S.72-3320 Advanced Digital Communication (4 cr)

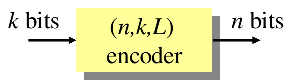
Convolutional Codes

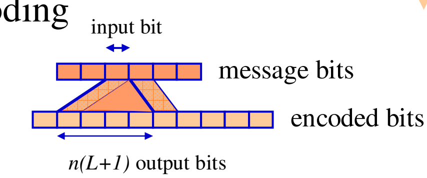
Targets today

- Why to apply convolutional coding?
- Defining convolutional codes
- Practical encoding circuits
- Defining quality of convolutional codes
- Decoding principles
- Viterbi decoding

Timo O. Korhonen, HUT Communication Laboratory

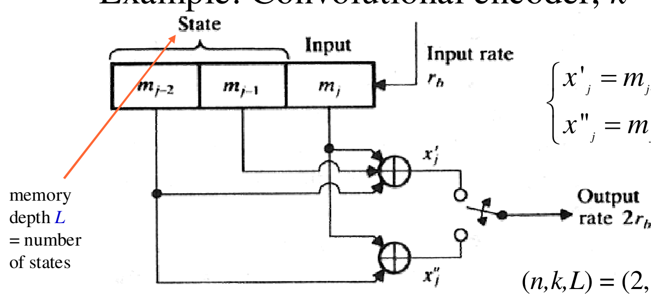
Convolutional encoding





- Convolutional codes are applied in applications that require good performance with low implementation complexity. They operate on code streams (not in blocks)
- Convolution codes have memory that utilizes previous bits to encode or decode following bits (block codes are memoryless)
- Convolutional codes are denoted by (n,k,L), where L is code (or encoder) **Memory depth** (number of register stages)
- **Constraint length $C=n(L+1)$** is defined as the number of encoded bits a message bit can influence to
- Convolutional codes achieve good performance by expanding their memory depth

Example: Convolutional encoder, $k = 1$, $n = 2$



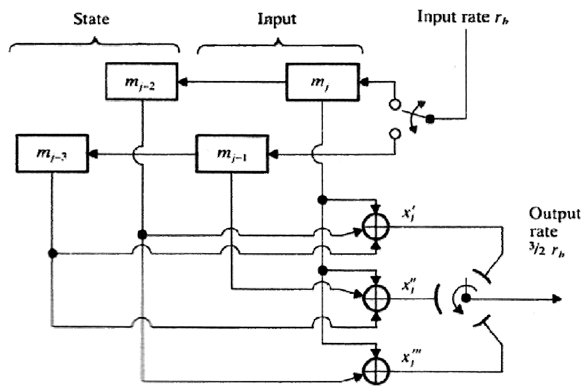
$$\begin{cases} x'_j = m_{j-2} \oplus m_{j-1} \oplus m_j \\ x''_j = m_{j-2} \oplus m_j \end{cases}$$

$(n,k,L) = (2,1,2)$ encoder

$$x_{out} = x'_1 x''_1 x'_2 x''_2 x'_3 x''_3 \ldots$$

- Convolutional encoder is a **finite state machine** (FSM) processing information bits in a serial manner
- Thus the generated code is a function of input and the state of the FSM
- In this $(n,k,L) = (2,1,2)$ encoder each message bit influences a span of $C= n(L+1)=6$ successive output bits = **constraint length C**
- Thus, for generation of n-bit output, we require in this example n shift registers in $k = 1$ convolutional encoder

Example: $(n,k,L)=(3,2,1)$ Convolutional encoder



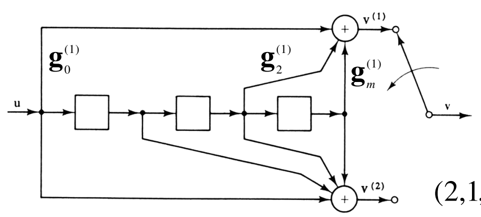
$$x'_j = m_{j-3} \oplus m_{j-2} \oplus m_j$$
$$x''_j = m_{j-3} \oplus m_{j-1} \oplus m_j$$
$$x'''_j = m_{j-2} \oplus m_j$$

- After each new block of k input bits follows a transition into new state
- Hence, from each input state transition, 2^k different output states may follow
- Each message bit influences a span of $C = n(L+1) = 3(1+1) = 6$ successive output bits

Generator sequences

k bits → (n,k,L) encoder → n bits

- (n,k,L) Convolutional code can be described by the generator sequences $\mathbf{g}^{(1)}, \mathbf{g}^{(2)}, \ldots \mathbf{g}^{(n)}$ that are the impulse responses for each coder n output branches:



(2,1,2) encoder

$$\mathbf{g}^{(n)} = [g_0^{(n)} \quad g_1^{(n)} \quad \cdots \quad g_m^{(n)}]$$

$$\begin{cases} \mathbf{g}^{(1)} = [1\,0\,1\,1] \\ \mathbf{g}^{(2)} = [1\,1\,1\,1] \end{cases}$$ Note that the generator sequence length exceeds register depth always by 1

- Generator sequences specify convolutional code completely by the associated generator matrix
- Encoded convolution code is produced by matrix multiplication of the input and the generator matrix

Convolution point of view in encoding and generator matrix

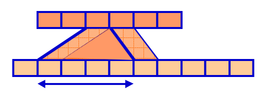
- Encoder outputs are formed by **modulo-2 discrete convolutions**:

$$x \otimes y(u) = \sum_{k \in A_{xy}} x(k)\, y(u-k)$$

$$\mathbf{v}^{(1)} = \mathbf{u} * \mathbf{g}^{(1)}, \mathbf{v}^{(2)} = \mathbf{u} * \mathbf{g}^{(2)} \ldots \mathbf{v}^{(j)} = \mathbf{u} * \mathbf{g}^{(j)}$$

where **u** is the information sequence:

$$\mathbf{u} = (u_0, u_1, \cdots)$$

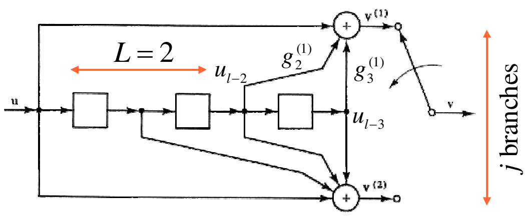


- Therefore, the l:th bit of the j:th output branch is*

$$v_l^{(j)} = \sum_{i=0}^{m} u_{l-i} g_i^{(j)} = u_l g_0^{(j)} + u_{l-1} g_1^{(j)} + \ldots + u_{l-m} g_m^{(j)}$$

where $m = L+1, u_{l-i} \triangleq 0, l < i$

- Hence, for this circuit the following equations result, (assume: $\begin{pmatrix} \mathbf{g}^{(1)} = [1\,0\,1\,1] \\ \mathbf{g}^{(2)} = [1\,1\,1\,1] \end{pmatrix}$



$$\begin{cases} v_l^{(1)} = u_l + u_{l-2} + u_{l-3} \\ v_l^{(2)} = u_l + u_{l-1} + u_{l-2} + u_{l-3} \end{cases}$$

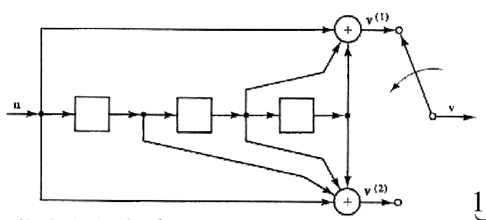
encoder output:

$$\mathbf{v} = [v_0^{(1)} v_0^{(2)} \; v_1^{(1)} v_1^{(2)} \; v_2^{(1)} v_2^{(2)} \ldots]$$

Timo O. Korhonen, HUT Communication Laboratory

*note that u is reversed in time as in the definition of convolution top right

Example: Using generator matrix



$$\begin{pmatrix} \mathbf{g}^{(1)} = [1\,0\,1\,1] \\ \mathbf{g}^{(2)} = [1\,1\,1\,1] \end{pmatrix}$$

$$\underbrace{\underbrace{11 \oplus 00}_{11} \oplus \underbrace{01 \oplus 11}_{10}}_{01} = 01$$

If **u** = (1 0 1 1 1), then

v = **uG**

$$v_l^{(j)} = u_l g_0^{(j)} + u_{l-1} g_1^{(j)} + \ldots + u_{l-m} g_m^{(j)}$$

$$= \underset{u_{l-m}}{(1 \;\; 0 \;\; 1 \;\; 1 \;\; 1)} \begin{array}{c} \mathbf{g}_0^{(1)}\mathbf{g}_0^{(2)} \;\; \mathbf{g}_1^{(1)}\mathbf{g}_1^{(2)} \qquad \mathbf{g}_m^{(1)}\mathbf{g}_m^{(2)} \\ \begin{bmatrix} 11 & 01 & 11 & 11 & & & & \\ & 11 & 01 & 11 & 11 & & & \\ & & 11 & 01 & 11 & 11 & & \\ & & & 11 & 01 & 11 & 11 & \\ & & & & 11 & 01 & 11 & 11 \end{bmatrix} \end{array}$$

$$= (1\;1,\; 0\;1,\; 0\;0,\; \boxed{0\;1,}\; 0\;1,\; 0\;1,\; 0\;0,\; 1\;1),$$

Verify that you can obtain the result shown!

Timo O. Korhonen, HUT Communication Laboratory

Let the information sequence $\mathbf{u} = (1\ 0\ 1\ 1\ 1)$. Then the output sequences are

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

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

and the code word is

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

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

$$\mathbf{G} = \begin{bmatrix} g_0^{(1)}g_0^{(2)} & g_1^{(1)}g_1^{(2)} & g_2^{(1)}g_2^{(2)} & \cdots & g_m^{(1)}g_m^{(2)} & & \\ & g_0^{(1)}g_0^{(2)} & g_1^{(1)}g_1^{(2)} & \cdots & g_{m-1}^{(1)}g_{m-1}^{(2)} & g_m^{(1)}g_m^{(2)} & \\ & & g_0^{(1)}g_0^{(2)} & \cdots & g_{m-2}^{(1)}g_{m-2}^{(2)} & g_{m-1}^{(1)}g_{m-1}^{(2)} & g_m^{(1)}g_m^{(2)} \\ & & \ddots & & & & \ddots \end{bmatrix},$$

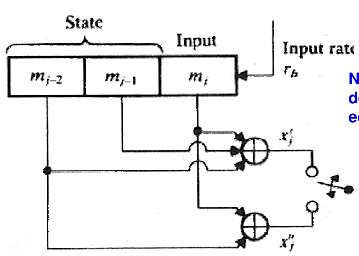
where the blank areas are all zeros, the encoding equations can be rewritten in matrix form as

$$\mathbf{v} = \mathbf{u}\mathbf{G},$$

where all operations are modulo-2. $\mathbf{G}$ is called the *generator matrix* of the code. Note that each row of $\mathbf{G}$ is identical to the preceding row but shifted $n = 2$ places to the right, and that $\mathbf{G}$ is a semi-infinite matrix, corresponding to the fact that the information sequence $\mathbf{u}$ is of arbitrary length. If $\mathbf{u}$ has finite length L, then $\mathbf{G}$ has L rows and $2(m + L)$ columns, and $\mathbf{v}$ has length $2(m + L)$.

S.Lin, D.J. Costello: Error Control Coding, II ed, p. 456

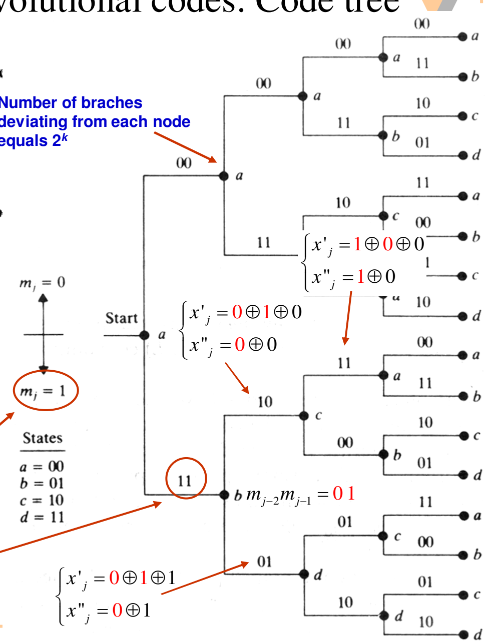
Representing convolutional codes: Code tree



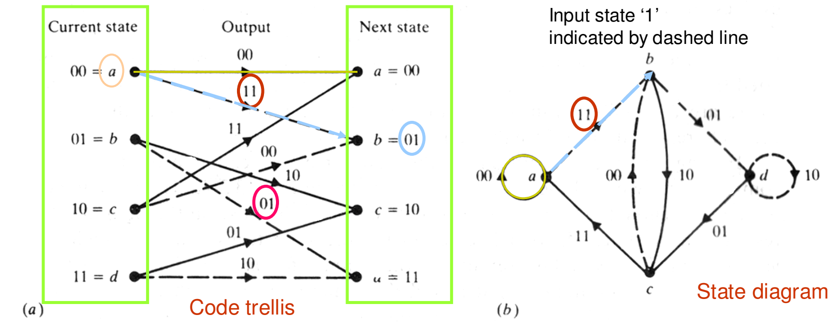
$(n,k,L) = (2,1,2)$ encoder

$$\begin{cases} x'_j = m_{j-2} \oplus m_{j-1} \oplus m_j \\ x''_j = m_{j-2} \oplus m_j \end{cases}$$

$$x_{out} = x'_1 x''_1 x'_2 x''_2 x'_3 x''_3 \ldots$$



Representing convolutional codes compactly: code trellis and state diagram



Input	1	1	0	1	1	1	0	0	1	0	0	0	
State	a	b	d	c	b	d	d	c	a	b	c	a	a
Output	11	01	01	00	01	10	01	11	11	10	11	00	

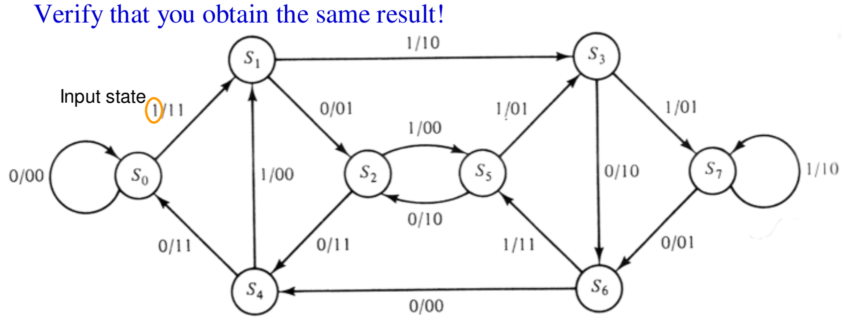
(c)

Shift register states

Inspecting state diagram: Structural properties of convolutional codes

- Each new block of k input bits causes a transition into new state
- Hence there are 2^k branches leaving each state
- Assuming encoder zero initial state, encoded word for any input of k bits can thus be obtained. For instance, below for **u**=(1 1 1 0 1), encoded word **v**=(1 1, 1 0, 0 1, 0 1, 1 1, 1 0, 1 1, 1 1) is produced:

Verify that you obtain the same result!



- encoder state diagram for (n,k,L)=(2,1,2) code
- note that the number of states is 8 = 2^{L+1} => L = 2 (two state bits)

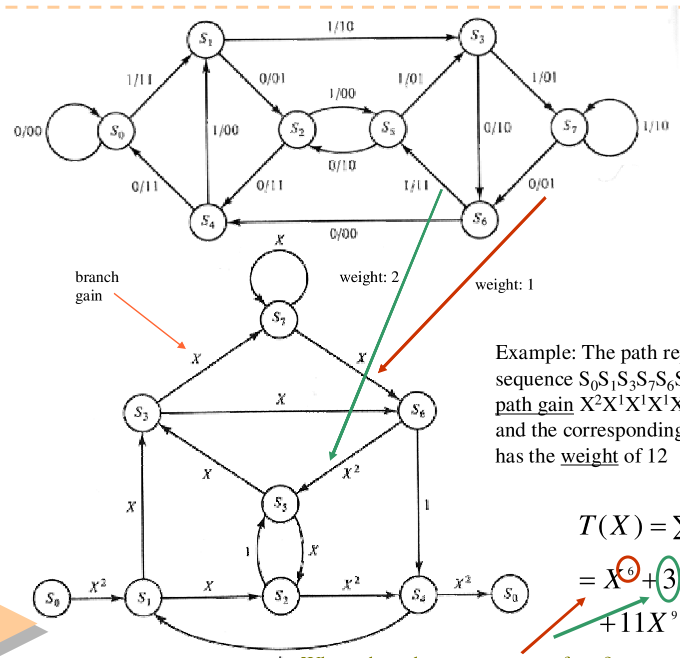
Code weight, path gain, and generating function

- The state diagram can be modified to yield information on code distance properties (= tells how good the code is to detect or correct errors)
- Rules (example on the next slide):
  - (1) Split S_0 into initial and final state, remove self-loop
  - (2) Label each branch by the branch gain X^i. Here i is the **weight*** of the n encoded bits on that branch
  - (3) Each path connecting the initial state and the final state represents a nonzero code word that diverges and re-emerges with S_0 only once
- The **path gain** is the product of the branch gains along a path, and the **weight** of the associated code word is the power of X in the path gain
- *Code weigh distribution* is obtained by using a weighted gain formula to compute its **generating function** (input-output equation)

$$T(X) = \sum_i A_i X^i$$

where A_i is the number of encoded words of weight i

*In linear codes, weight is the number of '1':s in the encoder output



Example: The path representing the state sequence $S_0S_1S_3S_7S_6S_5S_2S_4S_0$ has the path gain $X^2X^1X^1X^1X^2X^1X^2X^2=X^{12}$ and the corresponding code word has the weight of 12

$$T(X) = \sum_i A_i X^i$$
$$= X^6 + 3X^7 + 5X^8 + 11X^9 + 25X^{10} + \ldots.$$

Where does these terms come from?

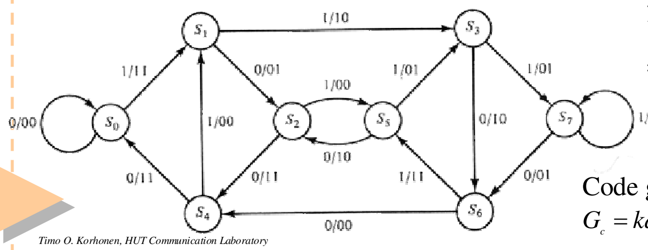
Distance properties of convolutional codes

- Code strength is measured by the **minimum free distance**:

$$d_{free} = \min\{d(\mathbf{v}', \mathbf{v}''): \mathbf{u}' \neq \mathbf{u}''\}$$

where **v**' and **v**'' are the encoded words corresponding information sequences **u**' and **u**''. Code can correct up to $t < d_{free}/2$ errors.

- The minimum free distance d_{free} denotes:
  - The minimum weight of all the paths in the state diagram that diverge from and remerge with the all-zero state S_0
  - The lowest power of the **code-generating function *T(X)***



$$T(X) = \sum_i A_i X^i$$
$$= X^6 + 3X^7 + 5X^8$$
$$+ 11X^9 + 25X^{10} + \ldots$$
$$\Rightarrow d_{free} = 6$$

Code gain*:
$$G_c = kd_{free}/(2n) = R_c d_{free}/2 > 1$$

* for derivation, see Carlson's, p. 583

Coding gain for some selected convolutional codes

- Here is a table of some selected convolutional codes and their code gains $R_c d_{free}/2$ expressed for hard decoding also by

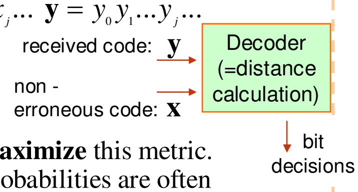
$$\gamma = 10\log_{10}(R_c d_{free}/2) \text{ dB}$$

n	k	R_c	L	d_f	$R_c d_f/2$
4	1	1/4	3	13	1.63
3	1	1/3	3	10	1.68
2	1	1/2	3	6	1.50
			6	10	2.50
			9	12	3.00
3	2	2/3	3	7	2.33
4	3	3/4	3	8	3.00

Decoding of convolutional codes

- Maximum likelihood decoding of convolutional codes means finding the code branch in the code trellis that was most likely transmitted
- Therefore maximum likelihood decoding is based on calculating code Hamming distances for each branch potentially forming encoded word
- Assume that the information symbols applied into an AWGN channel are equally alike and independent
- Let's denote by **x** encoded symbols (no errors) and by **y** received (potentially erroneous) symbols: $\mathbf{x} = x_0 x_1 x_2 \ldots x_j \ldots$ $\mathbf{y} = y_0 y_1 \ldots y_j \ldots$
- Probability to decode the symbols is then

$$p(\mathbf{y},\mathbf{x}) = \prod_{j=0}^{\infty} p(y_j \mid x_j)$$

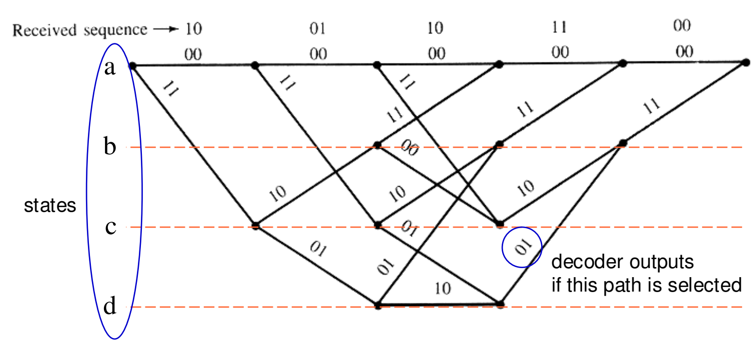


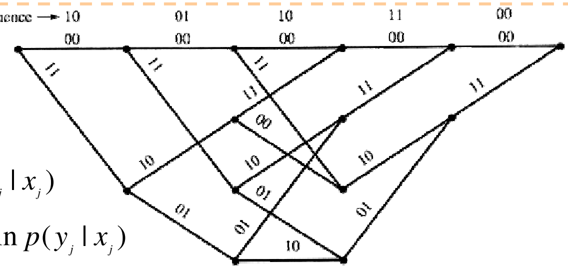
- The most likely path through the trellis will **maximize** this metric. Often ln() is taken from both sides, because probabilities are often small numbers, yielding: $$\ln\{p(\mathbf{y},\mathbf{x})\} = \sum_{j=1}^{\infty} \ln\{p(y_j | x_{mj})\}$$

(note this corresponds equavalently also the smallest Hamming distance)

Example of exhaustive maximal likelihood detection

- Assume a three bit message is transmitted and encoded by (2,1,2) convolutional encoder. To clear the decoder, two zero-bits are appended after message. Thus 5 bits are encoded resulting 10 bits of code. Assume channel error probability is $p = 0.1$. After the channel 10,01,10,11,00 is produced (including some errors). What comes after the decoder, e.g. what was most likely the transmitted code and what were the respective message bits?





$$p(\mathbf{y},\mathbf{x}) = \prod_{j=0}^{\infty} p(y_j \mid x_j)$$

$$\ln p(\mathbf{y},\mathbf{x}) = \sum_{j=0}^{\infty} \ln p(y_j \mid x_j)$$

Solution: Decoding is accomplished using Figure 7-10 by calculating the path metric of (7-53) for all of the eight distinct paths through the trellis and choosing the path with the largest metric. Decoding can also be accomplished by calculating the Hamming distance between the received sequence and the path sequence for all eight paths and choosing the path with the minimum Hamming distance. Both methods will be illustrated. The incremental (i.e., corresponding to one transmitted symbol) log-likelihood symbol metrics for the specified channel are

$$\ln[p(0 \mid 0)] = \ln[p(1 \mid 1)] = \ln(0.9) = -0.11$$

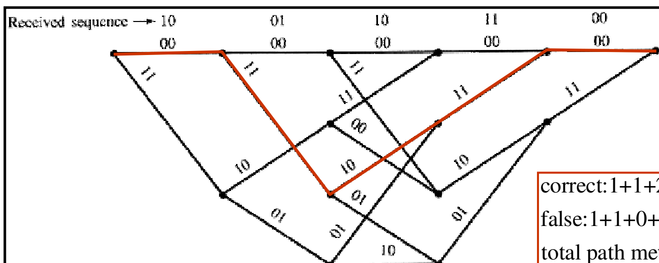
$$\ln[p(1 \mid 0)] = \ln[p(0 \mid 1)] = \ln(0.1) = -2.30$$

weight for prob. to receive bit in-error

Consider the all-zero path. The code sequence on this path differs from the received sequence in five positions. Thus the path metric is

$$5(-2.3) + 5(-0.11) = -12.05$$

errors correct



correct: $1+1+2+2+2=8; 8\cdot(-0.11) = -0.88$

false: $1+1+0+0+0=2; 2\cdot(-2.30) = -4.6$

total path metric: -5.48

The Hamming distance between this path and the received sequence is 5. All paths (specified by the encoder input bits) and their path metrics and Hamming distances are listed below.

The largest metric, verify that you get the same result!

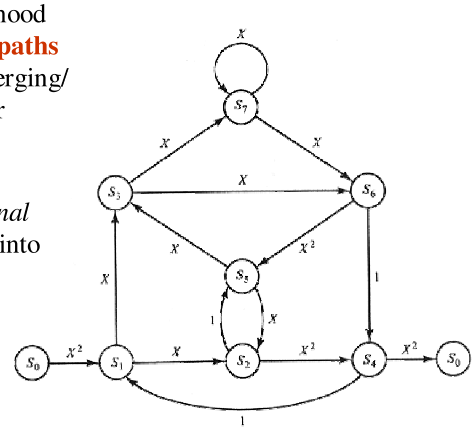
Received sequence: 10, 01, 10, 11, 00

Note also the Hamming distances!

Path	Code Sequence	Path Metric	Hamming Distance
0, 0, 0, 0, 0	00, 00, 00, 00, 00	−12.05	5
0, 0, 1, 0, 0	00, 00, 11, 10, 11	−11.24	6
0, 1, 0, 0, 0	00, 11, 10, 11, 00	−5.48	2
0, 1, 1, 0, 0	00, 11, 01, 01, 11	−16.43	7
1, 0, 0, 0, 0	11, 10, 11, 00, 00	−14.24	6
1, 0, 1, 0, 0	11, 10, 00, 10, 11	−16.43	7
1, 1, 0, 0, 0	11, 01, 01, 11, 00	−7.67	3
1, 1, 1, 0, 0	11, 01, 10, 01, 11	−9.86	4

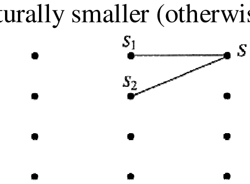
The Viterbi algorithm

- Problem of optimum decoding is to find the minimum distance path from the initial state back to the initial state (below from S_0 to S_0). The minimum distance is one of the sums of all path metrics from S_0 to S_0
- Exhaustive maximum likelihood method must search **all the paths** in phase trellis (2^k paths emerging/ entering from 2^{L+1} states for an (n,k,L) code)
- The Viterbi algorithm gets improvement in *computational efficiency* via concentrating into **survivor paths** of the trellis

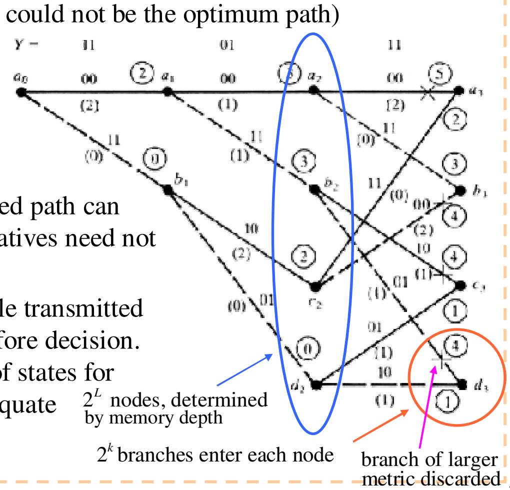


The survivor path

- Assume for simplicity a convolutional code with k=1, and thus up to 2^k = 2 branches can enter each state in trellis diagram
- Assume optimal path passes S. Metric comparison is done by adding the metric of S_1 and S_2 to S. At the survivor path the accumulated metric is naturally smaller (otherwise it could not be the optimum path)



- For this reason the non-survived path can be discarded -> all path alternatives need not to be further considered
- Note that in principle the whole transmitted sequence must be received before decision. However, in practice storing of states for input length of $5L$ is quite adequate

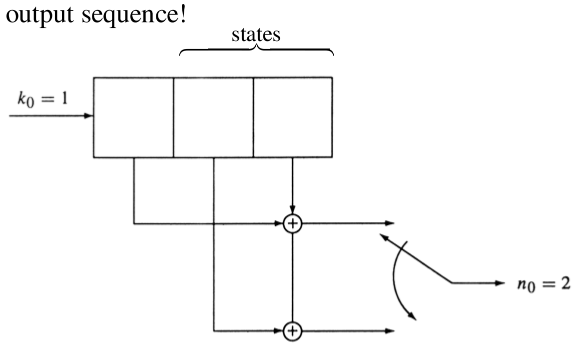


Example of using the Viterbi algorithm

- Assume the received sequence is

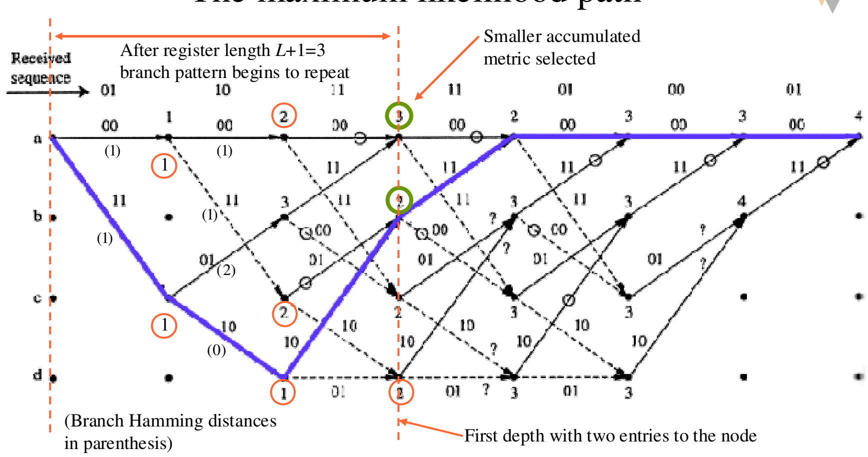
$$y = 01101111010001$$

and the (n,k,L)=(2,1,2) encoder shown below. Determine the Viterbi decoded output sequence!



(Note that for this encoder code rate is 1/2 and memory depth equals $L = 2$)

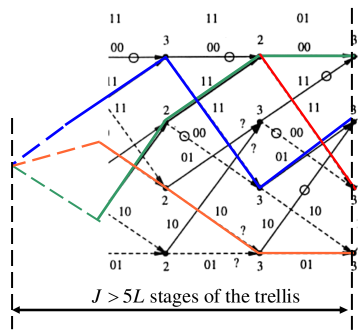
The maximum likelihood path



The decoded ML code sequence is 11 10 10 11 00 00 00 whose Hamming distance to the received sequence is 4 and the respective decoded sequence is 1 1 0 0 0 0 0 (why?). Note that this is the minimum distance path. (Black circles denote the deleted branches, dashed lines: '1' was applied)

How to end-up decoding?

- In the previous example it was assumed that the register was finally filled with zeros thus finding the minimum distance path
- In practice with long code words zeroing requires feeding of long sequence of zeros to the end of the message bits: this wastes channel capacity & introduces delay
- To avoid this *path memory truncation* is applied:
  - Trace all the surviving paths to the depth where they merge
  - Figure right shows a common point at a memory depth J
  - J is a random variable whose applicable magnitude shown in the figure ($5L$) has been experimentally tested for negligible error rate increase
  - Note that this also introduces the delay of $5L$!



Lessons learned

- You understand the differences between cyclic codes and convolutional codes
- You can create state diagram for a convolutional encoder
- You know how to construct convolutional encoder circuits based on knowing the generator sequences
- You can analyze code strengths based on known code generation circuits / state diagrams or generator sequences
- You understand how to realize maximum likelihood convolutional decoding by using exhaustive search
- You understand the principle of Viterbi decoding