



Chapter 6. Convolutional Codes and Viterbi Decoding

§1. Basic Definitions

Generally speaking, a binary convolutional code is a linear K/R system defined on the binary field $\{0, 1\}$ with modulo 2 operation.

$\underline{u}(D) = (u_1(D), \dots, u_k(D))$, $1 \times k$ information vector

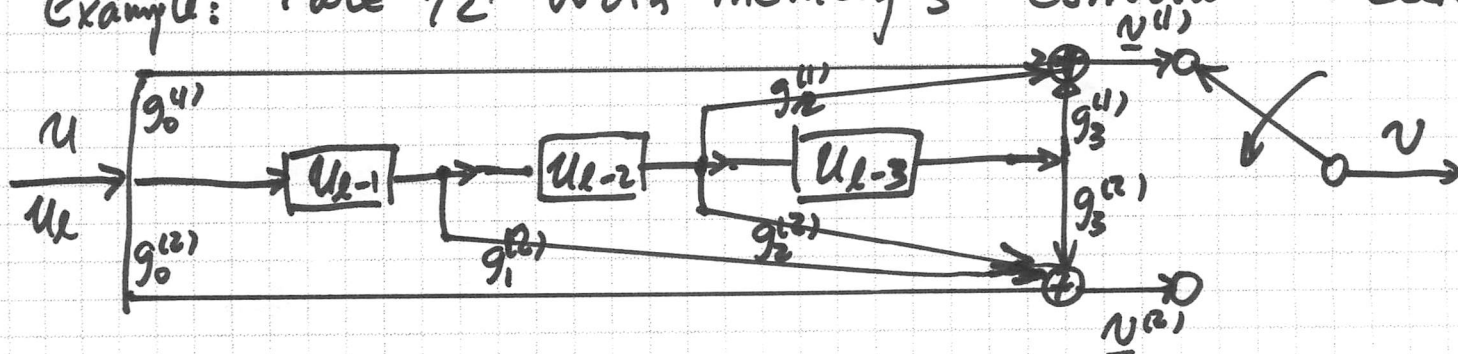
$\underline{v}(D) = (v_1(D), \dots, v_n(D))$, $1 \times n$ encoded vector

Code generator matrix

$$\underline{G}(D) = \begin{pmatrix} g_{11}(D) & \dots & g_{1n}(D) \\ g_{21}(D) & \dots & g_{2n}(D) \\ \vdots & \ddots & \vdots \\ g_{k1}(D) & \dots & g_{kn}(D) \end{pmatrix}$$

Encoding: $\underline{v}(D) = \underline{u}(D) \cdot \underline{G}(D)$

Example: rate $1/2$ with memory 3 Convolutional Code



In this case, the information sequence $\underline{u} = (u_0, u_1, \dots)$ enters the encoder one bit at a time. The two encoded output sequences :



$$\underline{v}^{(1)} = (v_0^{(1)}, v_1^{(1)}, \dots)$$

$$\underline{v}^{(2)} = (v_0^{(2)}, v_1^{(2)}, \dots)$$

are the convolution of the input sequence $\underline{u}$ with the two encoders "impulse responses"

$$\underline{g}^{(1)} = (1, 0, 1, 1)$$

$$\underline{g}^{(2)} = (1, 1, 1, 1)$$

$\Rightarrow$ called the generator sequences of the code.

The encoding equation in this case can be written as

$$\underline{v}^{(1)} = \underline{u} * \underline{g}^{(1)}, \quad \underline{v}^{(2)} = \underline{u} * \underline{g}^{(2)}$$

$$v_l^{(i)} = \sum_{j=0}^m u_{l-j} g_j^{(i)} = u_l g_0^{(i)} + u_{l-1} g_1^{(i)} + \dots + u_{l-m} g_m^{(i)}, \quad i=1, 2,$$

where $u_{l-j} = 0$ for all $l < j$.

$$\Rightarrow v_l^{(1)} = u_l + \quad + u_{l-2} + u_{l-3} \quad \text{mod } 2$$

$$v_l^{(2)} = u_l + u_{l-1} + u_{l-2} + u_{l-3} \quad \text{mod } 2$$

Finally, the output sequences are multiplexed into a single sequence called the codeword:

$$\underline{v} = (v_0^{(1)}, v_0^{(2)}, v_1^{(1)}, v_1^{(2)}, v_2^{(1)}, v_2^{(2)}, \dots)$$



Example: Let the information sequence $\underline{u} = (1, 0, 1, 1, 1)$.
Then, the output sequences are

$$\underline{v}^{(1)} = (1, 0, 1, 1, 1) * (1, 0, 1, 1) = (1, 0, 0, 0, 0, 0, 0, 1)$$

$$\underline{v}^{(2)} = (1, 0, 1, 1, 1) * (1, 1, 1, 1) = (1, 1, 0, 1, 1, 1, 0, 1)$$

$$\underline{v} = (1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1).$$

If the generator sequences $\underline{g}^{(1)}$ and $\underline{g}^{(2)}$ are interlaced
and then arranged in the matrix

$$\underline{G} = \begin{bmatrix} g_0^{(1)} & g_0^{(2)} & g_1^{(1)} & g_1^{(2)} & g_2^{(1)} & g_2^{(2)} & \dots & g_m^{(1)} & g_m^{(2)} \\ & g_0^{(1)} & g_0^{(2)} & g_1^{(1)} & g_1^{(2)} & \dots & g_{m-1}^{(1)} & g_{m-1}^{(2)} & g_m^{(1)} & g_m^{(2)} \\ & & & & & & & & & \\ & & & & & & & & & \end{bmatrix}$$

then encoding equations can be rewritten as

$$\underline{v} = \underline{u} \cdot \underline{G}$$

where $\underline{G}$ is called the generator matrix of the code.

If $\underline{u}$ has finite length L , the generator matrix $\underline{G}$
has L rows and $2(m+L)$ columns, and $\underline{v}$ has length
 $2(m+L)$.

Back to the example above:



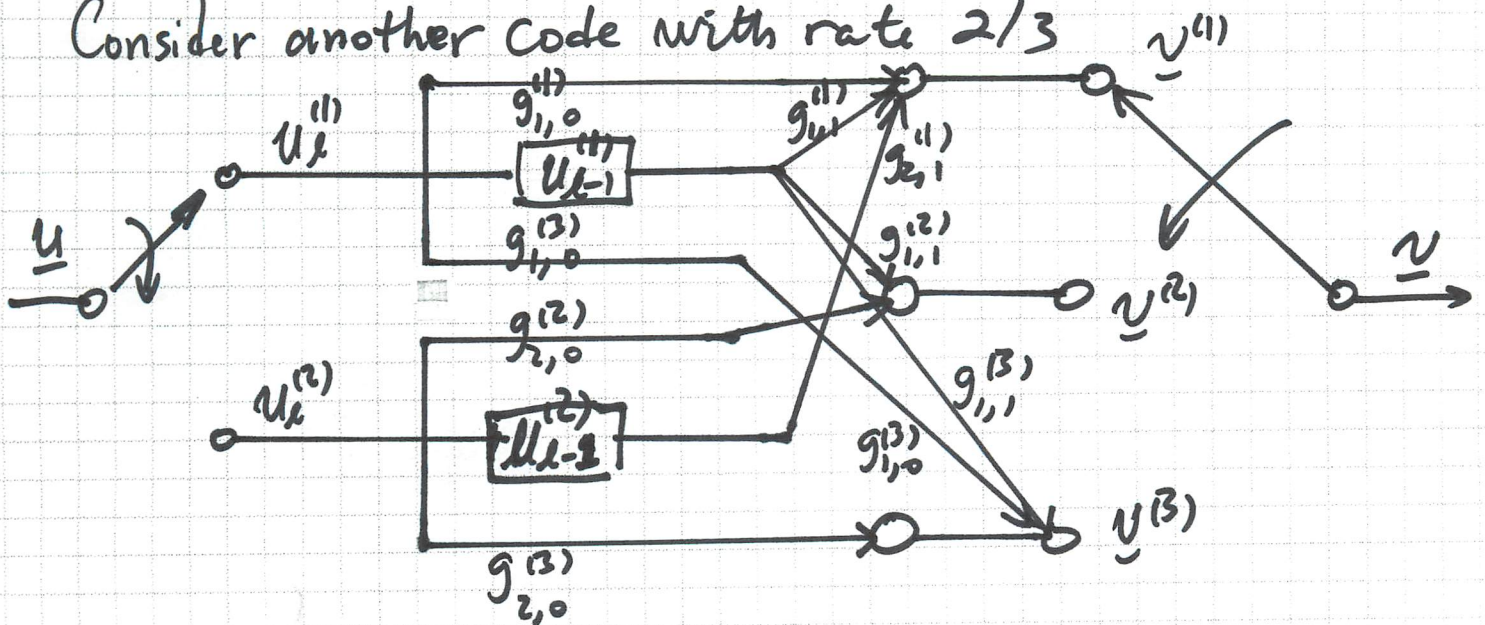
7-4

(1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1)

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

$$L=5, \quad m=3, \Rightarrow 2(3+5)=16$$

Consider another code with rate $2/3$



$$k=2, \quad n=3, \quad m=1.$$

two inputs generate three outputs

The information sequence enters the encoder $k=2$ bits at a time, which is written as

$$\underline{u} = (u_0^{(1)} u_0^{(2)} u_1^{(1)} u_1^{(2)} u_2^{(1)} u_2^{(2)}, \dots)$$

or two input sequences

$$\underline{u}^{(1)} = (u_0^{(1)}, u_1^{(1)}, u_2^{(1)}, \dots)$$

$$\underline{u}^{(2)} = (u_0^{(2)}, u_1^{(2)}, u_2^{(2)}, \dots)$$



There are three generator sequences corresponding to each input sequence.

Let $\underline{g}_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \dots, g_{i,m}^{(j)})$ represent the generator sequence corresponding to input i and output j , the generator sequences for the above encoder are

$$\underline{g}_1^{(1)} = (1, 1), \quad \underline{g}_1^{(2)} = (0, 1), \quad \underline{g}_1^{(3)} = (1, 1)$$

$$\underline{g}_2^{(1)} = (0, 1), \quad \underline{g}_2^{(2)} = (1, 0), \quad \underline{g}_2^{(3)} = (1, 0)$$

The encoding equations can be written as

$$v^{(1)} = u^{(1)} * \underline{g}_1^{(1)} + u^{(2)} * \underline{g}_2^{(1)} = u_1^{(1)} + u_{2-1}^{(1)} + u_{2-1}^{(2)}$$

$$v^{(2)} = u^{(1)} * \underline{g}_1^{(2)} + u^{(2)} * \underline{g}_2^{(2)} = u_1^{(2)} + u_{2-1}^{(1)}$$

$$v^{(3)} = u^{(1)} * \underline{g}_1^{(3)} + u^{(2)} * \underline{g}_2^{(3)} = u_1^{(1)} + u_1^{(2)} + u_{2-1}^{(1)}$$

The Codeword is

$$\underline{v} = (v_0^{(1)} v_0^{(2)} v_0^{(3)} \quad v_1^{(1)} v_1^{(2)} v_1^{(3)} \quad \dots)$$

The generator matrix of a $(3, 2, m)$ code is

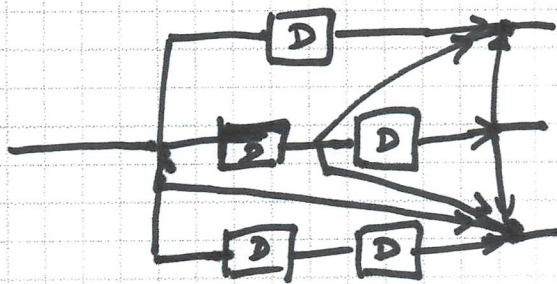
$$\underline{G} = \begin{bmatrix} \underbrace{\{g_{1,0}^{(1)} g_{1,0}^{(2)} g_{1,0}^{(3)} \}}_n & \underbrace{\{g_{1,1}^{(1)} g_{1,1}^{(2)} g_{1,1}^{(3)} \}}_{\vdots} & \dots & \underbrace{\{g_{1,m}^{(1)} g_{1,m}^{(2)} g_{1,m}^{(3)} \}}_{\vdots} \\ \underbrace{\{g_{2,0}^{(1)} g_{2,0}^{(2)} g_{2,0}^{(3)} \}}_n & \underbrace{\{g_{2,1}^{(1)} g_{2,1}^{(2)} g_{2,1}^{(3)} \}}_{\vdots} & \dots & \underbrace{\{g_{2,m}^{(1)} g_{2,m}^{(2)} g_{2,m}^{(3)} \}}_{\vdots} \\ \vdots & \vdots & \ddots & \vdots \end{bmatrix}$$

m blocks



$\underline{u} = \underline{u} \cdot \underline{G}$. If $\underline{u}$ has length $2L$
 then $\underline{G}$ has size $2L \times [3(L+m)]$
 for a $(2, 3, m)$ code.
 Codeword length $2L(3(L+m))$
 $= 6L(L+m)$

For a rate k/n code, there are k shift registers,
 not all of which must have the same length. For example:



Let K_i be the length of the i th shift register, then the
encoder memory order m is defined as

$$m \triangleq \max_{1 \leq i \leq k} K_i$$

that is the maximum length of all k shift registers.
 The constraint length is defined as

$$\nu \triangleq \sum_{i=1}^k K_i, \text{ sum of all } k \text{ shift register lengths.}$$

Total # of delay elements D's.

When $k=1$, $\nu = K_1 = m$.

Three important parameters for a convolutional code:

(k, n, ν)



Consider a general k/n code

$$\underline{G}(D) = \begin{bmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \dots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \dots & g_2^{(n)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \dots & g_k^{(n)}(D) \end{bmatrix} \xrightarrow{\deg} \begin{matrix} \max K_1 \\ + \\ \max K_2 \\ + \\ \max K_k \end{matrix}$$

||
Constraint length v

Encoding $\underline{V}(D) = \underline{U}(D) \underline{G}(D)$

The i th shift register length

$$K_i = \max_{1 \leq j \leq n} [\deg(g_i^{(j)}(D))]$$

Final coded sequence

$$V(D) = V^{(1)}(D^n) + D V^{(2)}(D^n) + \dots + D^{n-1} V^{(n)}(D^n)$$

$$\text{Let } g_i^{(j)}(D) = \sum_{l=0}^m g_{i,j,l} D^l$$

$$\underline{G} = \begin{bmatrix} \underline{G}_0 & \underline{G}_1 & \underline{G}_2 & \dots & \underline{G}_m \\ \underline{G}_0 & \underline{G}_1 & & & \underline{G}_{m-1} & \underline{G}_m \\ & & \ddots & & & \\ & & & \ddots & & \end{bmatrix} \quad (*)$$

where

$$\underline{G}_l = \begin{bmatrix} g_{1,l}^{(1)} & g_{1,l}^{(2)} & \dots & g_{1,l}^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,l}^{(1)} & g_{k,l}^{(2)} & \dots & g_{k,l}^{(n)} \end{bmatrix}$$

$$\text{Then } \underline{V} = \underline{U} \underline{G},$$



Where $u = (u_0, u_1, \dots) = (u_0^{(1)} u_0^{(2)} \dots u_0^{(K)}, u_1^{(1)} u_1^{(2)} \dots u_1^{(K)}, \dots)$
 $v = (v_0, v_1, \dots) = (v_0^{(1)} v_0^{(2)} \dots v_0^{(m)}, v_1^{(1)} v_1^{(2)} \dots v_1^{(m)}, \dots)$

Let kL be the input sequence length, then the generator matrix G has size

$$kL \times n(L+m).$$

In this case, ~~the~~^a convolutional code can be treated as a block code with rate $kL/n(L+m)$

When $L \gg m$, the rate

$$\frac{kL}{n(L+m)} \rightarrow \frac{k}{n}.$$

Note that in this case the generator matrix G is not arbitrary but structured as shown in (*), a kind of blocked Toeplitz matrix.

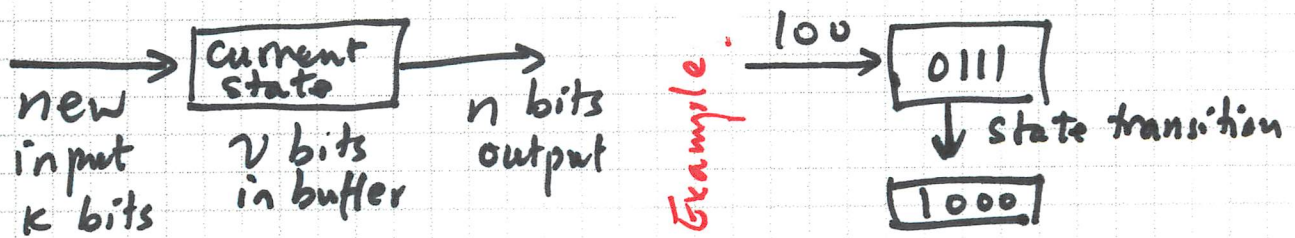


§2. Structure Properties of Convolutional Codes: (CC) State Diagram

A (k, n, v) binary CC needs k input bits and v memory bits (buffer bits) to output n coded bits.

There are total 2^v states.

Each new block of k input bits causes a transition from a current state to a new state



There are 2^k branches leaving each state, one corresponding to each different input block.

Each branch is labeled with k input bits causing the transition $(u_x^{(1)}, u_x^{(2)}, \dots, u_x^{(k)})$ and the n corresponding output bits $(N_x^{(1)}, N_x^{(2)}, \dots, N_x^{(n)})$.

The states are labeled $S_0, S_1, \dots, S_{2^v-1}$, where by convention S_i represents the state whose binary v -tuple representation $b_0 b_1 \dots b_{v-1}$ is equivalent to the integer $i = b_0 2^0 + b_1 2^1 + \dots + b_{v-1} 2^{v-1}$.

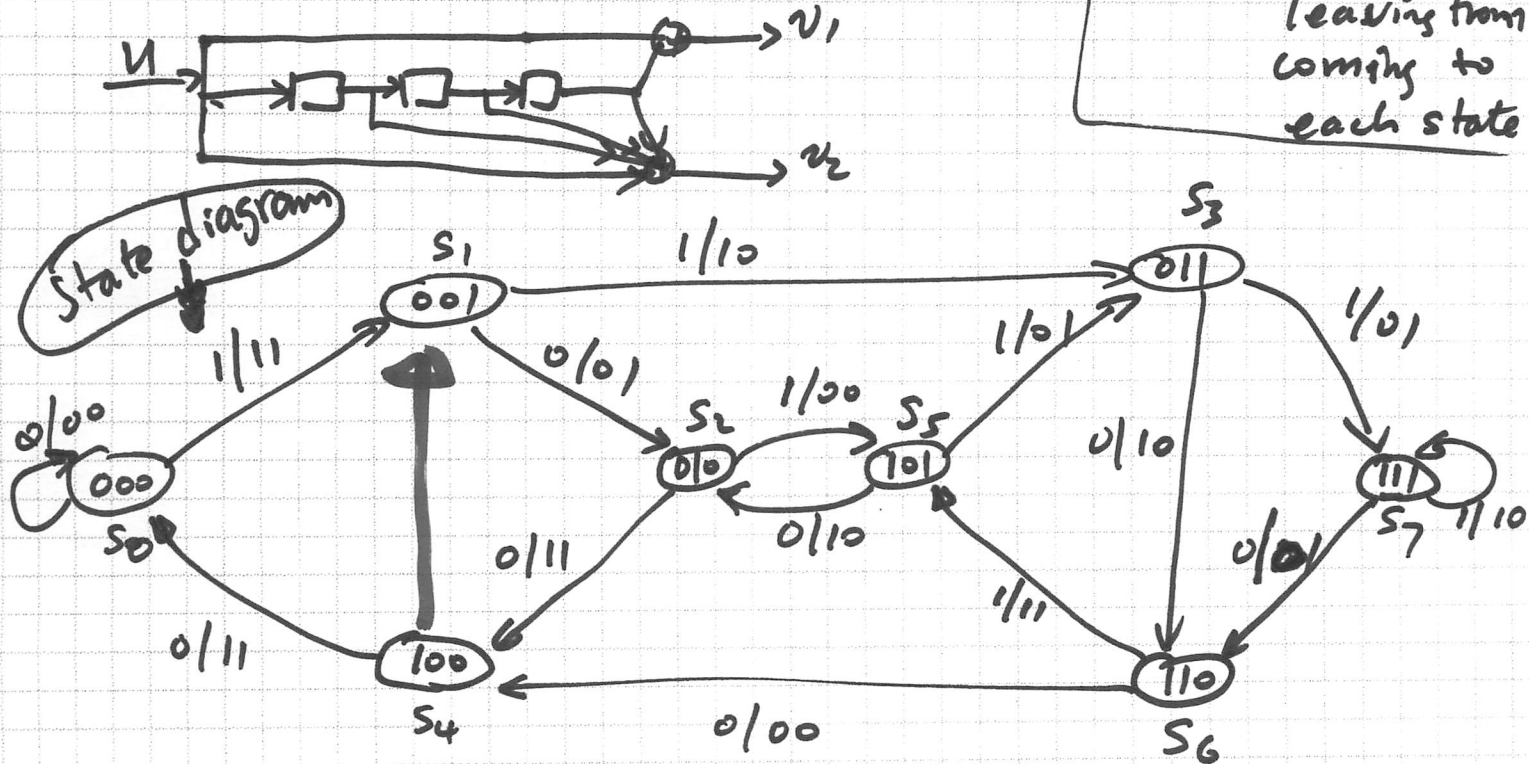


7-10

We now consider the code (1, 2, 3)

$$v = m = 3, \quad k = 1, \quad n = 2$$

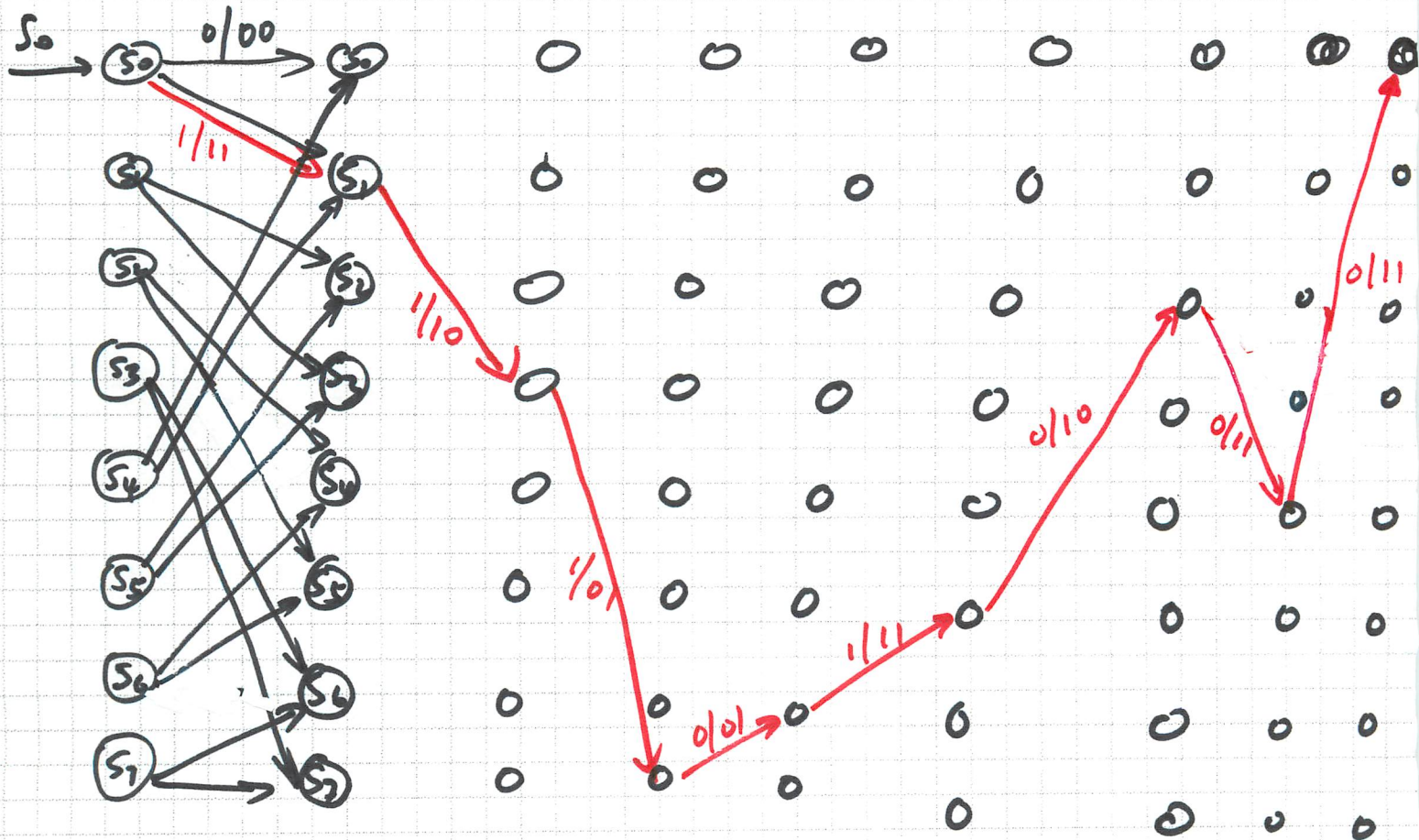
$2^k = 2$ branches
leaving from
coming to
each state



Assuming that the encoder is initially in state S_0 (the all-zero state), the codeword corresponding to any given information sequence can be obtained by following the path through the state diagram determined by the information sequence and noting the corresponding outputs on the branch labels. Following the last nonzero information block, the encoder is returned to state 0 (S_0).



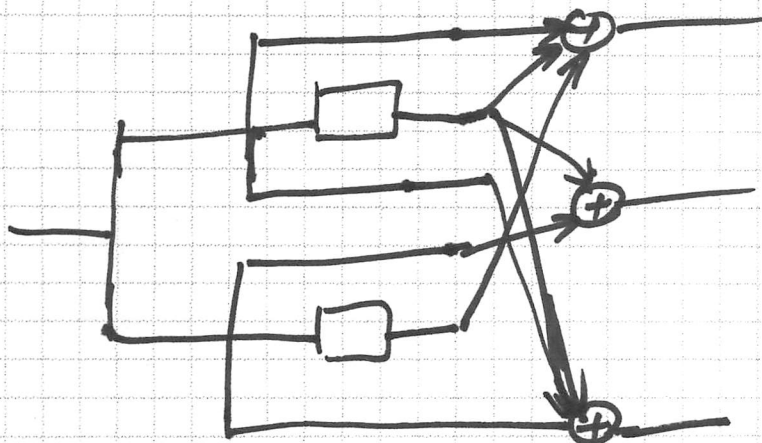
For example, if $\underline{u} = (11101)$, the codeword
 $\underline{v} = (11, 10, 01, 01, 11, 10, 11, 11)$



trellis diagram



Another example

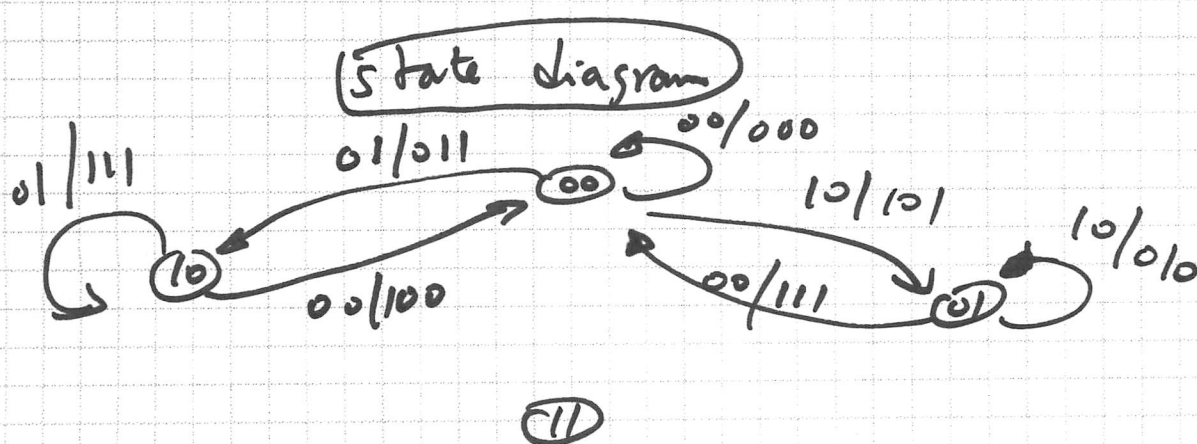


$$v=2$$

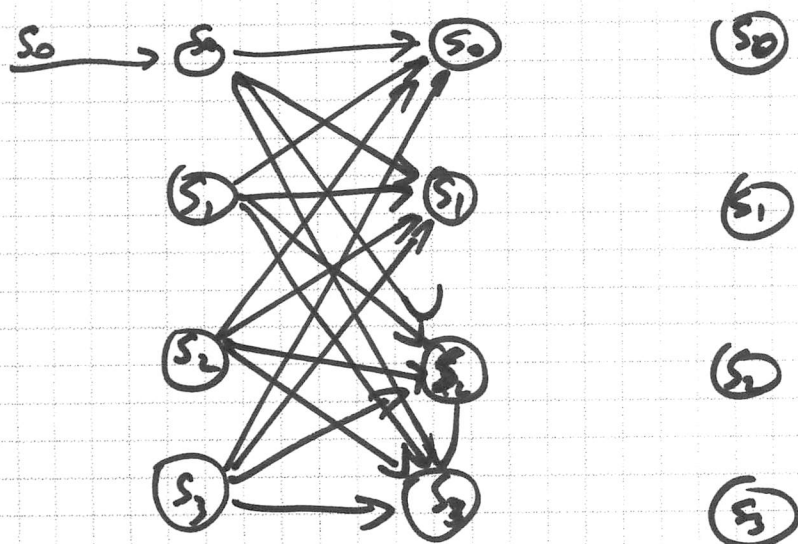
$$k=2$$

$$n=3$$

$2^k = 4$ branches
leaving
coming
each
state



trellis diagram



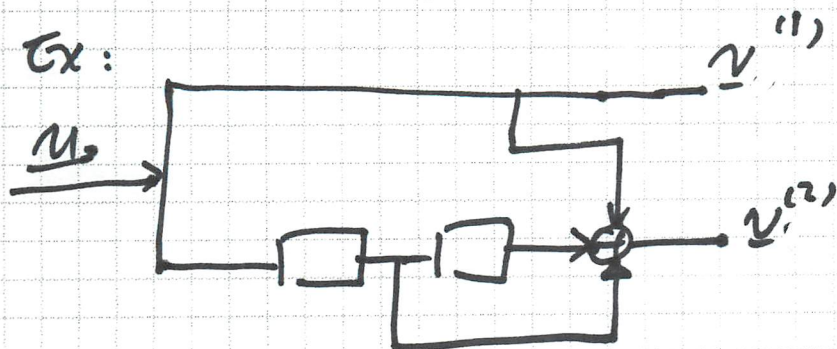


Systematic Codes:

If

$$G(D) = \begin{bmatrix} 1 & 0 & \dots & 0 & g_1^{(k+1)}(D) & \dots & g_1^{(n)}(D) \\ 0 & 1 & \dots & 0 & g_2^{(k+1)}(D) & \dots & g_2^{(n)}(D) \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & g_k^{(k+1)}(D) & \dots & g_k^{(n)}(D) \end{bmatrix}$$

$$\underline{V} = (\underbrace{u_0^{(1)} \dots u_0^{(k)}}_{\text{direct information bits}} \underbrace{v_0^{(k+1)} \dots v_0^{(n)}}_{\text{direct information bits}} \underbrace{u_1^{(1)} \dots u_k^{(k)}}_{\text{direct information bits}} \underbrace{v_1^{(k+1)} \dots v_1^{(n)}}_{\text{direct information bits}} \dots)$$



$$\underline{V}(D) = \underline{U}(D) G(D)$$

When $\underline{U}(D)$ can be recovered from $\underline{V}(D)$? or

When $G(D)$ is invertible?

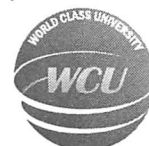
When $k=1$, $G(D) = (g_1^{(1)}(D) \dots g_1^{(n)}(D))$

is (FIR) invertible iff

$$\gcd\{g_1^{(1)}(D), g_1^{(2)}(D), \dots, g_1^{(n)}(D)\} = D^l$$

for some l .

In general, $\gcd\{\Delta_k(D), k=1, 2, \dots, \binom{n}{k}\} = D^l$
 where $\Delta_k(D)$ is the determinant of a $k \times k$ submatrix of $G(D)$.



* FIR invertible: $\exists H(D)$ such that

$$H(D)G(D) = I_{k \times k}$$

where $H(D)$ is matrix of elements as polynomials of D .

* What will happen when $G(D)$ does not have an FIR inverse?

Ex: Consider $G(D) = (1+D, 1+D^2)$

$$\Rightarrow \gcd\{1+D, 1+D^2\} = 1+D$$

$\Rightarrow$ It doesn't have an FIR inverse.

$$\text{Let } u(D) = \frac{1}{1+D} = 1+D+D^2+\dots$$

$$\Rightarrow u = (1, 1, 1, \dots)$$

Then the output $V(D) = (1, 1+D)$

$$\Rightarrow v = (1, 1, 1)$$

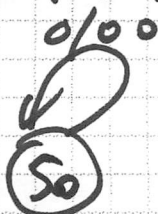
This means that the codeword only contains three nonzero bits although the information sequence has infinite weight. If this codeword is transmitted over a BSC and ~~the~~ these 3 nonzero bits are changed to zeros by the channel noise, then the received sequence will be all zeros.

$\Rightarrow$ No way to recover information

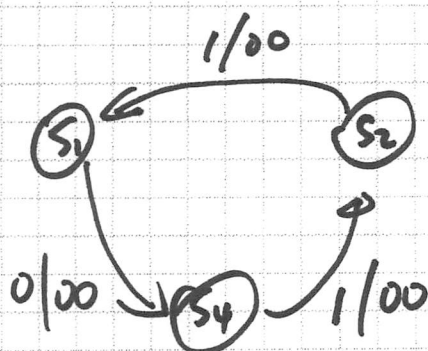
$\Rightarrow$ It causes catastrophic error $\Rightarrow$ Catastrophic code.



In terms of the state diagram, a code is catastrophic iff the state diagram contains a loop of zero weight other than the self-loop around the state S_0 .



or





§3. Convolutional Code Decoding and Viterbi Algorithm

(A.) Introduction

Hard decision decoding: received signal is quantized to the original signal levels, such as binary (usually loses information).

Soft decision decoding: decoding directly on the received signal (no quantization before decoding) (no information is lost before decoding).

Soft decision decoding is better than hard decision decoding (performance), about 2 dB.

Let $\underline{x}^{(i)} = (x_1^{(i)}, \dots, x_L^{(i)})$ be complex symbols for message i . Let $p(t)$ be the pulse waveform. (such as BPSK ± 1)

Then the corresponding waveform for the i th message is

$$s^{(i)} = \sum_{r=1}^L x_r^{(i)} p(t - rT).$$

Let $r(t)$ be the received signal, then the sampled response of the matched filter is (at the sampling $t = rT$).

$$y_r = \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT) d\alpha$$

where

$$y_r = \int_{-\infty}^{\infty} r(\alpha) p(\alpha - rT) d\alpha$$



If $p(t)$ is a Nyquist pulse with unit energy, the energy of the signal $S^{(i)}(t)$ is

$$|S^{(i)}|^2 = \sum_{r=-L}^L |x_r^{(i)}|^2$$

For MFSK channel, the maximum Likelihood receiver will select the sequence $\underline{x}^{(i)}$ that maximizes

$$J^{(i)} = 2 \sum_{r=-L}^L \operatorname{Re} \{ x_r^{(i)} y_r^* \} - \sum_{r=-L}^L |x_r^{(i)}|^2,$$

which is called the metric of the sequence $\underline{x}^{(i)}$ and this metric is to be maximized over all allowable choices of $\underline{x}^{(i)}$.

The partial metric at time n is

$$J_n^{(i)} = 2 \sum_{r=-L}^n \operatorname{Re} \{ x_r^{(i)} y_r^* \} - \sum_{r=-L}^n |x_r^{(i)}|^2 \triangleq L(\underline{x}^{(i)}, \underline{y})$$

Then

$$J_n^{(i)} = J_{n-1}^{(i)} + 2 \operatorname{Re} \{ x_n^{(i)} y_n^* \} - |x_n^{(i)}|^2$$

past metric

(current) branch metric

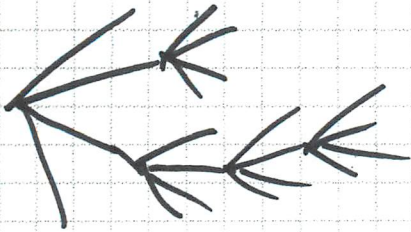
(Likelihood function.)

This simply tells us a tree structure to evaluate the metrics for all the allowable signal sequences.



The tree has 2^k branches leaving each node.

The complexity is exponential
in terms of L (length).
(depth of the tree).



(B) The Stack Algorithm

We start exploring the tree and store the metric for every node explored.

At each stage now, we simply extend the node with the largest such metric.

This is, in essence, the stack algorithm proposed (first) by Zigangirov and Jelinek.

The stack algorithm is one of the many variations of what has become known as sequential decoding of trellis code

⚡ was introduced by Wozencraft.



Stack Algorithm:

Step 1: Initialize an empty stack S of visited nodes and their metric. Deposit the empty partial sequence $()$ at the top of the stack with its metric $L((), \underline{y}) = 0$.

Step 2: Extend the node corresponding to the top entry $\{\underline{\tilde{x}}_{\text{top}}, L(\underline{\tilde{x}}_{\text{top}}, \underline{y})\}$ by forming $L(\underline{\tilde{x}}_{\text{top}}, \underline{y}) - |x_r - y_r|^2 / N_0$ for all 2^k extensions of $\underline{\tilde{x}}_{\text{top}}$ to $(\underline{\tilde{x}}_{\text{top}}, x_r) \triangleq \underline{\tilde{x}}^{(i)}$. Delete $\{\underline{\tilde{x}}_{\text{top}}, L(\underline{\tilde{x}}_{\text{top}}, \underline{y})\}$ from the stack.

Step 3: Place the new entries $\{\underline{\tilde{x}}^{(i)}, L(\underline{\tilde{x}}^{(i)}, \underline{y})\}$ from Step 2 into the stack such that the stack remains ordered with the entry with the largest metric at the top of the stack.

Step 4: If the top entry of the stack is a path to one of the terminal nodes at depth l , stop and select $\underline{x}_{\text{top}}$ as the transmitted symbol sequence. Otherwise go to Step 2.



There are some practical problems associated with the stack algorithm:

- The number of computations that the algorithm performs depends greatly on channel quality.

If the channel is very noisy, the received sample value y_r will be unreliable and numerous possible paths will have similar metrics, which have to be stored in the stack.

This causes a computational speed problem

① storage problem
(buffer overflow)

② sorting becomes more complex.

(C). The Fano Algorithm

(good for low rate)

1963

Unlike the stack algorithm, the Fano algorithm is a depth-first tree search procedure. This algorithm stores only one path and this, essentially, requires no storage.

Since the Fano algorithm only stores one path, it must allow for back-tracking. (There can be no jumping between non connected nodes).



Moves in the tree by proceeding from one node to a successor node with a suitably large metric. If no such node can be found, the algorithm backtracks and looks for other branches from previously visited nodes.

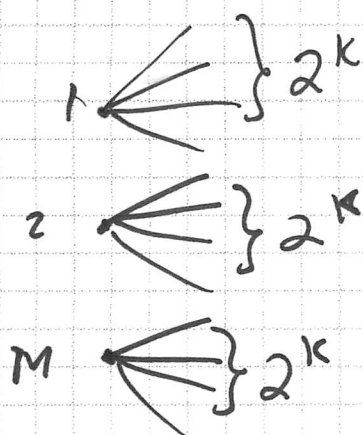
(D) The M-algorithm

M-algorithm is a breadth-first algorithm. It keeps M candidate paths at each iteration and deletes all others from further consideration.

At each time unit, the algorithm extends all M currently held nodes to form $2^k M$ new nodes, from among which those M with the best metrics are retained.

Step 1: Initialize an empty list L of candidate paths and their metrics. Deposit the zero-length path $()$ with its metric $L((), y) = 0$ in the list. Set $n = -1$.

Step 2: Extend 2^k partial paths $\tilde{x}^{(i)} \rightarrow (\tilde{x}^{(i)}, x_n^{(i)})$ from each of the at-most M paths $\tilde{x}^{(i)}$ in the list. Delete the entries in the original list.



Step 3: Find the at-most M partial paths with the best metrics among the extensions and save them in the list L .

Delete the rest of the extensions. Set $n = n + 1$.

(From two or more extensions leading to the same state all but the one with the best metric may be discarded)

Step 4: If at the end of the tree (i.e., $n \geq l$) release the output symbols corresponding to the path with the best metric in the list L ; otherwise go to Step 2.

The decoding problem with the M -algorithm is the loss of the correct path from the list of candidates, after which the algorithm might spend a long time re-synchronizing.

The computational load of the M -algorithm is independent of the size of the code & it is proportional to M .

It is also independent of the channel quality.