EENG 373

Communication Systems II

Convolutional Codes

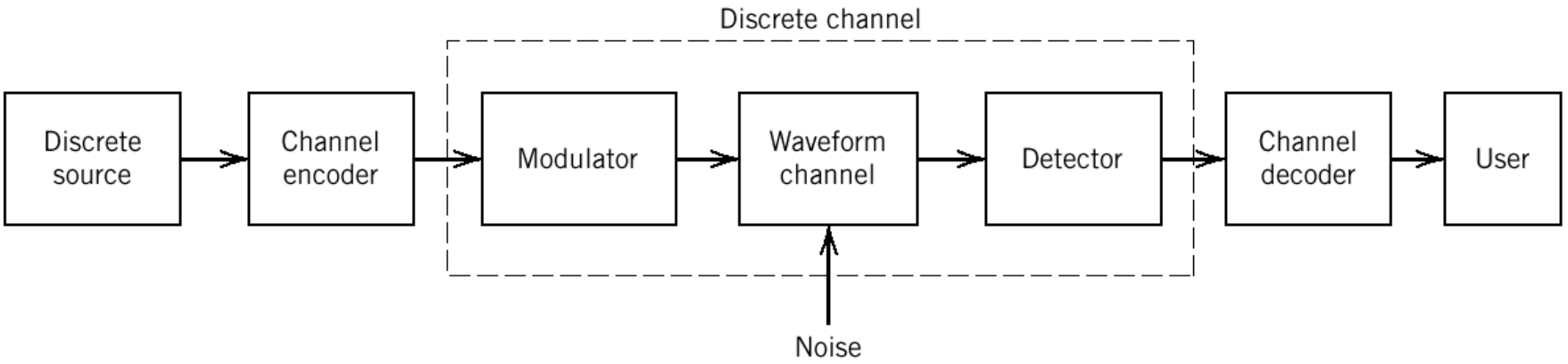
Prof. Mohab A. Mangoud

Professor of Wireless Communications

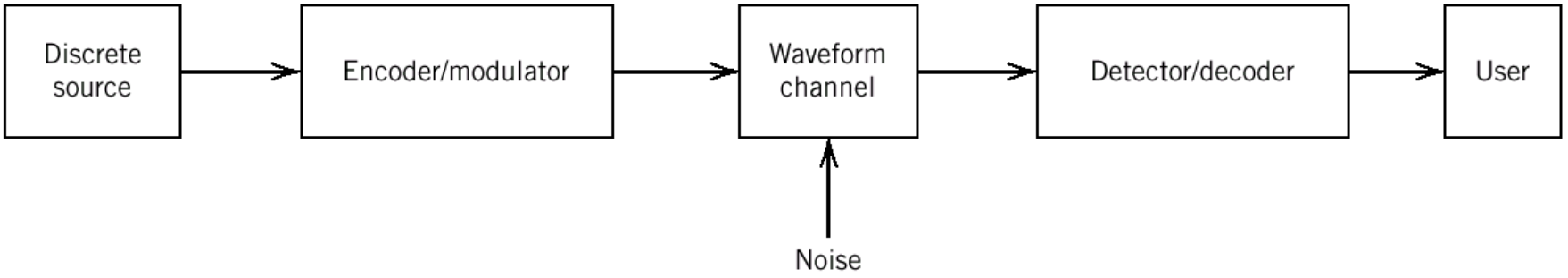
University of Bahrain, College of Engineering,
Department of Electrical and Electronics Engineering,
P.O. Box 32038, Isa Town,Kingdom of Bahrain

Office: +973 17876033/6261
Email : mmangoud@uob.edu.bh
URL: http://mangoud.com

Figure 10.1
Simplified models of digital communication system.
(*a*) Coding and modulation performed separately.
(*b*) Coding and modulation combined.



(*a*)



(*b*)

code word. There are applications, however, where the message bits come in *serially* rather than in large blocks, in which case the use of a buffer may be undesirable. In such situations, the use of *convolutional coding* may be the preferred method. A convolutional coder generates redundant bits by using *modulo-2 convolutions*, hence the name.

The encoder of a binary convolutional code with rate $1/n$, measured in bits per symbol, may be viewed as a *finite-state machine* that consists of an M-stage shift register with prescribed connections to n modulo-2 adders, and a multiplexer that serializes the outputs of the adders. An L-bit message sequence produces a coded output sequence of length $n(L + M)$ bits. The *code rate* is therefore given by

$$r = \frac{L}{n(L + M)} \quad \text{bits/symbol} \tag{10.53}$$

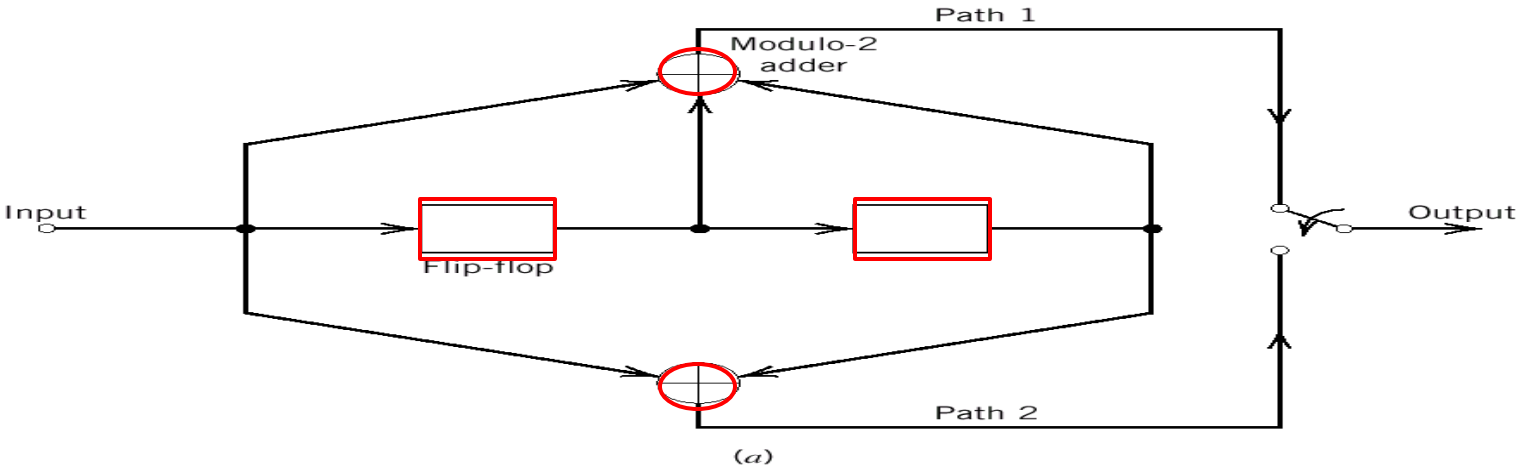
Typically, we have $L \gg M$. Hence, the code rate simplifies to

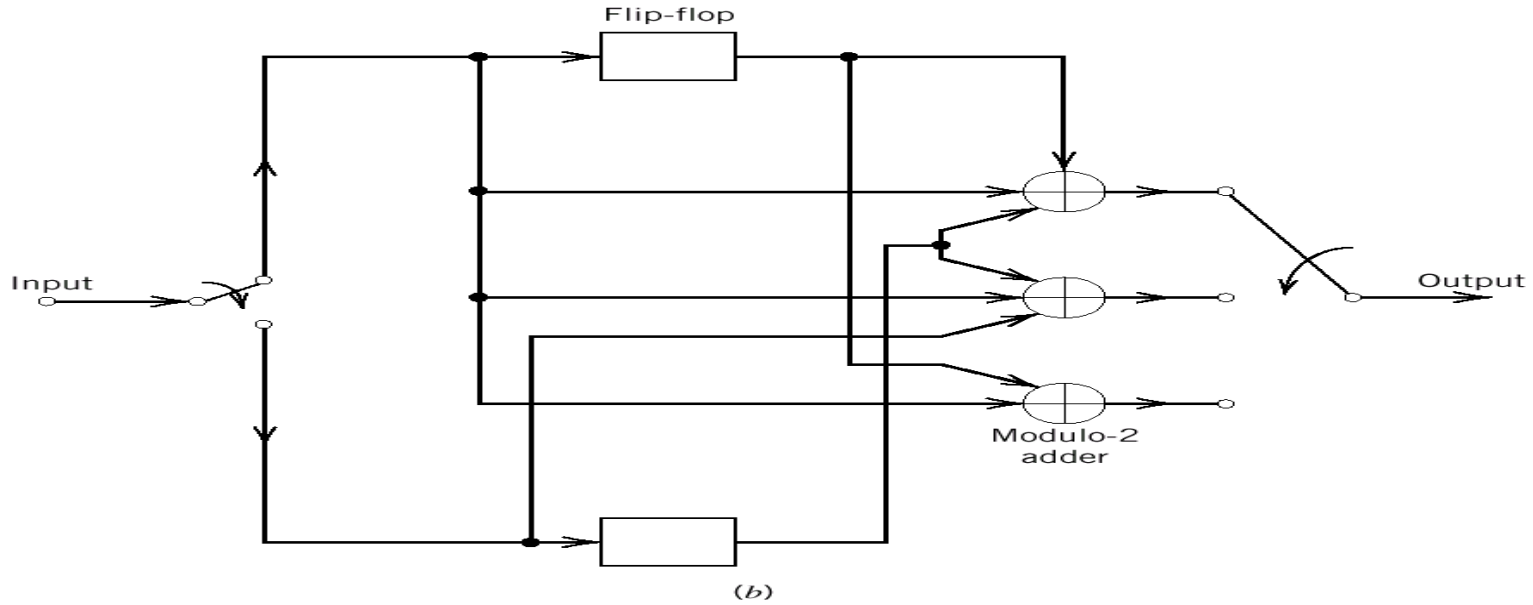
$$r \simeq \frac{1}{n} \quad \text{bits/symbol} \tag{10.54}$$

The *constraint length* of a convolutional code, expressed in terms of message bits, is defined as the number of shifts over which a single message bit can influence the encoder output. In an encoder with an M-stage shift register, the *memory* of the encoder equals M message bits, and $K = M + 1$ shifts are required for a message bit to enter the shift register and finally come out. Hence, the constraint length of the encoder is K.

Figure 10.13*a* shows a convolutional encoder with $n = 2$ and $K = 3$. Hence, the code rate of this encoder is 1/2. The encoder of Figure 10.13*a* operates on the incoming message sequence, one bit at a time.

We may generate a binary convolutional code with rate k/n by using k separate shift registers with prescribed connections to n modulo-2 adders, an input multiplexer and an output multiplexer. An example of such an encoder is shown in Figure 10.13*b*, where $k = 2$, $n = 3$, and the two shift registers have $K = 2$ each. The code rate is 2/3. In this second example, the encoder processes the incoming message sequence two bits at a time.



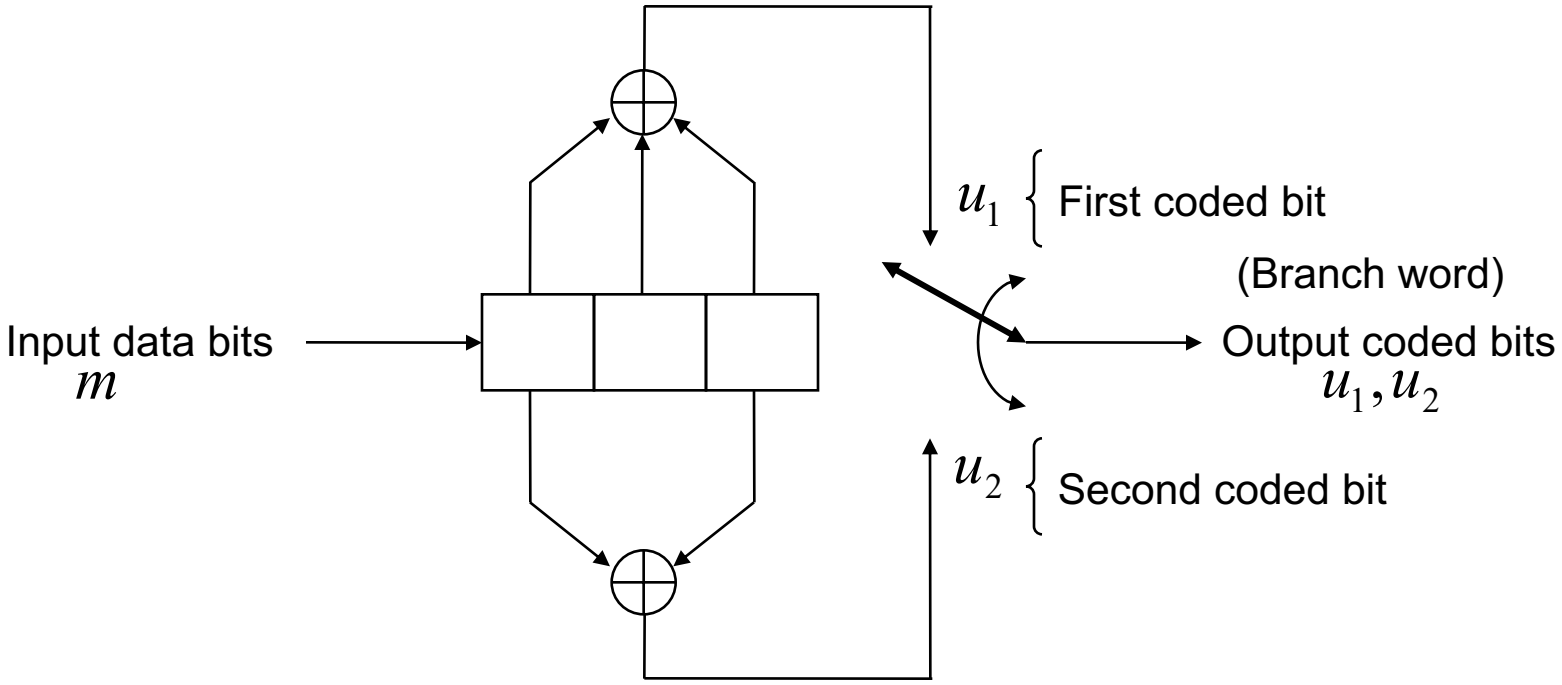


(*a*) Constraint length-3, rate -$\frac{1}{2}$ convolutional encoder.

(*b*) Constraint length-2, rate-$\frac{2}{3}$ convolutional encoder.

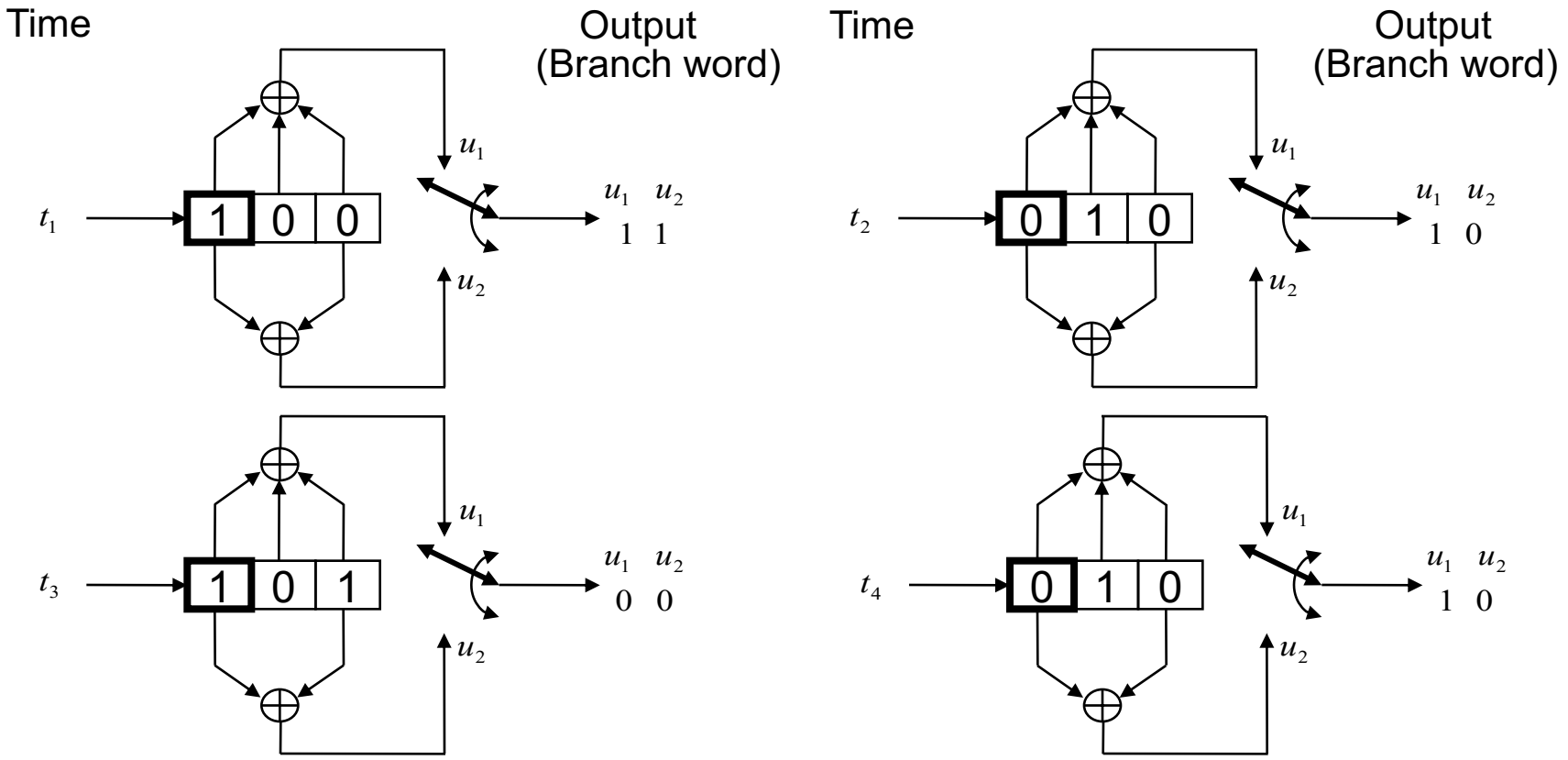
A Rate ½ Convolutional encoder

- Convolutional encoder (rate ½, K=3)
  - 3 shift-registers where the first one takes the incoming data bit and the rest, form the memory of the encoder.

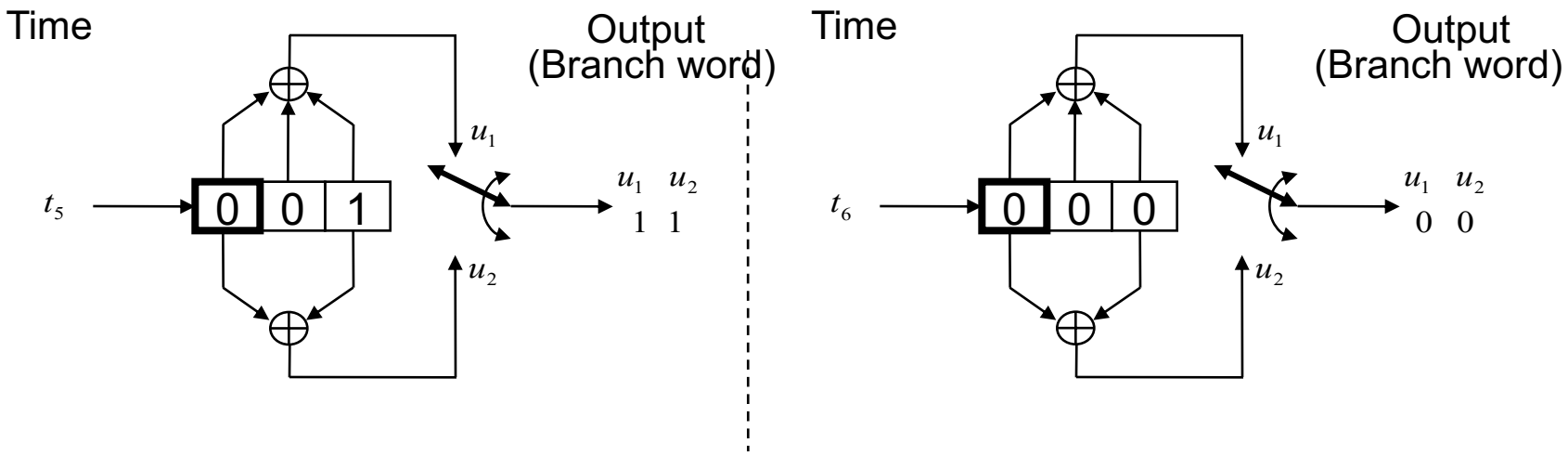


A Rate ½ Convolutional encoder

Message sequence: $\mathbf{m} = (101)$



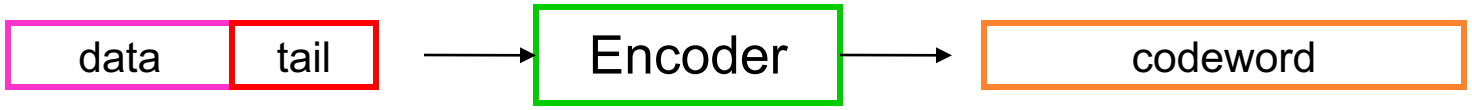
A Rate ½ Convolutional encoder



$\mathbf{m} = (101)$ → Encoder → $\mathbf{U} = (11 \quad 10 \quad 00 \quad 10 \quad 11)$

Effective code rate

- Initialize the memory before encoding the first bit (all-zero)
- Clear out the memory after encoding the last bit (all-zero)
  - Hence, a tail of zero-bits is appended to data bits.



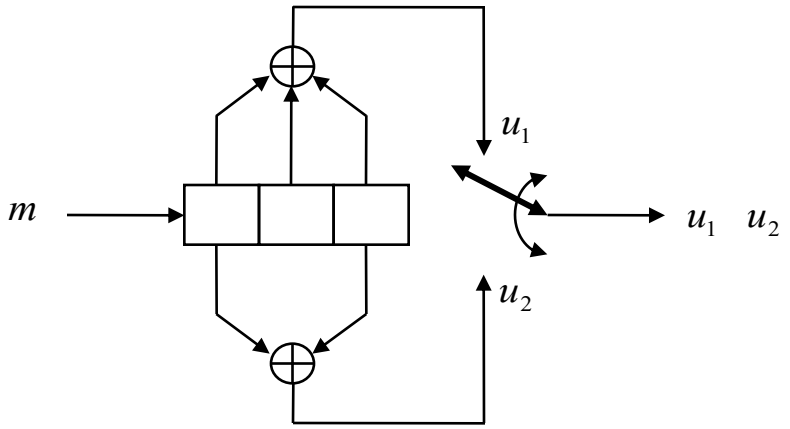
- Effective code rate :
  - L is the number of data bits and $k=1$ is assumed:

$$R_{eff} = \frac{L}{n(L+K-1)} < R_c$$

Encoder representation

- Vector representation:
  - We define n binary vector with K elements (one vector for each modulo-2 adder). The i:th element in each vector, is “1” if the i:th stage in the shift register is connected to the corresponding modulo-2 adder, and “0” otherwise.
    - Example:

$$\mathbf{g}_1 = (111)$$
$$\mathbf{g}_2 = (101)$$



Encoder representation – cont'd

- Impulse response representaiton:
  - The response of encoder to a single "one" bit that goes through it.
    - Example:

Register contents	Branch word u_1	u_2
100	1	1
010	1	0
001	1	1

Input sequence : 1 0 0

Output sequence : 11 10 11

Input **m**	Output				
1	11	10	11		
0		00	00	00	
1			11	10	11
Modulo-2 sum:	11	10	00	10	11

Encoder representation – cont'd

- Polynomial representation:
  - We define n generator polynomials, one for each modulo-2 adder. Each polynomial is of degree *K-1* or less and describes the connection of the shift registers to the corresponding modulo-2 adder.
    - Example:

$$\mathbf{g}_1(D) = g_0^{(1)} + g_1^{(1)}.D + g_2^{(1)}.D^2 = 1 + D + D^2$$

$$\mathbf{g}_2(D) = g_0^{(2)} + g_1^{(2)}.D + g_2^{(2)}.D^2 = 1 + D^2$$

The output sequence is found as follows:

$$\mathbf{U}(D) = \mathbf{m}(D)\mathbf{g}_1(D) \text{ interlaced with } \mathbf{m}(D)\mathbf{g}_2(D)$$

Encoder representation – cont'd

In more details:

$$\mathbf{m}(D)\mathbf{g}_1(D) = (1+D^2)(1+D+D^2) = 1+D+D^3+D^4$$

$$\mathbf{m}(D)\mathbf{g}_2(D) = (1+D^2)(1+D^2) = 1+D^4$$

$$\mathbf{m}(D)\mathbf{g}_1(D) = 1+D+0.D^2+D^3+D^4$$

$$\mathbf{m}(D)\mathbf{g}_2(D) = 1+0.D+0.D^2+0.D^3+D^4$$

$$\mathbf{U}(D) = (1,1)+(1,0)D+(0,0)D^2+(1,0)D^3+(1,1)D^4$$

$$\mathbf{U} = 11 \quad 10 \quad 00 \quad 10 \quad 11$$

The convolutional codes generated by the encoders of Figure 10.13 are *nonsystematic codes*. Unlike block coding, the use of nonsystematic codes is ordinarily preferred over systematic codes in convolutional coding.

Each path connecting the output to the input of a convolutional encoder may be characterized in terms of its *impulse response*, defined as the response of that path to a symbol 1 applied to its input, with each flip-flop in the encoder set initially in the zero state. Equivalently, we may characterize each path in terms of a *generator polynomial*, defined as the *unit-delay transform* of the impulse response. To be specific, let the *generator sequence* $(g_0^{(i)}, g_1^{(i)}, g_2^{(i)}, \ldots, g_M^{(i)})$ denote the impulse response of the ith path, where the coefficients $g_0^{(i)}, g_1^{(i)}, g_2^{(i)}, \ldots, g_M^{(i)}$ equal 0 or 1. Correspondingly, the *generator polynomial* of the ith path is defined by

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)}D + g_2^{(i)}D^2 + \cdots + g_M^{(i)}D^M \tag{10.55}$$

where D denotes the unit-delay variable. The complete convolutional encoder is described by the set of generator polynomials $\{g^{(1)}(D), g^{(2)}(D), \ldots, g^{(n)}(D)\}$. Traditionally, different variables are used for the description of convolutional and cyclic codes, with D being commonly used for convolutional codes and X for cyclic codes.

EXAMPLE 10.5

Consider the convolutional encoder of Figure 10.13*a*, which has two paths numbered 1 and 2 for convenience of reference. The impulse response of path 1 (i.e., upper path) is (1, 1, 1). Hence, the cortesponding generator polynomial is given by

$$g^{(1)}(D) = 1 + D + D^2$$

The impulse response of path 2 (i.e., lower path) is (1, 0, 1). Hence, the corresponding generator polynomial is given by

$$g^{(2)}(D) = 1 + D^2$$

For the message sequence (10011), say, we have the polynomial representation

$$m(D) = 1 + D^3 + D^4$$

As with Fourier transformation, convolution in the time domain is transformed into multiplication in the D-domain. Hence, the output polynomial of path 1 is given by

$$\begin{aligned} c^{(1)}(D) &= g^{(1)}(D)m(D) \\ &= (1 + D + D^2)(1 + D^3 + D^4) \\ &= 1 + D + D^2 + D^3 + D^6 \end{aligned}$$

From this we immediately deduce that the output sequence of path 1 is (1111001). Similarly, the output polynomial of path 2 is given by

$$\begin{aligned} c^{(2)}(D) &= g^{(2)}(D)m(D) \\ &= (1 + D^2)(1 + D^3 + D^4) \\ &= 1 + D^2 + D^3 + D^4 + D^5 + D^6 \end{aligned}$$

The output sequence of path 2 is therefore (1011111). Finally, multiplexing the two output sequences of paths 1 and 2, we get the encoded sequence

$$\mathbf{c} = (11, 10, 11, 11, 01, 01, 11)$$

Note that the message sequence of length $L = 5$ bits produces an encoded sequence of length $n(L + K - 1) = 14$ bits. Note also that for the shift register to be restored to its zero initial state, a terminating sequence of $K - 1 = 2$ zeros is appended to the last input bit of the message sequence. The terminating sequence of $K - 1$ zeros is called the *tail of the message*.

Part II

State diagram

- A finite-state machine only encounters a finite number of states.
- State of a machine: the smallest amount of information that, together with a current input to the machine, can predict the output of the machine.
- In a Convolutional encoder, the state is represented by the content of the memory.
- Hence, there are 2^{K-1} states.
- A state diagram is a way to represent the encoder.
- A state diagram contains all the states and all possible transitions between them.
- Only two transitions initiating from a state
- Only two transitions ending up in a state

The example encoder has two bits of memory, so there are four possible states.

Let's give the left-hand flip-flop a binary weight of 2^1, and the right-hand flip-flop a binary weight of 2^0.

Initially, the encoder is in the all-zeroes state.

If the first input bit is a zero, the encoder stays in the all zeroes state at the next clock edge.

But if the input bit is a one, the encoder transitions to the 10_2 state at the next clock edge.

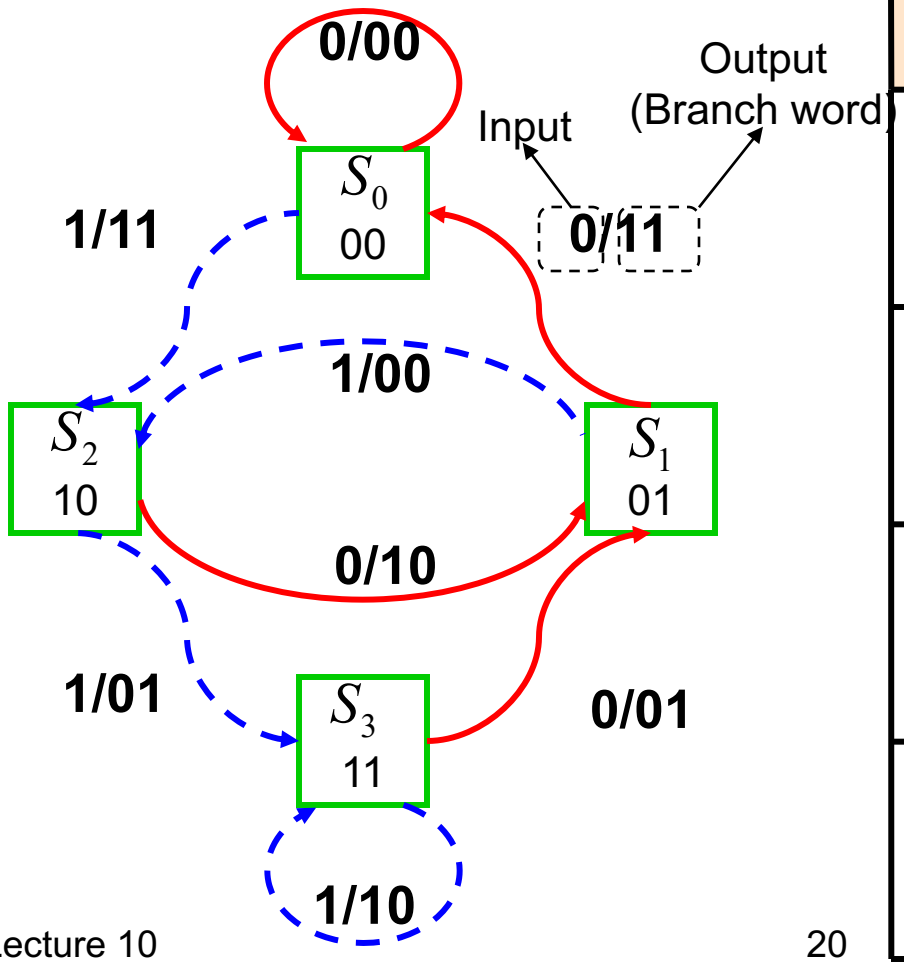
Then, if the next input bit is zero, the encoder transitions to the 01_2 state, otherwise, it transitions to the 11_2 state.

Current state	input	Next state	output
00	0	00	00
	1	10	11
01	0	00	11
	1	10	00
10	0	01	10
	1	11	01
11	0	01	01
	1	11	10

	Next State, if	
Current State	Input = 0:	Input = 1:
00	00	10
01	00	10
10	01	11
11	01	11

	Output Symbols, if	
Current State	Input = 0:	Input = 1:
00	00	11
01	11	00
10	10	01
11	01	10

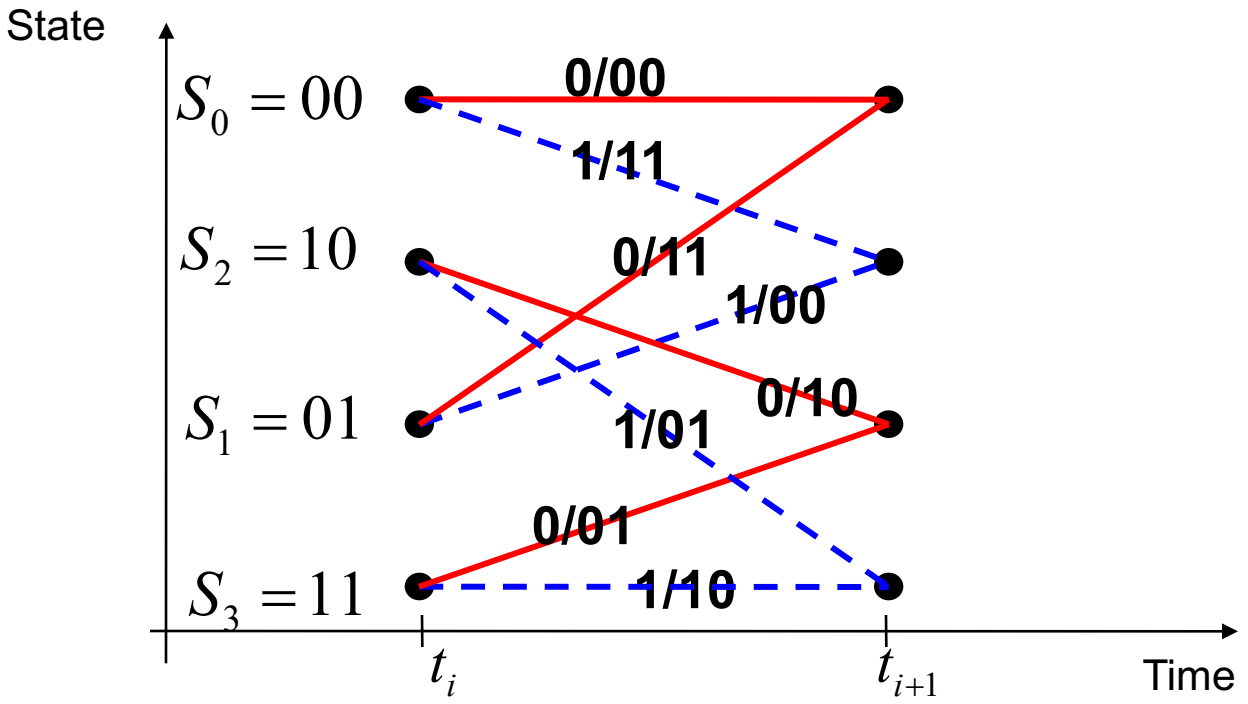
State diagram – cont'd



Current state	input	Next state	output
S_0 00	0	S_0	00
	1	S_2	11
S_1 01	0	S_0	11
	1	S_2	00
S_2 10	0	S_1	10
	1	S_3	01
S_3 11	0	S_1	01
	1	S_3	10

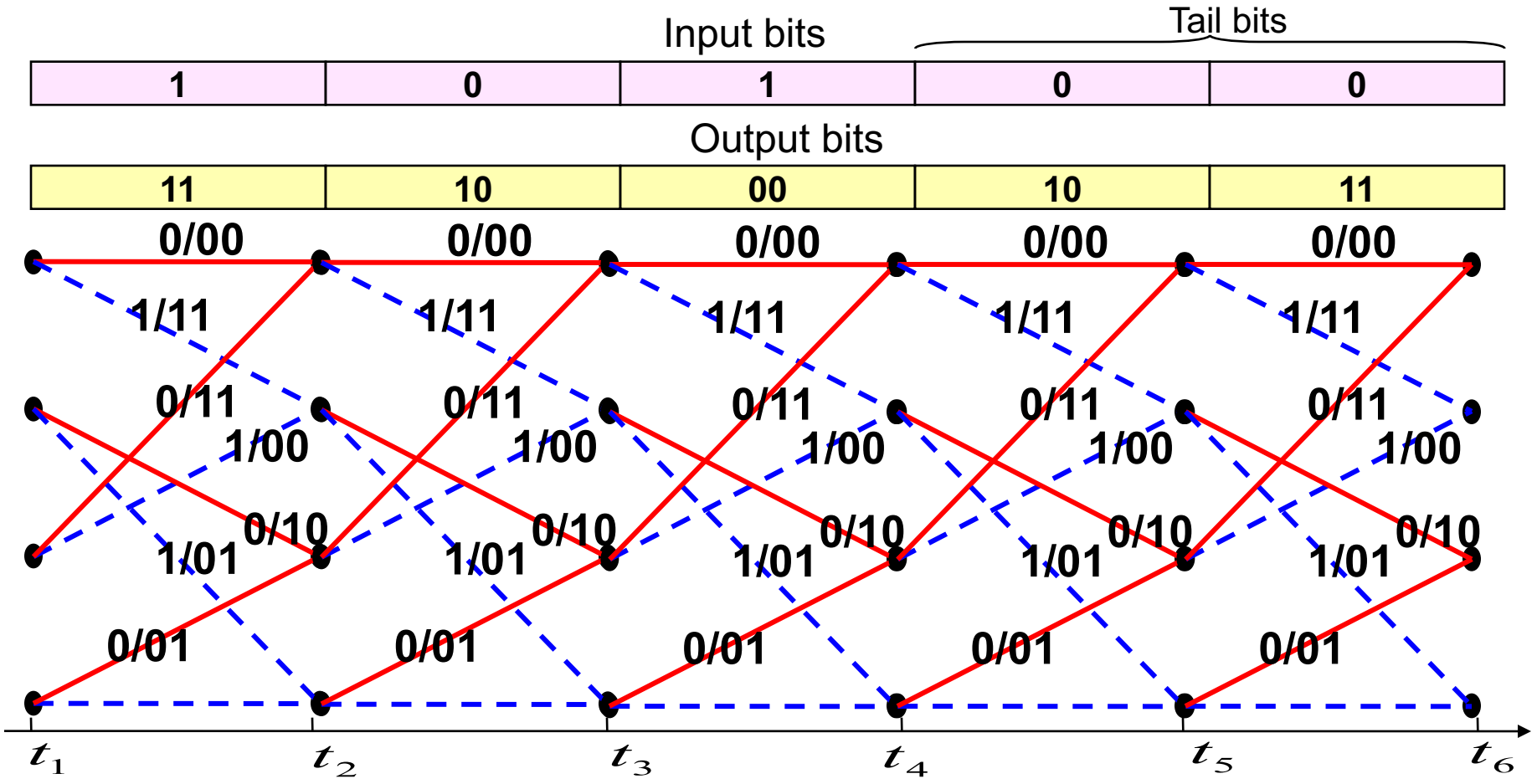
Trellis – cont'd

- Trellis diagram is an extension of the state diagram that shows the passage of time.
  - Example of a section of trellis for the rate ½ code

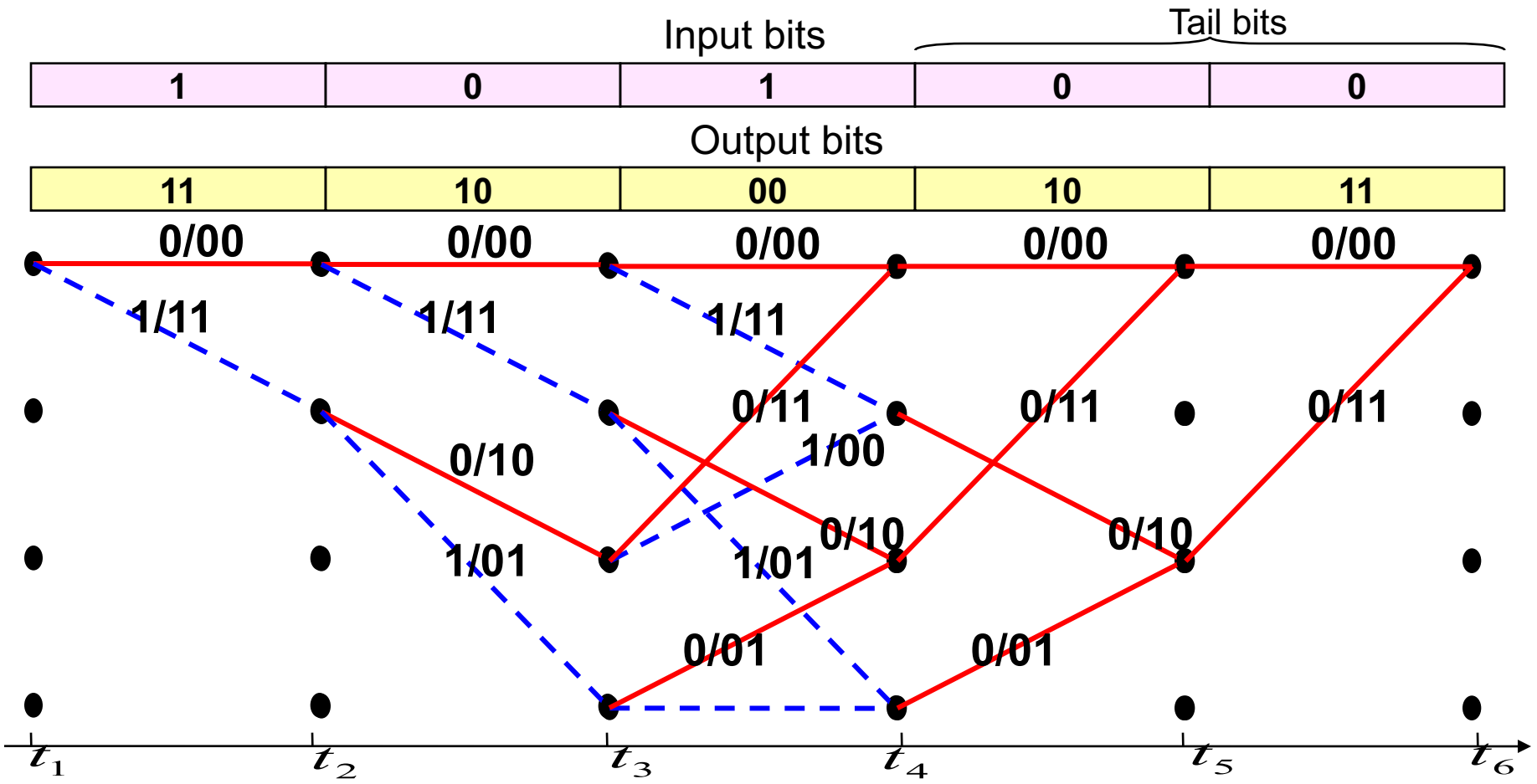


Trellis –cont’ d

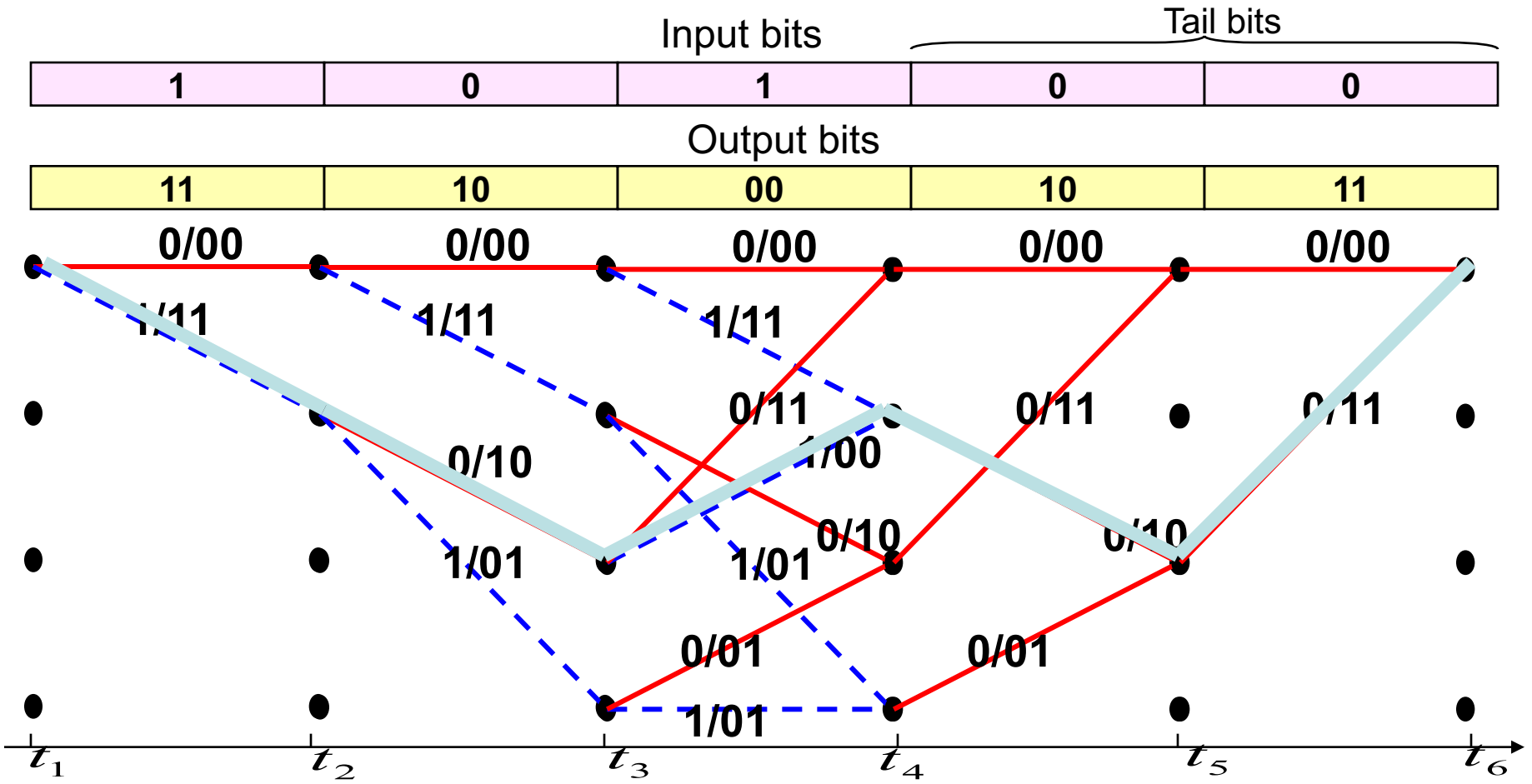
- A trellis diagram for the example code



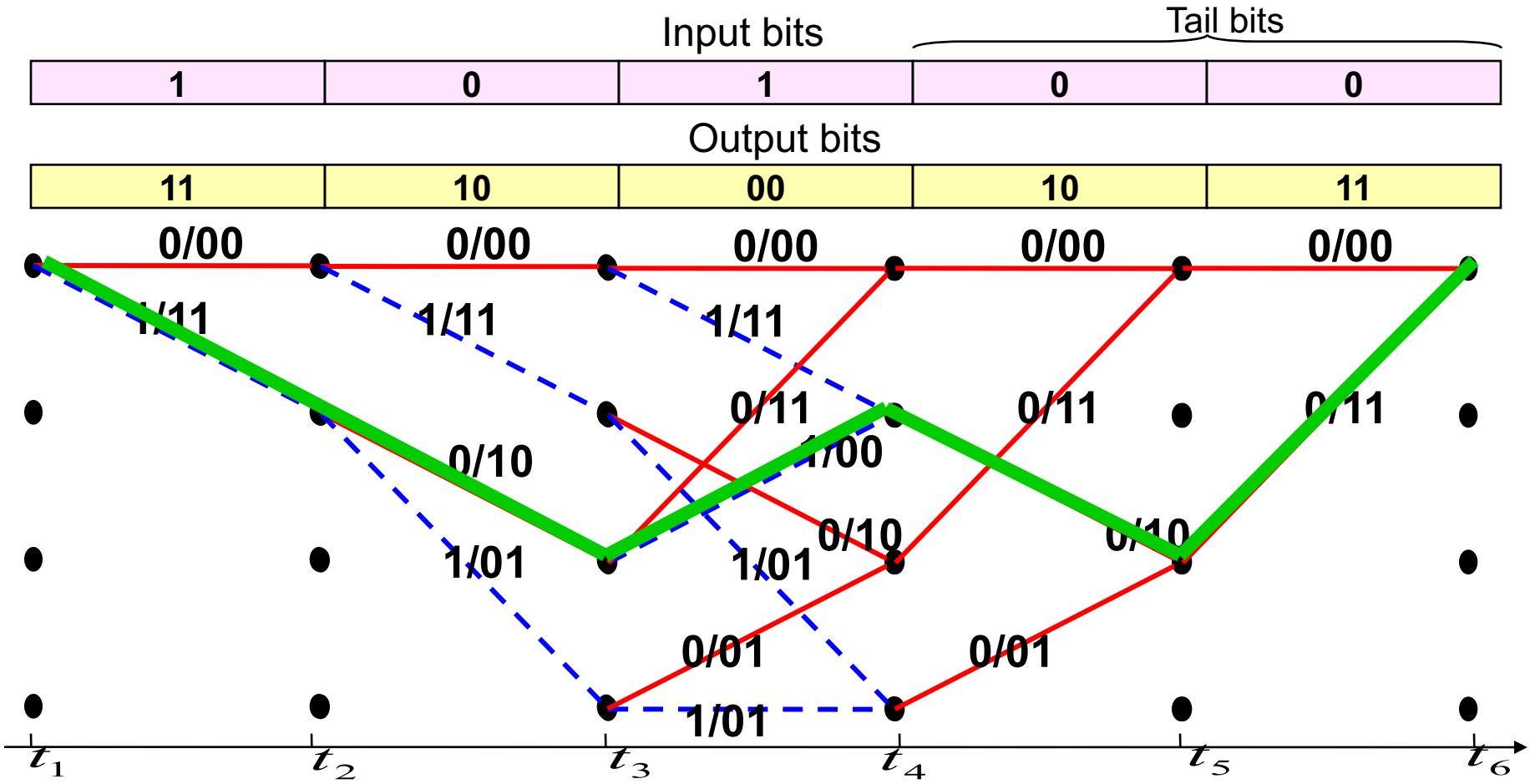
Trellis – cont'd



Trellis of an example ½ Conv. code



Trellis of an example ½ Conv. code



The Viterbi algorithm

- The Viterbi algorithm performs Maximum likelihood decoding.
- It finds a path through the trellis with the largest metric (maximum correlation or minimum distance).
  - It processes the demodulator outputs in an iterative manner.
  - At each step in the trellis, it compares the metric of all paths entering each state, and keeps only the path with the smallest metric, called the survivor, together with its metric.
  - It proceeds in the trellis by eliminating the least likely paths.
- It reduces the decoding complexity to $L2^{K-1}$!

The Viterbi algorithm - cont'd

- Viterbi algorithm:

A. Do the following set up:

- For a data block of L bits, form the trellis. The trellis has $L+K-1$ sections or levels and starts at time t_1 and ends up at time t_{L+K}.
- Label all the branches in the trellis with their corresponding branch metric.
- For each state in the trellis at the time t_i which is denoted by $S(t_i) \in \{0,1,...,2^{K-1}\}$, define a parameter $\Gamma(S(t_i), t_i)$

B. Then, do the following:

The Viterbi algorithm - cont'd

1. Set $\Gamma(0, t_1) = 0$ and $i = 2$.
2. At time t_i, compute the partial path metrics for all the paths entering each state.
3. Set $\Gamma(S(t_i), t_i)$ equal to the best partial path metric entering each state at time t_i

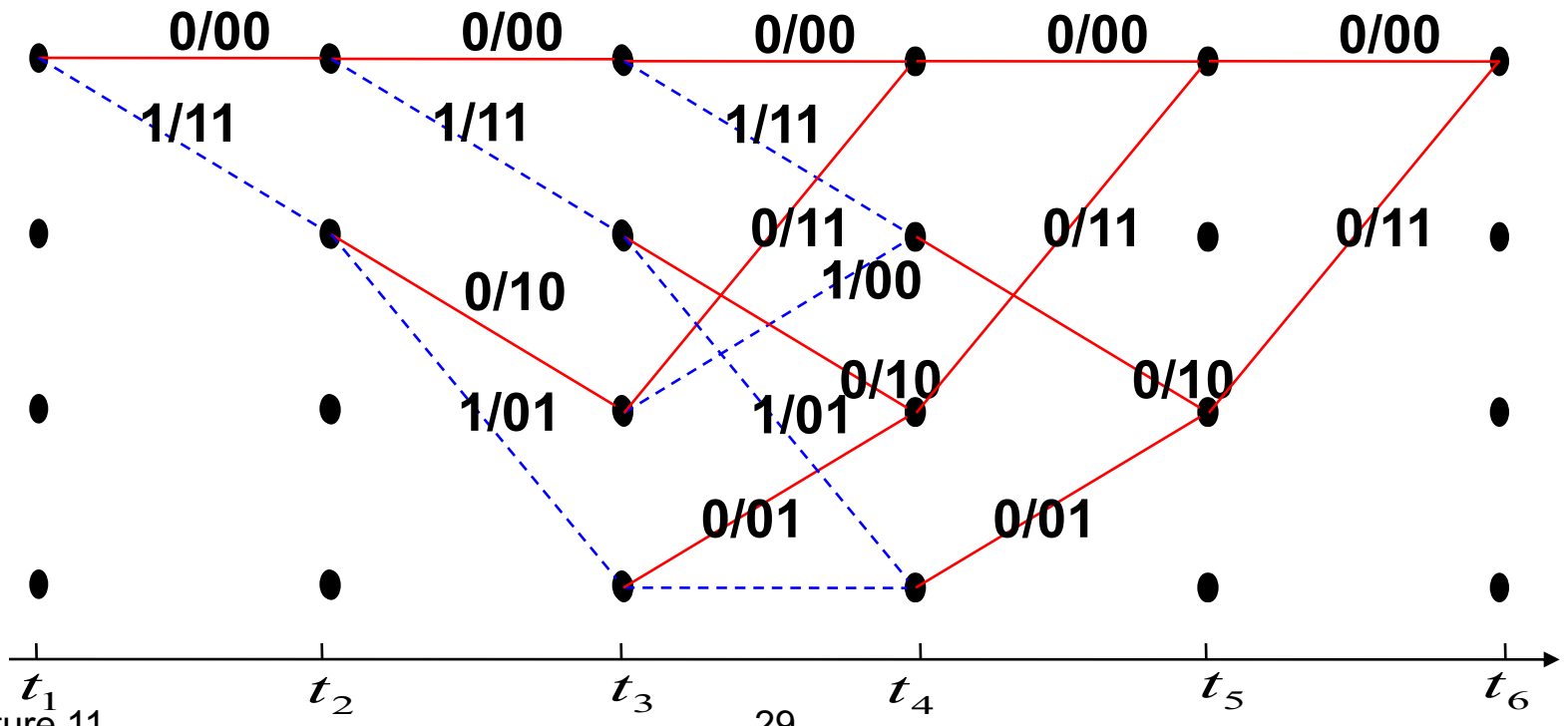
   Keep the survivor path and delete the dead paths from the trellis.

1. If $i < L + K$, increase i by 1 and return to step 2.

A. Start at state zero at time t_{L+K}. Follow the surviving branches backwards through the trellis. The path found is unique and corresponds to the ML codeword.

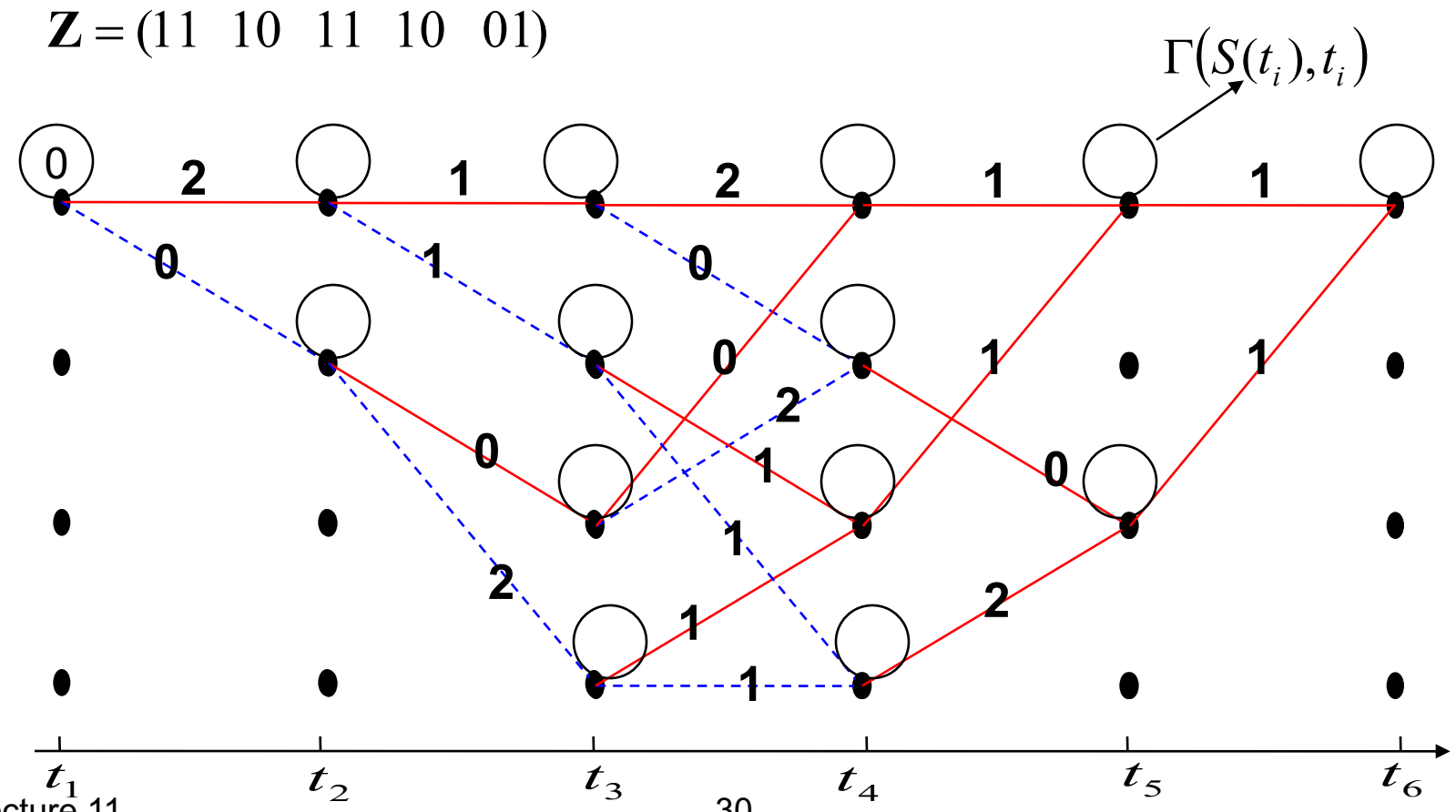
Example of Hard decision Viterbi decoding

$$\mathbf{m} = (101)$$
$$\mathbf{U} = (11 \;\; 10 \;\; 00 \;\; 10 \;\; 11)$$
$$\mathbf{Z} = (11 \;\; 10 \;\; 11 \;\; 10 \;\; 01)$$



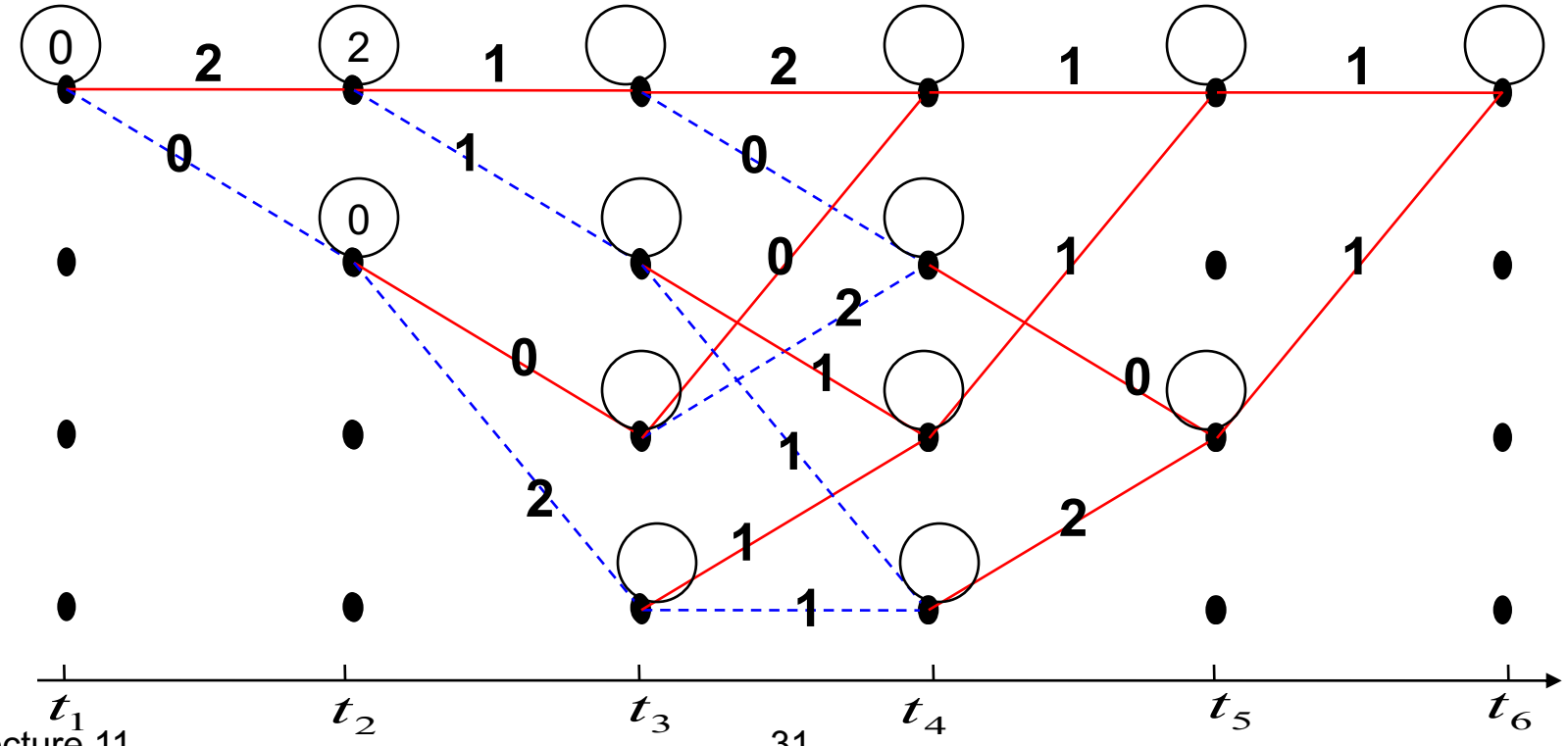
Example of Hard decision Viterbi decoding-cont'd

- Label all the branches with the branch metric (Hamming distance)



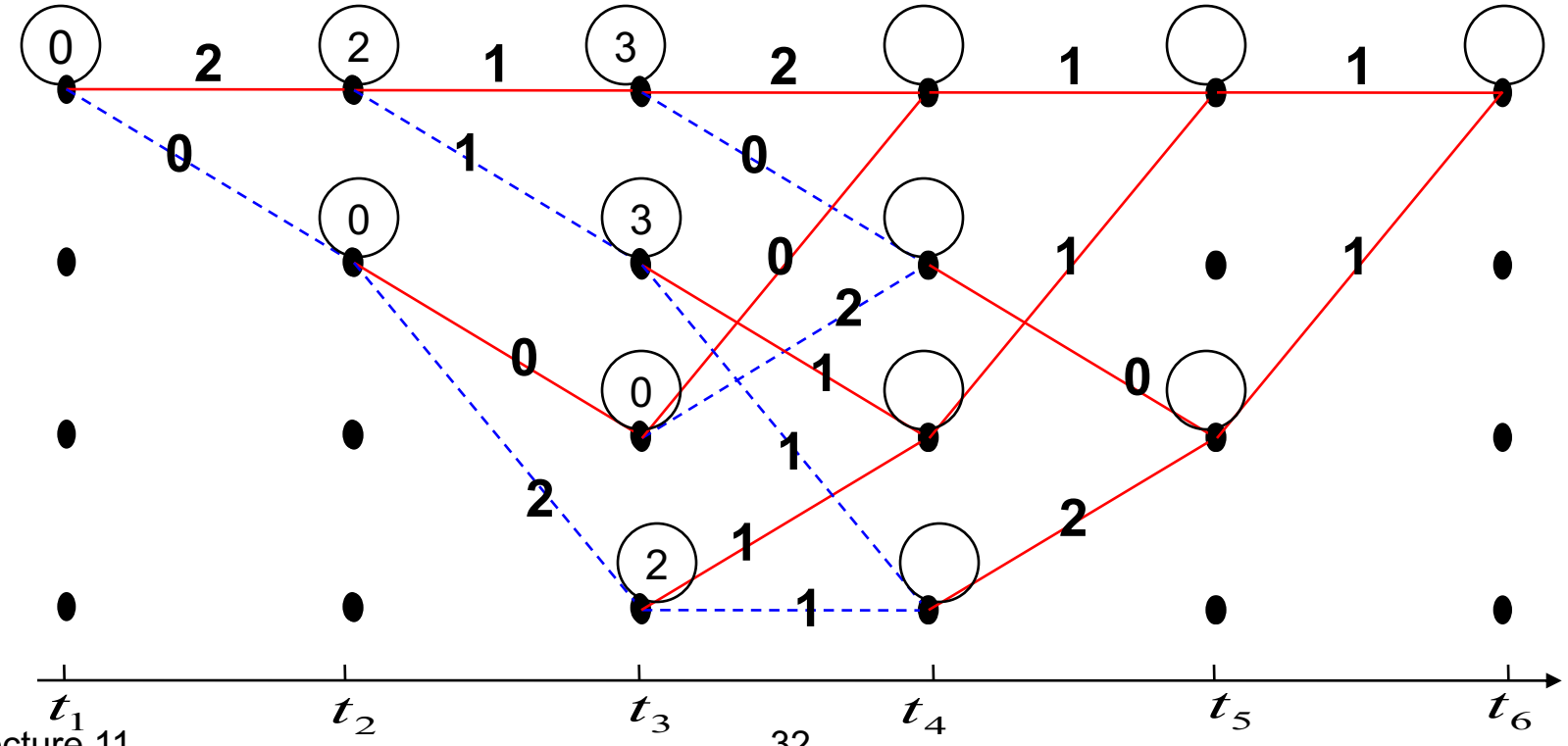
Example of Hard decision Viterbi decoding-cont'd

- i=2



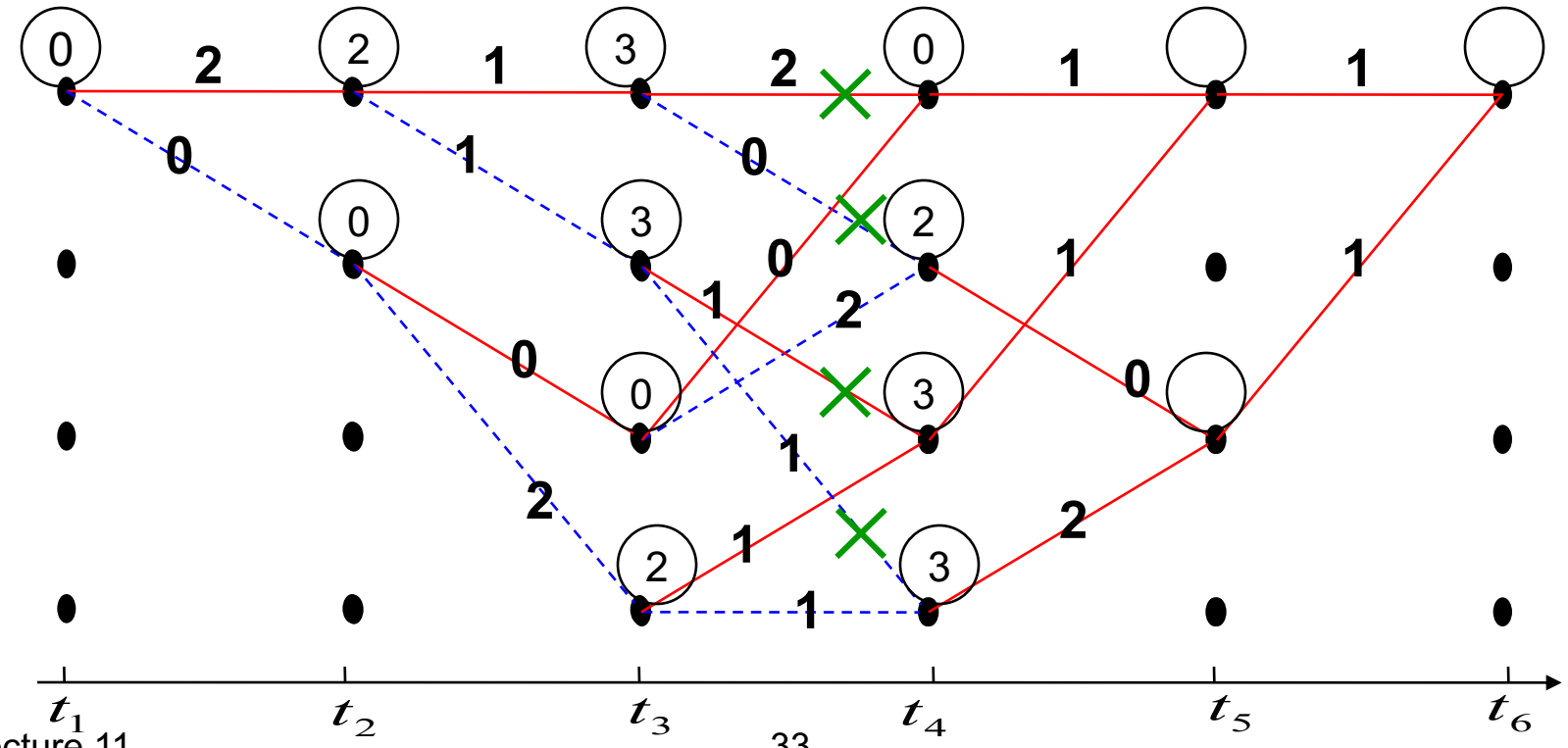
Example of Hard decision Viterbi decoding-cont'd

- i=3



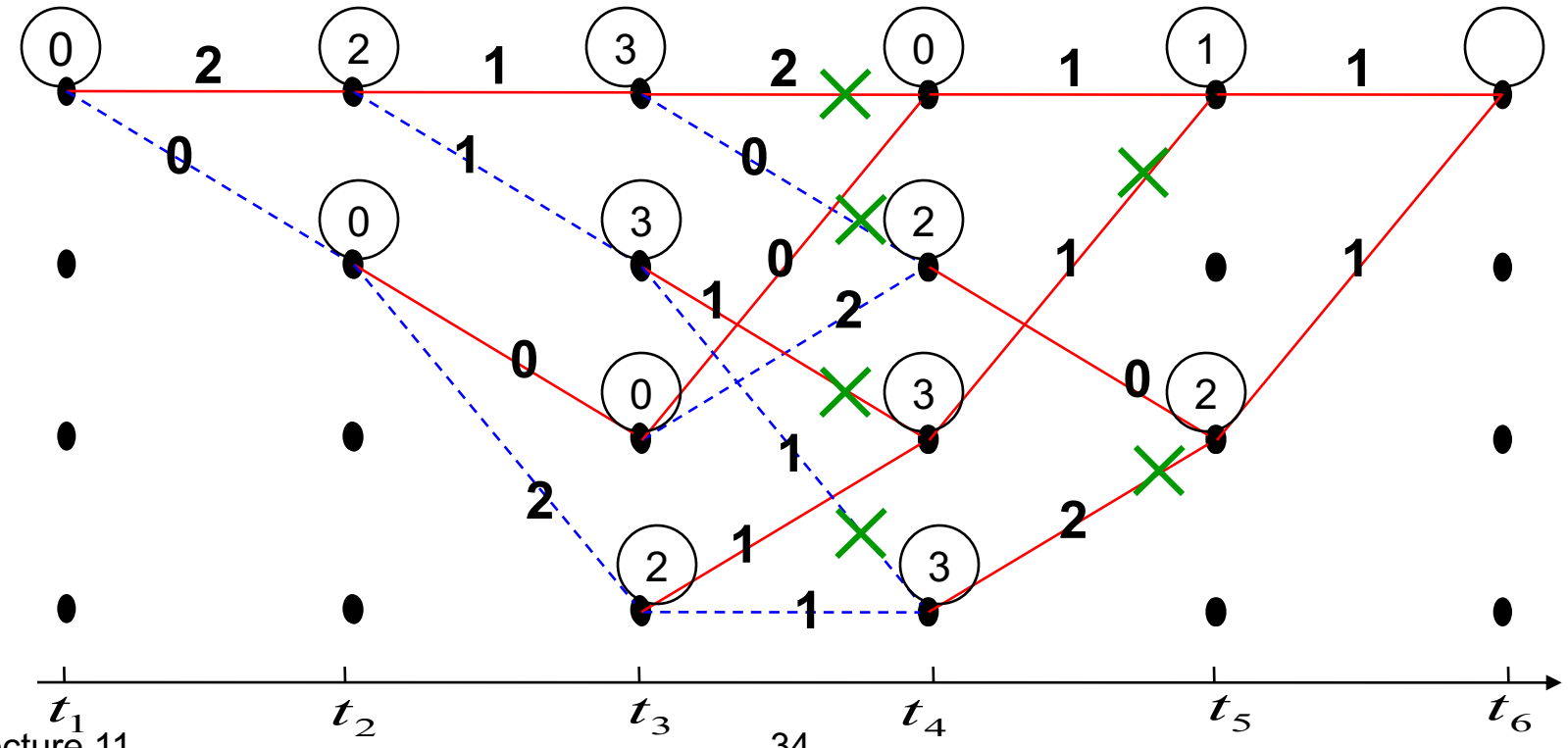
Example of Hard decision Viterbi decoding-cont'd

- i=4



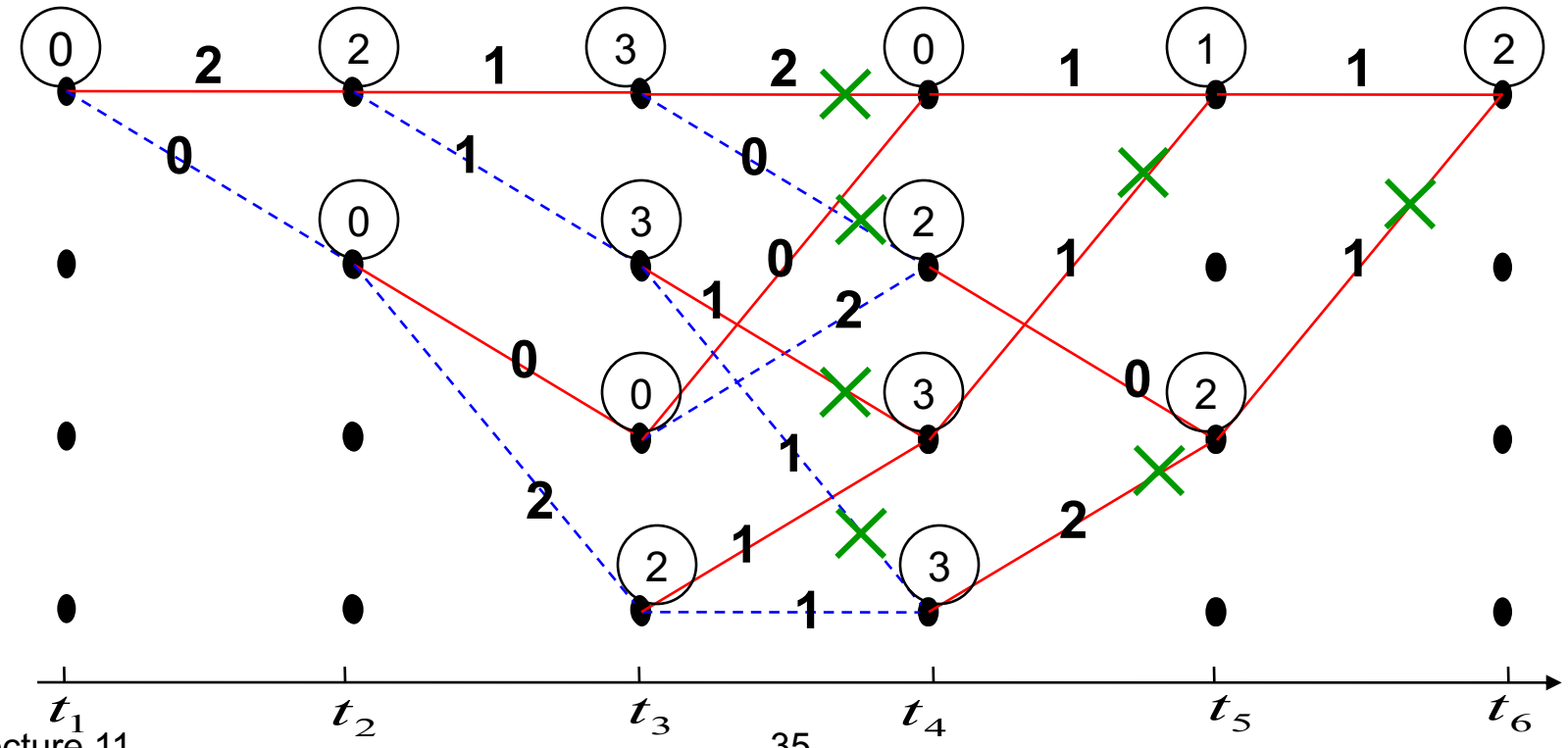
Example of Hard decision Viterbi decoding-cont'd

- i=5



Example of Hard decision Viterbi decoding-cont'd

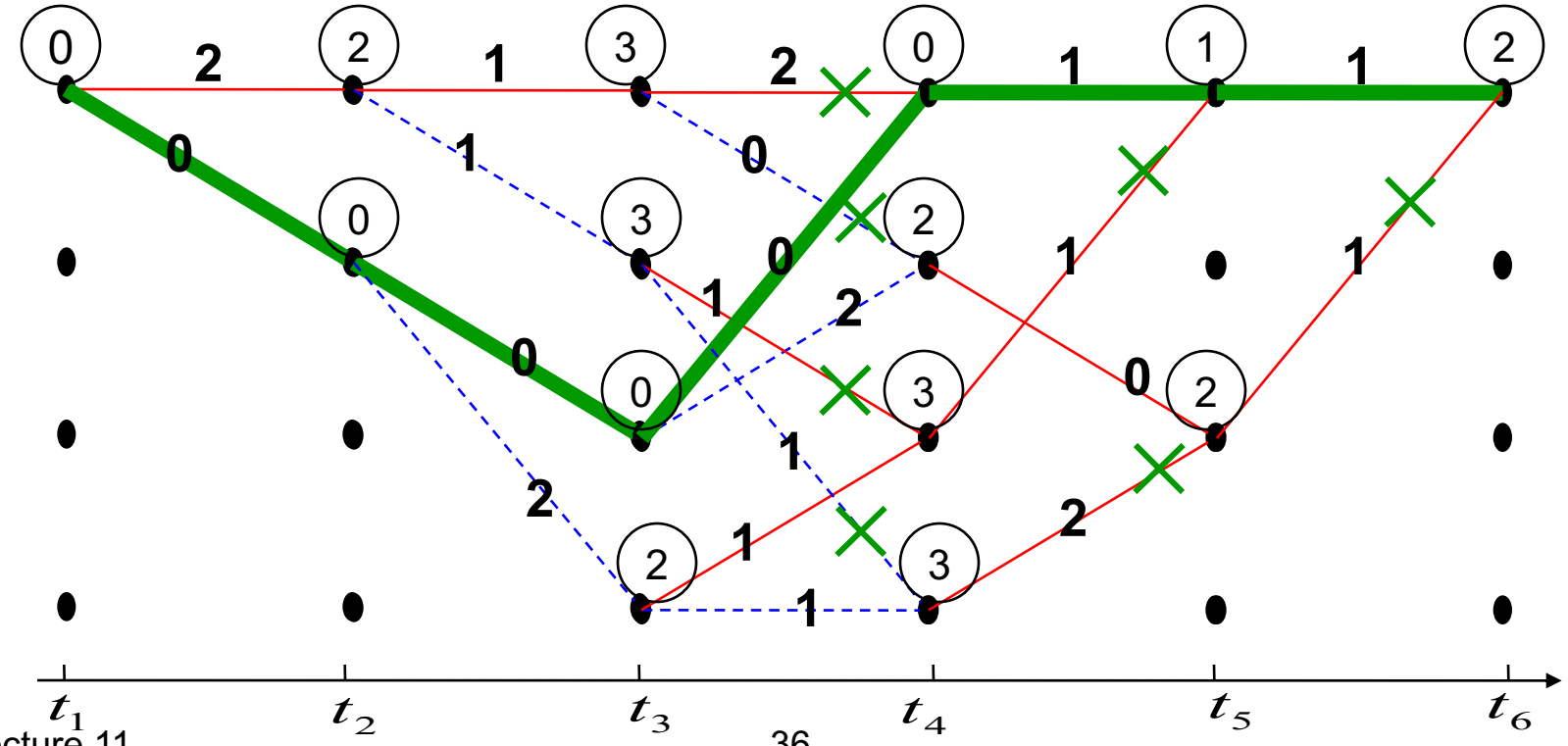
- i=6



Example of Hard decision Viterbi decoding-cont'd

- Trace back and then:

$$\hat{\mathbf{m}} = (100)$$
$$\hat{\mathbf{U}} = (11 \;\; 10 \;\; 11 \;\; 00 \;\; 00)$$



▪ THE VITERBI ALGORITHM[9]

The equivalence between maximum likelihood decoding and minimum distance decoding for a binary symmetric channel implies that we may decode a convolutional code by choosing a path in the code tree whose coded sequence differs from the received sequence in the fewest number of places. Since a code tree is equivalent to a trellis, we may equally limit our choice to the possible paths in the trellis representation of the code. The reason for preferring the trellis over the tree is that the number of nodes at any level of the trellis

does not continue to grow as the number of incoming message bits increases; rather, it remains constant at 2^{K-1}, where K is the constraint length of the code.

Consider, for example, the trellis diagram of Figure 10.15 for a convolutional code with rate $r = 1/2$ and constraint length $K = 3$. We observe that at level $j = 3$, there are two paths entering any of the four nodes in the trellis. Moreover, these two paths will be identical onward from that point. Clearly, a minimum distance decoder may make a decision at that point as to which of those two paths to retain, without any loss of performance. A similar decision may be made at level $j = 4$, and so on. This sequence of decisions is exactly what the *Viterbi algorithm* does as it walks through the trellis. The algorithm operates by computing a *metric* or discrepancy for every possible path in the trellis. The metric for a particular path is defined as the Hamming distance between the coded sequence represented by that path and the received sequence. Thus, for each node (state) in the trellis of Figure 10.15 the algorithm compares the two paths entering the node. The path with the lower metric is retained, and the other path is discarded. This computation is repeated for every level j of the trellis in the range $M \leq j \leq L$, where $M = K - 1$ is the encoder's memory and L is the length of the incoming message sequence. The paths that are retained by the algorithm are called *survivor* or *active paths*. For a convolutional code of constraint length $K = 3$, for example, no more than $2^{K-1} = 4$ survivor paths and their metrics will ever be stored. This list of 2^{K-1} paths is always guaranteed to contain the maximum-likelihood choice.

A difficulty that may arise in the application of the Viterbi algorithm is the possibility that when the paths entering a state are compared, their metrics are found to be identical. In such a situation, we make the choice by flipping a fair coin (i.e., simply make a guess).

In summary, the Viterbi algorithm is a maximum-likelihood decoder, which is optimum for an AWGN channel. It proceeds in a step-by-step fashion as follows:

Initialization

Label the left-most state of the trellis (i.e., the all-zero state at level 0) as 0, since there is no discrepancy at this point in the computation.

Computation step j + 1

Let $j = 0, 1, 2, \ldots$, and suppose that at the previous step j we have done two things:

- All survivor paths are identified.
- The survivor path and its metric for each state of the trellis are stored.

Then, at level (clock time) $j + 1$, compute the metric for all the paths entering each state of the trellis by adding the metric of the incoming branches to the metric of the connecting survivor path from level j. Hence, for each state, identify the path with the lowest metric as the survivor of step $j + 1$, thereby updating the computation.

Final Step

Continue the computation until the algorithm completes its forward search through the trellis and therefore reaches the termination node (i.e., all-zero state), at which time it makes a decision on the maximum likelihood path. Then, like a block decoder, the sequence of symbols associated with that path is released to the destination as the decoded version of the received sequence. In this sense, it is therefore more correct to refer to the Viterbi algorithm as a *maximum likelihood sequence estimator.*

However, when the received sequence is very long (near infinite), the storage requirement of the Viterbi algorithm becomes too high, and some compromises must be made.

The approach usually taken is to "truncate" the path memory of the decoder as described here. A *decoding window* of length ℓ is specified, and the algorithm operates on a corresponding frame of the received sequence, always stopping after ℓ steps. A decision is then made on the "best" path and the symbol associated with the first branch on that path is released to the user. The symbol associated with the last branch of the path is dropped. Next, the decoding window is moved forward one time interval, and a decision on the next code frame is made, and so on. The decoding decisions made in this way are no longer truly maximum likelihood, but they can be made almost as good provided that the decoding window is long enough. Experience and analysis have shown that satisfactory results are obtained if the decoding window length ℓ is on the order of 5 times the constraint length K of the convolutional code or more.

▶ EXAMPLE 10.6 Correct Decoding of Received All-Zero Sequence

Suppose that the encoder of Figure 10.13*a* generates an all-zero sequence that is sent over a binary symmetric channel, and that the received sequence is (0100010000 . . .). There are two errors in the received sequence due to noise in the channel: one in the second bit and the other in the sixth bit. We wish to show that this double-error pattern is correctable through the application of the Viterbi decoding algorithm.

In Figure 10.17, we show the results of applying the algorithm for level $j = 1, 2, 3, 4, 5$. We see that for $j = 2$ there are (for the first time) four paths, one for each of the four states of the encoder. The figure also includes the metric of each path for each level in the computation.

In the left side of Figure 10.17, for $j = 3$ we show the paths entering each of the states, together with their individual metrics. In the right side of the figure, we show the four survivors that result from application of the algorithm for level $j = 3, 4, 5$.

Examining the four survivors in Figure 10.17 for $j = 5$, we see that the all-zero path has the smallest metric and will remain the path of smallest metric from this point forward. This clearly shows that the all-zero sequence is the maximum likelihood choice of the Viterbi decoding algorithm, which agrees exactly with the transmitted sequence. ◀

▶ EXAMPLE 10.7 Incorrect Decoding of Received All-Zero Sequence

Suppose next that the received sequence is (1100010000 . . .), which contains three errors compared to the transmitted all-zero sequence.

In Figure 10.18, we show the results of applying the Viterbi decoding algorithm for $j = 1, 2, 3, 4$. We see that in this example the correct path has been eliminated by level $j = 3$. Clearly, a triple-error pattern is uncorrectable by the Viterbi algorithm when applied to a convolutional code of rate 1/2 and constraint length $K = 3$. The exception to this rule is a triple-error pattern spread over a time span longer than one constraint length, in which case it is very likely to be correctable. ◀

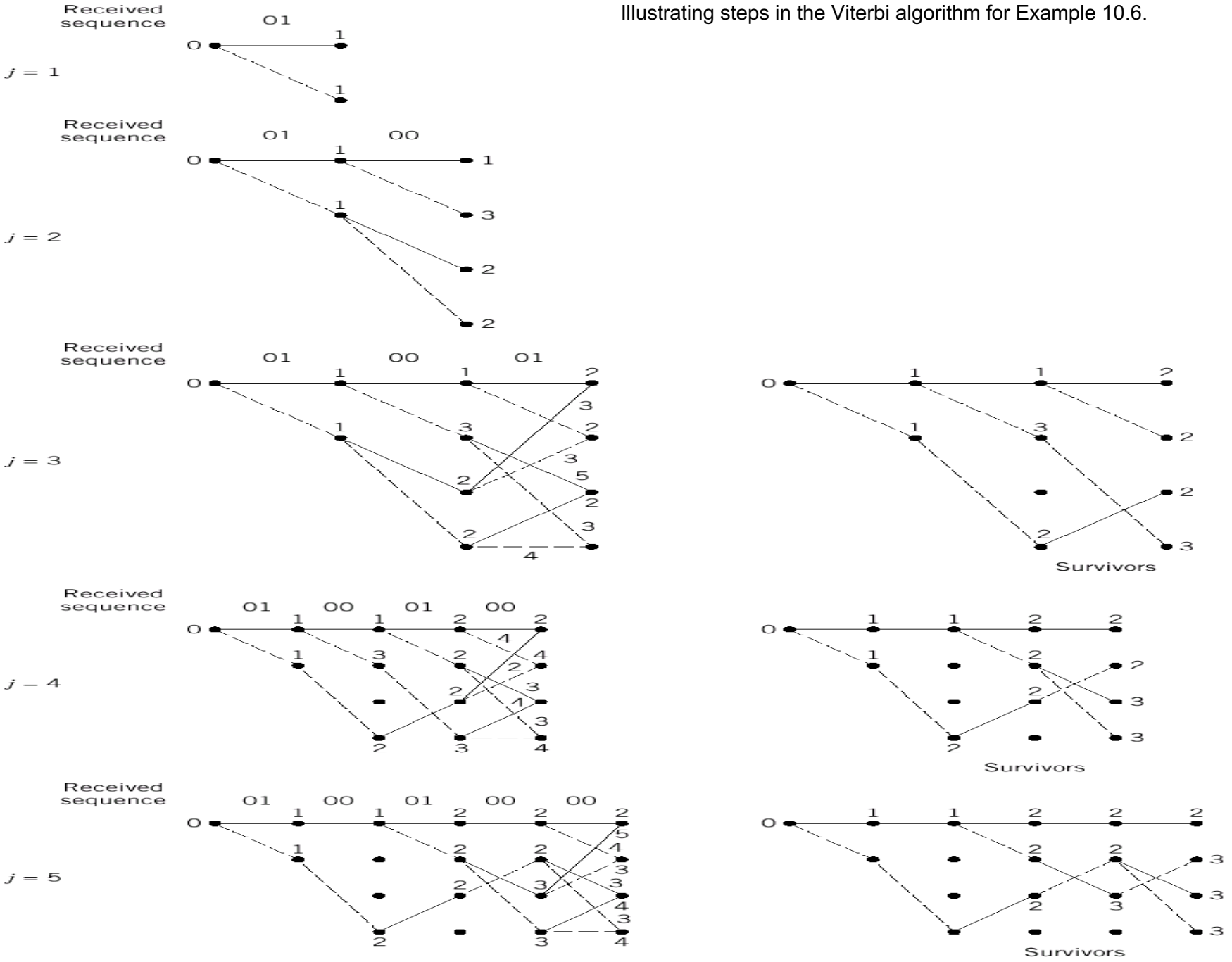
Free Distance of a Convolutional Code

The performance of a convolutional code depends not only on the decoding algorithm used but also on the distance properties of the code. In this context, the most important single measure of a convolutional code's ability to combat channel noise is the free distance, denoted by d_{free}. The *free distance* of a convolutional code is defined as *the minimum Hamming distance between any two code words in the code.* A convolutional code with free distance d_{free} can correct t errors if and only if d_{free} is greater than $2t$.

The free distance can be obtained quite simply from the state diagram of the convolutional encoder. Consider, for example, Figure 10.16*b*, which shows the state diagram

Figure 10.17

Illustrating steps in the Viterbi algorithm for Example 10.6.



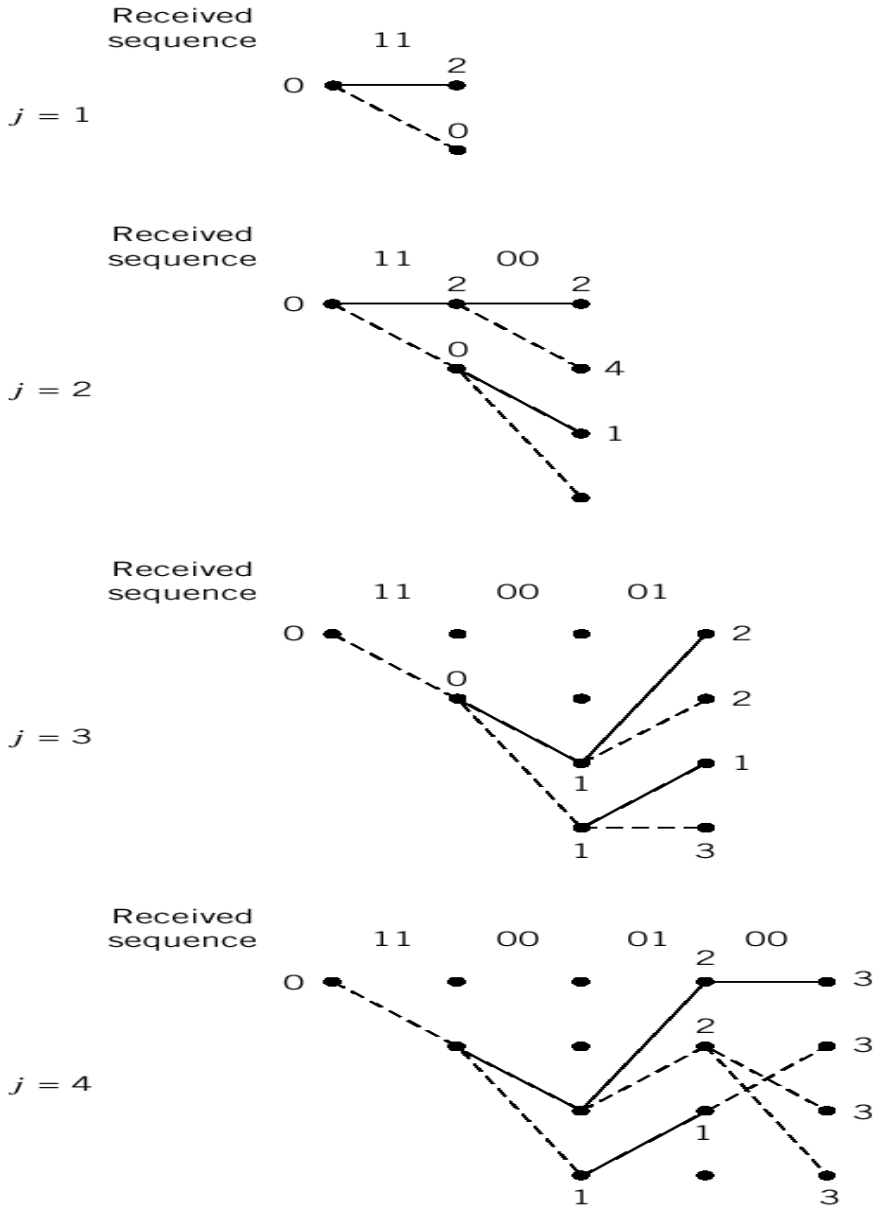


Figure 10.18

Illustrating breakdown of the Viterbi algorithm in Example 10.7.

Lecture sides adopted from:

[1] Anders Ahlén Professor in Signal Processing, Uppsala University, Lecture notes
http://www.signal.uu.se/Courses/CourseDirs/ModDemKod/2009/body.html

[2] Symon Haykin communication systems v4