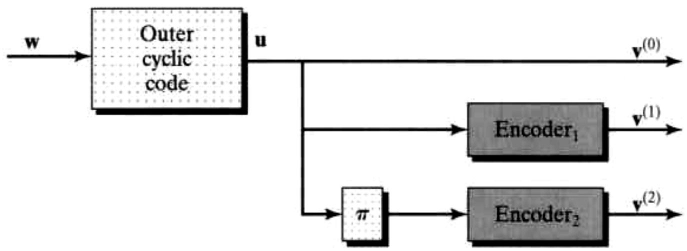


Figure : A concatenation of an outer cyclic code with an inner turbo code.

- After each iteration, the hard-decision output of the turbo decoder is used to check the syndrome of the cyclic code.
- If no errors are detected, decoding is assumed correct and the iterations are stopped.
- It is important to choose an outer code with a low undetected error probability, so that iterative decoding is not stopped prematurely.
- For this reason it is usually advisable not to check the syndrome of the outer code during the first few iterations, when the probability of undetected error may be larger than the probability that the turbo decoder is error free.

- This method of stopping the iterations is particularly effective for large block lengths, since in this case the rate of the outer code can be made very high, thus resulting in a negligible overall rate loss.
- For large block lengths, the foregoing idea can be extended to include outer codes, such as BCH codes, that can correct a small number of errors and still maintain a low undetected error probability.
- In this case, the iterations are stopped once the number of hard-decision errors at the output of the turbo decoder is within the error-correcting capability of the outer code.

- This method also provides a low word-error probability for the complete system; that is, the probability that the entire information block contains one or more decoding errors can be made very small.

LDPC Codes

Communication and Coding Laboratory

Dept. of Electrical Engineering,
National Chung Hsing University

Reference

1. (*)
   -
2. -
3. -

4. -
5. -
6. -
7. -
8. -

- **Chapter 11: LDPC Codes**
  1. Introduction
  2. A geometric construction of LDPC code
  3. EG-LDPC code
  4. PG-LDPC code
  5. Random LDPC code
  6. Decoding of LDPC code

Introduction

An LDPC code is defined as the null space of a parity-check matrix **H** that has the following structural properties:

1. Each row consists of ρ 1's.
2. Each column consists of γ 1's.
3. The number of 1's in common between ant two columns, denoted by λ, is no greater than 1; that is $\lambda = 0$ or 1.
4. Both ρ and γ are small compared with length of the code and the number of rowsin **H** [1,2].

- Properties (1) and (2) say that the parity check matrix **H** has constant row and column weights ρ and γ . Property (3) implies that no two rows of **H** have more than one 1 in common
- We define the density r of the parity-check matrix **H** as the ratio of the total number of 1's in **H** to the total number of entries in **H**. Then, we readily see that

$$r = \rho/n = \gamma/J$$

where J is number of rows in **H**.
- The LDPC code given by the definition is called a $(\gamma, \rho) - \textit{regular}$ LDPC code, If all the columns or all the rows of the parity check matrix **H** do not have the same weight, an LDPC code is then said to be irregular.

Ex: Consider the matrix **H** given below. Each column and each row of this matrix consist of four 1's, respectively. It can be checked easily that no two columns (or two rows) have more than one 1 ion common. The density of this matrix 1s 0.267. Therefore, it's is a low-density matrix. The null space of this matrix given a (15,7) LDPC code with a minimum distance of 5. It will be shown in a later section that this code is cyclic and is a BCH code.

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

Let k be a positive integer greater than 1. For a given choice of ρ and γ, Gallager gave the following construction of a class of linear codes specified by their parity-check matrices. Form a $k\gamma \times k\rho$ matrix $\mathbf{H}$ that consists of γ $k \times k\rho$ sub matrices, $\mathbf{H}_1$ $\mathbf{H}_2$ ... $\mathbf{H}_\gamma$. Each rows of submatrix has ρ 1's and each column of a submatrix contains a single 1. Therefore, each submatrix has a total of $k\rho$ 1's. For $1 \leq i \leq k$, the ith row of $\mathbf{H}_1$ contains all its ρ 1,s in colimns $(i-1)\rho + 1toi\rho$. The other submatrices are merely $columnpermutations$ $\mathbf{H}_1$. Then,

$$H = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \\ \vdots \\ \mathbf{H}_\gamma \end{bmatrix}$$

From the construction of $\mathbf{H}$, it is clear that:

1. No two rows in a submatrix of $\mathbf{H}$ have any 1-component in common.
2. No two columns in a submatrix of $\mathbf{H}$ have more than one 1 in common.

Because the total number of ones in $\mathbf{H}$ is $k\rho\gamma$ and the total number of entries in $\mathbf{H}$ is $k^2\rho\gamma$, the density of $\mathbf{H}$ is $1/k$. If k is chosen much greater than 1, $\mathbf{H}$ has a very small density and is a sparse matrix.

Consider an LDPC code C of length n specified by a $J \times n$ parity-check matrix $\mathbf{H}$. Let $\mathbf{h}_1$,$\mathbf{h}_2$,...,$\mathbf{h}_J$ denote the rows of $\mathbf{H}$ where:

$$\mathbf{h}_j = (h_{j,0}, h_{j,1}, \ldots, h_{j,n-1})$$

for $1 \leq j \leq J$. $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ is a codeword in C. Then, the inner product:

$$s_j = \mathbf{v} \cdot \mathbf{h} = \sum_{l=0}^{n-1} v_l h_{j,l} = 0$$

give a parity-check sum. There are a total of J such parity-check sums specified by the J rows of $\mathbf{H}$.

A code bit v_l is said to be checked by the sum $v \cdot h_j$ if $h_{j,l} = 1$. For $0 \leq l \leq n$, let $A_l = \{h_1^{(l)}, h_2^{(l)}, \ldots, h_\gamma^{(l)}\}$ denote the set of rows in $\mathbf{H}$ that check on the code bit v_l. For $1 \leq j \leq \gamma$, let

$$\mathbf{h}_j^{(l)} = (h_{j,0}^{(l)}, h_{j,1}^{(l)}, \ldots, h_{j,n-1}^{(l)})$$

Then, $h_{1,l}^{(l)} = h_{2,l}^{(l)} = \cdots = h_{\gamma,l}^{(l)} = 1$

Any error patten with $\lfloor\gamma/2\rfloor$ or fewer errors can be corrected. Consequently, the minimum distance d_{min} of the code is at least $\gamma+1$; that is $d_{min} \geq \gamma+1$ If γ is too small, the one-step majority-logic decoding of an LDPC code will give very poor error performance.

A geometric construction of LDPC code

Let $\mathbf{Q}$ be a finite geometry with n points and J lines has following properties:

1. Every limes consist of ρ points.
2. Every points lies on γ lines.
3. Two points are connected by one and only one line.
4. Two lines are either disjoint or they intersect one and only one point.

We denote the points and lines in $\mathbf{Q}$ with $\{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n\}$ and $\{L_1, L_2, \cdots, L_J\}$, respectively Let

$$\mathbf{v} = (v_1, v_2, \cdots, v_n)$$

be an n-tuple over $GF(2)$ whose components correspond to the n points of geometry $\mathbf{Q}$, where the ith component v_i corresponds to the ith point $\mathbf{p}_i$ of $\mathbf{Q}$. Let L be a line in $\mathbf{Q}$, we define a vector based on the points on L as follow:

$$\mathbf{v}_L = (v_1, v_2, \cdots, v_n)$$

where:

$$v_i = \begin{cases} 1 & \text{if } \mathbf{p}_i \text{ is a point on } L \\ 0 & \text{otherwise} \end{cases}$$

We form a $J \times n$ matrix $\mathbf{H}_{\mathbf{Q}}^{(1)}$ whose rows are the incidence vectors of the J lines of the finite geometry $\mathbf{Q}$ and whose columns correspond to the n points of $\mathbf{Q}$.
$\mathbf{H}_{\mathbf{Q}}^{(1)}$ has four properties :

1. Each row consists of ρ 1's.
2. Each column has γ 1's.
3. No two rows have more than one 1 in common.
4. No two column have more than one 1 in common

The null space of $\mathbf{H}_{\mathbf{Q}}^{(1)}$ gives an LDPC code of length n. This code is called the $type - I\ geometry - \mathbf{Q}$ LDPC code, denoted by $C_{\mathbf{Q}}^{(1)}$, which has a minimum distance of at least $\gamma + 1$.

- The $n \times J$ matrix: $\mathbf{H}_{\mathbf{Q}}^{(2)}$ is transpose of $\mathbf{H}_{\mathbf{Q}}^{(1)}$, denote: $\mathbf{H}_{\mathbf{Q}}^{(2)} = [\mathbf{H}_{\mathbf{Q}}^{(1)}]^T$
- The properties and rank of $\mathbf{H}_{\mathbf{Q}}^{(1)}$ and $\mathbf{H}_{\mathbf{Q}}^{(2)}$ are the same.
- The null space of $\mathbf{H}_{\mathbf{Q}}^{(2)}$ gives an LDPC code of length J with a minimum distance of at least $\rho + 1$. This LDPC code is called the $type - II\ geometry - \mathbf{Q}$ LDPC code.

Four classes of finite-geometry LDPC codes can be constructed

1. Type-I Eucliden geometry (EG)-LDPC codes.
2. Type-II EG-LDPC codes.
3. Type-I projective geometry (PG)-LDPC codes.
4. Type-II PG-LDPC codes.

EG-LDPC code

- A type-I EG-LDPC code based on EG$(m, 2^s)$, we form the parity-checked matrix $\mathbf{H}_{EG}^{(1)}$, whose rows are the incidence vectors of all the lines in $EG(m, 2^s)$ and whose columns correspond to all the points in EG$(m, 2^s)$. therefore, $\mathbf{H}_{EG}^{(1)}$ consist of

$$J = \frac{2^{(m-1)s}(2^{ms} - 1)}{2^s - 1}$$

consists rows and $n = 2^{ms}$ columns.

- Because each line in EG$(m, 2^s)$ consists of 2^s points, each row of $\mathbf{H}_{EG}^{(1)}$ has weigh $\rho = 2^s$. Since each poi9nt in EG$(m, 2^s)$ is interested by $(2^{ms} - 1)/(2^s - 1)$ lines, each column of $\mathbf{H}_{EG}^{(1)}$ has weight $\gamma = (2^{ms} - 1)/(2^s - 1)$. The density γ of $\mathbf{H}_{EG}^{(1)}$

$$\gamma = \frac{\rho}{n} = \frac{2^s}{2^{ms}} = 2^{-(m-1)s}$$

- For $m \geq 2$ and $s \geq 2$, $r \leq 1/4$ and $\mathbf{H}_{EG}^{(1)}$ is a low-density parity-check matrix. The null space of $\mathbf{H}_{EG}^{(1)}$ hence give an LDPC code of length $n = 2^{ms}$, which is called an m-dimensional type-I (0,s)th order EG-LDPC code denoted by $C_{EG}^{(1)}(m, 0, s)$ The minimum distance of this code is lower bounded as follows:

$$d_{min} \geq \gamma + 1 = \frac{2^{ms} - 1}{2^s - 1} + 1$$

- To construct an m-dimensional type-II EG-LDPC code, we take the transpose of $\mathbf{H}_{EG}^{(1)}$, which gives the parity-checked matrix

$$\mathbf{H}_{EG}^{(2)} = [\mathbf{H}_{EG}^{(1)}]^T$$

Matrix $\mathbf{H}_{EG}^{(2)}$ consist of $J = 2^{ms}$ rows and $n = 2^{(m-1)s}(2^{ms} - 1)/(2^s - 1)$ columns.

- Let α be a primitive element of $GF(2^{ms})$, Then $\alpha^0 = 1, \alpha, alpha^2, \ldots, \alpha^{2^{ms}-2}$ are all the $2^{ms} - 1$ nonorigin points of EG$(m, 2^s)$. Let

$$v = (v_0, v_1, \ldots, v_{2^{ms}-2})$$

be a $(2^{ms} - 1)$-tuple over $GF(2)$ whose components correspond to the $2^{ms} - 1$ nonorigin points of EG$(m, 2^s)$, where v_i corresponds to the point α^j with $0 \leq i < 2^{ms} - 1$

- Let L be a line in EG$(m, 2^s)$, that does not pass through the origin, Based on L, we form $(2^{ms} - 1)$-tuple over GF(2) as follow:

$$V_L = (v_0, v_2, \ldots, v_{2^{ms}-2})$$

whose ith component

$$v_i = \begin{cases} 1 & \text{if } \alpha^i \text{ is a point on } L \\ 0 & \text{otherwise} \end{cases}$$

The vector $\mathbf{v}_L$ is the incidence vector on the line L.

$$J_0 = \frac{(2^{(m-1)s} - 1)(2^{ms} - 1)}{2^s - 1}$$

lines in EG$(m, 2^s$ that do not pass through the origin.

Let $\mathbf{H}_{EG,c}^{(1)}$ be a matrix whose rows are incidence vectors of all the J_0 lines in EG$(m, 2^s)$ and whose columns correspond to the $n = 2^{ms} - 1$ nonorigin points of EG$(m, 2^s)$. The matrix has following properties:

1. Each rows has weight $\rho = 2^s$.
2. Each columns has weight $\gamma = (2^{ms} - 1)/(2^s - 1) - 1$.
3. No two columns have more than one 1's in common; that is $\lambda = 0$ or 1
4. No two rows have more than one 1 in common.

The density of $H_{EG}^{(1)}$ is

$$r = \frac{2^s}{2^{ms} - 1}$$

Again, for $m \geq 2$ and $s \geq 2$, r is relatively small compared with 1. Therefore, $\mathbf{H}_{EG,c}^{(1)}$ is a low-density matrix.

Let α be a primitive element of $GF(2^{ms})$. Let h be a nonnegative integer less than $2^{ms} - 1$. For a nonnegative integer l, let $h^{(l)}$ be the remainder resulting from dividing $2^l h$ by $2^{ms} - 1$. Then, $g_{EG,c}^{(1)}(X)$ has α^h as a root if and only if

$$0 < \max_{0 \leq l < s} W_{2^s}(h^{(l)}) \leq (m-1)(2^s - 1)$$

where $W_{2^s}(h^{(l)})$ is the 2^s-weight of $h^{(l)}$.

Let h_0 be the smallest integer that does not satisfy the condition. It can be shown that

$$\begin{aligned} h_0 &= (2^s - 1) + (2^s - 1)2^s + \cdots + (2^s - 1)2^{(m-3)s} + 2^{(m-2)s} + 2^{(m-1)s} \\ &= 2^{(m-1)s} + 2^{(m-2)s+1} - 1 \end{aligned}$$

Therefore, $g_{EG,c}^{(1)}(X)$ has following sequence of consecutive powers of

α:

$$\alpha, \alpha^2, \ldots, \alpha^{h_0 - 1}$$

as roots. It follows from the BCH bound that the d_{min} of the m-dimensional type-I cyclic $(0, s)$th-order EG-LDPC code $C_{EG}^{(1)}(m, 0, s)$is lower bound as follows

$$d_{EG,c}^{(1)} \geq 2^{(m-1)s} + 2^{(m-2)s+1} - 1$$

A special subclass of type-I cyclic EG-LDPC code is the class of two-dimensional type-I cyclic $(0, s)$th-order EG-LDPC codes. $C_{EG,C}^{(1)}(2, 0, s)$ has the following parameters:

Length	$n = 2^{2s} - 1$
Number of parity bits	$n - k = 3^s - 1$
Dimension	$k = 2^{2s} - 3^s$
Minimum distance	$d_{min} = 2^s + 1$
Density	$r = \frac{2^s}{2^{2s} - 1}$

Two-dimensional type-I cyclic (0,s)th-order EG-LDPC codes

s	n	k	d_{min}	ρ	γ	r
2	15	7	5	4	4	0.267
3	63	37	9	8	8	0.127
4	255	175	17	16	16	0.0627
5	1023	781	33	32	32	0.0313
6	4095	3367	65	64	64	0.01563
7	16383	14197	129	128	128	0.007813

The companion code of the m-dimensional type-I (0,s)th-order EG-LDPC code $C_{EG,C}^{(1)}$ is the null space of the parity-check matrix

$$\mathbf{H}_{EG,qc}^{(2)} = [\mathbf{H}_{EG,c}^{(1)}]^T$$

This LDPC code has length

$$n = J_0 = \frac{(2^{(m-1)s} - 1)(2^{ms} - 1)}{2^s - 1}$$

and a minimum distance d_{min} of at least $2^s + 1$. It is not cyclic but can be put in quasicyclic form. We call this code an m-dimensional type-II quasi-cyclic (0,s)th-order EG-LDPC code, denoted by $C_{EG,qc}^{(2)}(m, 0, s)$.

- To put $C_{EG,qc}^{(2)}(m, 0, s)$ artition the J_0 incidence vectors of the lines in EG(m,2^s) not pass through origin into $K=\frac{2^{(m-1)s}-1}{2^s-1}$ cyclic classes.
- Each of these K cyclic classes contains 2^{ms}-1 incidence vectors, which are obtained by cyclically shifting any incidence vectors in the class 2^{ms}-1 times
- The (2^{ms}-1)×K matrix $\mathbf{H}_0$ whose K columns are the K representative incidence vectors of K cyclic class .The $\mathbf{H}_{\mathbf{EG,qc}}^{(2)}$=[$\mathbf{H}_0$,$\mathbf{H}_1$.....$\mathbf{H}_{2^{ms}-2}$], $\mathbf{H}_i$ is the a (2^{ms}-1)×K matrix whose columns are the ith downward cyclic shift of the column of $\mathbf{H}_0$.
- The null space of $\mathbf{H}_{\mathbf{EG,qc}}^{(2)}$ gives the type-II EG-LDPC code $\mathrm{C}_{EG,qc}^{(2)}$(m,0,s) in quasi-cyclic form.

PG-LDPC code

Let α be a primitive element of $GF(2^{(m+1)s})$, which is considered as an extension field of $GF(2^s)$. Let

$$n = \frac{2^{(m+1)s} - 1}{2^s - 1}$$

Then, the n elements,

$$(\alpha^0)(\alpha^1)(\alpha^2), \ldots, (\alpha^{n-1})$$

form an m-dimensional projective geometry over $GF(2^s)$, $PG(m, 2^s)$. The element $(\alpha^0)(\alpha^1)(\alpha^2), \ldots, (\alpha^{n-1})$ are the point of $PG(m, 2^s)$. A line in $PG(m, 2^s)$ consists of $2^s + 1$ points. There are

$$J = \frac{(1 + 2^s + \ldots + 2^{ms})(1 + 2^s + \ldots + 2^{(m-1)s})}{1 + 2^s}$$

lines in PG$(m, 2^s)$.

- Every point (α^i) in PG$(m, 2^s)$ is intersected by
$$\gamma = \frac{2^{ms} - 1}{2^s - 1}$$
lines. Two lines in PG$(m, 2^s)$ are either disjoint or intersect at one and only one point.
- We form a matrix $\mathbf{H}^{(1)}_{\mathbf{PG}}$ whose rows are the incidence vectors of lines in PG$(m, 2^s)$ and whose columns correspond to the points of PG$(m, 2^s)$. Then, $\mathbf{H}^{(1)}_{PG}$ has
$$J = \frac{(2^{(m-1)s} + \cdots + 2^s + 1)(2^{ms} + \cdots + 2^s + 1)}{2^s + 1}$$
and $n = 2^{(m+1)s} - 1/2^s - 1$

The matrix $\mathbf{H}^{(1)}_{PG}$ has the following properties:

1. Each rows has weight $\rho = 2^s + 1$.
2. Each columns has weight $\gamma = \frac{2^{ms}-1}{2^s-1}$.
3. No two columns have more than one 1 in common.
4. No two rows have more than one 1 in common. The density of $\mathbf{H}^{(1)}_{PG}$ is
$$r = \frac{\rho}{n} = \frac{(2^s - 1)(2^s + 1)}{2^{(m+1)s} - 1}$$

The code $C^{(1)}_{PG}$(m,0,s) is the null space of $\mathbf{H}^{(1)}_{\mathbf{PG}}$ has $d_{min} \geq \gamma+1$ and it is called m-dimensional type-I (0,s)th order PG-LDPC code.

Specified generator polynomial

Let h be a nonegative integer less than $2^{(m+1)s}$-1, and
$2^l h$=q($2^{(m+1)s}$-1)+$h^{(l)}$,
α^h as roots if and only if 0<max $W_{2^s}(h^{(l)}) \leqq j(2^s$-1).
Let $\xi = \alpha^{2^s-1}$.The order of ξ is then n=$\frac{2^{(m+1)s}-1}{2^s-1}$.
$g^{(1)}_{PG}$(X) has the following consecutive powers of ξ: $\xi, \xi^2 \ldots\ldots \xi^{\frac{2^{ms}-1}{2^s-1}}$

Special subclass is 2-dimensional type-I cyclic (0,s)th-order PG-LDPC code,whose null space is $C_{PG}^{(1)}$(2,0,s) of length n=2^{2s}-1
$C_{PG}^{(1)}$(2,0,s) has parameter:

Length	$n = 2^{2s} + 2^s + 1$
Number of parity bits	$n - k = 3^s + 1$
Dimension	$k = 2^{2s} + 2^s - 3^s$
Minimum distance	$d = 2^s + 2$
Density	$r = \frac{2^s+1}{2^{2s}+2^s+1}$

Two-dimensional type-I PG-LDPC codes

s	n	k	d_{min}	ρ	γ	r
2	21	11	6	5	5	0.2381
3	73	45	10	9	9	0.1233
4	273	191	18	17	17	0.0623
5	1057	813	34	33	33	0.0312
6	4161	3431	66	65	65	0.0156
7	16513	14326	130	129	129	0.0078

The type-II PG-LDPC code matrix:

$$\mathbf{H}_{PG}^{(2)} = [\mathbf{H}_{EG}^{(1)}]^T$$

$\mathbf{H}_{PG}^{(2)}$ has $J = \frac{2^{(m+1)s}-1}{2^s-1}$ rows and

$$n = \frac{(2^{(m-1)s} + \cdots + 2^s + 1)(2^{ms} + \cdots + 2^s + 1)}{2^s + 1}$$

The row and column weights of $\mathbf{H}_{PG}^{(2)}$ are $\rho = \frac{2^{ms}-1}{2^s-1}$ and $\gamma = 2^s + 1$, respectively. Therefore, $C_{PG}^{(2)}(m, 0, s)$ has length n and minimum distance

$$d_{min} \geq 2^s + 2$$

Random LDPC code

To construct a parity check matrix, to choose an appropriate column weight γ and a appropriate number J rows, J must be close to equal $n-k$. If the rows of $\mathbf{H}$ weight is ρ, the number of 1 is

$$\gamma \times n = \rho \times (n-k)$$

If n is divisible by $(n-k)$,

$$\rho = \frac{\gamma n}{n-k}$$

this case can be constructed as a regular LDPC code.
If n is not divisible by $(n-k)$,

$$\gamma \times n = \rho(n-k) + b$$

b is constant.

$$\gamma \times n = \rho(n-k-b) + b(\rho+1)$$

Suggest the parity check matrix has two row weights, top b rows weight=ρ+1, bottom $(n-k-b)$ rows weight=ρ.

A $n-k$ tuple column $\mathbf{h}_i$ has weight γ,and add to partial of the parity check matrix:

$$\mathbf{H}_{i-1} = \{\mathbf{h}_1, \mathbf{h}_i, \ldots, \mathbf{h}_{i-1}\}$$

There are 3 steps:

1. Chosen $\mathbf{h}_i$ at random from the remaining binary $(n-k)$-tuples that are not being used in $\mathbf{H}_{i-1}$ and that were not reject earlier
2. Check whether $\mathbf{h}_i$ has more than one 1-component with any column, if it is not go to 3, reject $\mathbf{h}_i$, go back 1 step.
3. Add $\mathbf{h}_i$ to $\mathbf{H}_{i-1}$ form a temporary partial parity check matrix, If all the top b rows weight $\leq \rho+1$ and bottom $(n-k-b)$ rows weight $\leq \rho$,then permanently add $\mathbf{h}_i$ to $\mathbf{H}_{i-1}$ to form $\mathbf{H}_i$ and go to step 1 continue construction process, else reject $\mathbf{h}_i$ and choose a new column.

Decoding of LDPC code

An LDPC code can be decoded in various ways namely:

- Bit-flipping decoding algorithm
- The sum-product algorithm

- A codeword $\mathbf{v} = (v_0, v_1, \cdots, v_{n-1})$ is mapped into a bipolar sequence $\mathbf{x} = (x_0, x_1, \cdots, x_{n-1})$ before its transmission, where $x_l = (2v_l - 1) = +1$ for $v_l = 1$, and $x_l = -1$ for $v_l = 0$ with $0 \leq l \leq n-1$
- Let $\mathbf{y} = (y_0, y_1, \cdots, y_{n-1})$ be the soft-decision received sequence at the output of the receiver matched filter. For $0 \leq l \leq n-1$, $y_l = \pm 1 + n_l$, where n_l is a Gaussian random variable with zero mean and variance $N_0/2$.
- $\mathbf{z} = (z_0, z_1, \cdots, z_{n-1})$ be the binary hard-decision sequence obtained from $textbfy$ as follow:

$$z_l = \begin{cases} 1 & \text{for } y_l > 0 \\ 0 & \text{for } y_l \leq 0 \end{cases}$$

Let H be the parity-check matrix of and LDPC code C with j rows and n column. Let $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_J$, denote the rows of $\mathbf{H}$,where

$$\mathbf{h}_j = (h_{j,0}, h_{j,1}, \cdots, h_{j,n-1})$$

for $1 \leq j \leq J$ Then,

$$\mathbf{s} = (s_1, s_2, \cdots, s_J) = \mathbf{z} \cdot \mathbf{H}^T$$

gives the syndrome of the received sequence $\mathbf{Z}$, where the jth syndrome component s_j is given by the checke-sum

$$s_j = \mathbf{z}.\mathbf{h}_j = \sum_{l=0}^{n-1} z_l h_{j,l}$$

The received vector $\mathbf{z}$ is a codeword if and only if $\mathbf{s{=}0}$. If $\mathbf{s} \neq \mathbf{0}$, error in $\mathbf{z}$ are detected

A non zero syndrome component s_j indicates a $parity failure$. The number of parity failure is equal to the number of nonzero syndrome components in $\mathbf{s}$ Let

$$\begin{aligned} \mathbf{e} &= (e_0, e_1, \cdots, e_{n-1}) \\ &= (v_0, v_1, \cdots, v_{n-1}) + (z_0, z_1, \cdots, z_{n-1}) \end{aligned}$$

Then, $\mathbf{e}$ is the error pattern in $\mathbf{z}$. This error pattern $\mathbf{e}$ and the syndrome $\mathbf{s}$ satisfy the condition

$$\mathbf{s} = (s_1, s_2, \cdots, s_J) = \mathbf{e} \cdot \mathbf{H}^T$$

where

$$s_j = \mathbf{e} \cdot \mathbf{h}_j = \sum_{l=0}^{n-1} e_l h_{j,l}$$

Bit-flipping decoding algorithm

- Bit-flipping decoding was devised by Gallager in 1960.
- A very simple BF decoding algorithm is given here:
  1. Compute the parity-check sum. If all parity-check sums are zero, stop the decoding.
  2. Find the number of failed parity-check equations for each bit, denoted by f_i, $i = 0, 1, \ldots, n-1$.
  3. Identify the set S of bits for which f_i is the largest
  4. Flip the bits in the set S
  5. Repeat step 1 to 4 until all parity-check sum are zero, or a preset maximum number of iterations is reached

- If preset maximum number of iterations is reached and not all parity-check sum are zero, we may simply declare a decoding failure or decode the unmodified received sequence $\mathbf{z}$ with MLG decoding to obtain a decoded sequence, which may not be a codeword in C
- The parameter δ called threshold, is a design parameter that should be chosen to optimize the error performance while minimizing the number of computations of parity-check sums.
- The value of δ depend on the code parameters ρ, γ, d_{min} and SNR
- If decoding fails for a given value of δ,then the value of δ should be reduced to allow further decoding iterations.

The sum-product algorithm

- The sum-product algorithm decoding is a soft decision base on log-likelihood ratio,i t can improve the reliability measure.

We consider an LDPC code C of length n specified by a parity-check matrix $\mathbf{H}$ with J rows, $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_J$, where

$$\mathbf{h}_j = (h_{j,0}, h_{j,1} \cdots, h_{j,n-1})$$

For $1 \leq j \leq J$, we define the following index set for $\mathbf{h}_j$

$$B(\mathbf{h}_j) = \{l : h_{j,l} = 1, 0 \leq l < n\}$$

which is called the *sopport* of $\mathbf{h}_j$

The implementation of the sum-product algorithm bases on the computation of the marginal a posteriori probabilities

$$P(v_l|\mathbf{y})$$

for $0 \leq l < n$, where $\mathbf{y}$ is the soft-decision received sequence.Then, the LLR for each code bit is given by

$$L(v_l) = \log \frac{p(v_l = 1|y)}{p(v_l = 0|y)}$$

Let $p_l^0 = P(v_l = 0)$ and $p_l^1 = P(v_l = 1)$ be the prior probabilities of $v_l = 0$ and $v_l = 1$.

For $0 \le l < n$, $1 \le j \le n$, and each $\mathbf{h}_j \in \mathbf{A}_l$, let $q_{j,l}^{x,(i)}$ be the conditional probability that the transmitted code bit v_l has value x, given the check-sums computed based on the check vectors in $A_l/\mathbf{h}_j$ at the ith decoding iteration.
For $0 \le l < n$, $1 \le j \le n$, and $\mathbf{h}_j \in \mathbf{A}_l$, let $\sigma_{j,l}^{x,(i)}$ be the conditional probability that the check-sum s_j is satisfied (i.e., $s_j = 0$),given $v_l = x$ (0 or 1) and the other code bits in $\mathbf{B}(\mathbf{h}_j)$ have a separable distribution $\{q_{j,t}^{v_t,(i)} : t \in B(\mathbf{h}_j) \setminus l\}$; that is:

$$\sigma_{j,l}^{x,(i)} = \sum_{\{v_t : t \in B(\mathbf{h}_j) \setminus l\}} P(s_j = 0 | v_l = x, \{v_t : t \in B(\mathbf{h}_j) \setminus l\}) \cdot \prod_{t \in B(\mathbf{h}_j) \setminus l} q_{j,t}^{v_t,(i)}$$

To computed values of $\sigma_{j,l}^{x,(i)}$, and then used to update the value $q_{j,l}^{x,(i+1)}$ as follows:

$$q_{j,l}^{x,(i+1)} = \alpha_{j,l}^{(i+1)} \cdot_l^x \prod_{\mathbf{h}_t \in A_l \setminus \mathbf{h}_j} \sigma_{t,l}^{x,(i)}$$

where $\alpha_{j,l}^{i+1}$ is chosen such that

$$q_{j,l}^{0,(i+1)} + q_{j,l}^{1,(i+1)} = 1$$

At the ith step, the pseudo-prior probabilities are given by

$$P^i(v_l = x | y) = \alpha_l^{(i)} p_l^x \prod_{\mathbf{h}_j \in A_l} \sigma_{j,l}^{x,(i-1)}$$

where α_l^i is chosen such that $P^{(i)}(v_l = 0|\mathbf{y}) + P^{(i)}(v_l = 1|y) = 1$

Base on these probabilities, we can form the following vector as the decoded candidate:

$$\mathbf{z}^{(i)} = (z_0^{(i)}, z_1^{(i)}, \cdots, z_{n-1}^{(i)})$$

with

$$z_l^{(i)} = \begin{cases} 1 & \text{for } P^{(i)}(v_l = 1|\mathbf{y}) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Then, compute $\mathbf{z}^{(i)} \cdot \mathbf{H}^T$. If $\mathbf{z}^{(i)} \cdot \mathbf{H}^T = 0$, stop the decoding iteration process, and output $\mathbf{z}^{(i)}$ as the decoded codeword.

The sum-product algorithm decoding in terms of probability consist of the following steps :
Initialization: Set $i = 0$ and maximum number of iterations to I_{max}. For every pair (j, l) such that $h_{j,l} = 1$ with $1 \le j \le J$ and $0 \le l \le n$, set $q_{j,l}^{0,(0)} = p_l^0$ and $q_{j,l}^{1,(0)} = p_l^1$.

1. For $0 \le l \le n$, $1 \le j \le J$, and each $\mathbf{h}_j \in A_l$, compute the probabilities of $\sigma_{j,l}^{0,(i)}$ and $\sigma_{j,l}^{1,(i)}$. Go to step 2
2. For $0 \le l \le n$, $1 \le j \le J$, and each $\mathbf{h}_j \in A_l$, compute the values of $q_{j,l}^{0,(i+1)}$ and $q_{j,l}^{1,(i+1)}$ and the values of $P^{(i+1)}(v_l = 0|\mathbf{y})$ and $P^{(i+1)}(v_l = 1|\mathbf{y})$. Form $\mathbf{z}^{(i+1)}$ and test $\mathbf{Z}^{(i+1)} \cdot \mathbf{H}^T$. If $\mathbf{Z}^{(i+1)} \cdot \mathbf{H}^T = 0$ or the maximum iteration number I_{max} is reached, go to step 3. Otherwise, set $i \doteq i + 1$ and go to step 1.
3. Output $z^{(i+1)}$ as the decoded codeword and stop the decoding process