EEE8099 – Information Theory & Coding
EEE8104 – Digital Communications

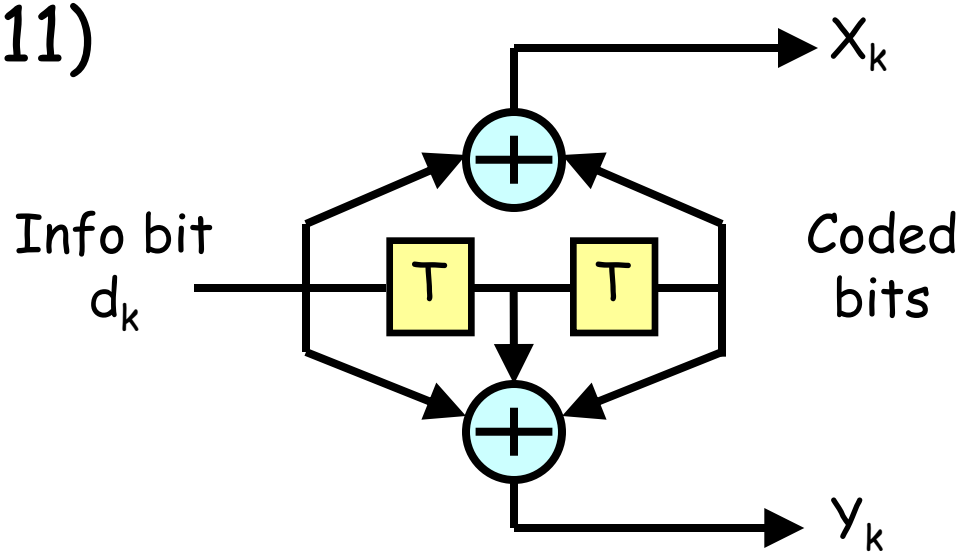
S. Le Goff
School of Engineering @ Newcastle University

Part 6
Practical Error-Correcting Codes
1948 - 1993
Convolutional Codes

Convolutional codes (1955)

1955: Peter Elias introduces the concept of convolutional codes (CCs) in which the stream of info bits is encoded by a finite-state machine.

Ex: $R_c = \frac{1}{2}$, constraint length K = 3, generator polynomials 5 (connections = 101) and 7 (connections = 111)

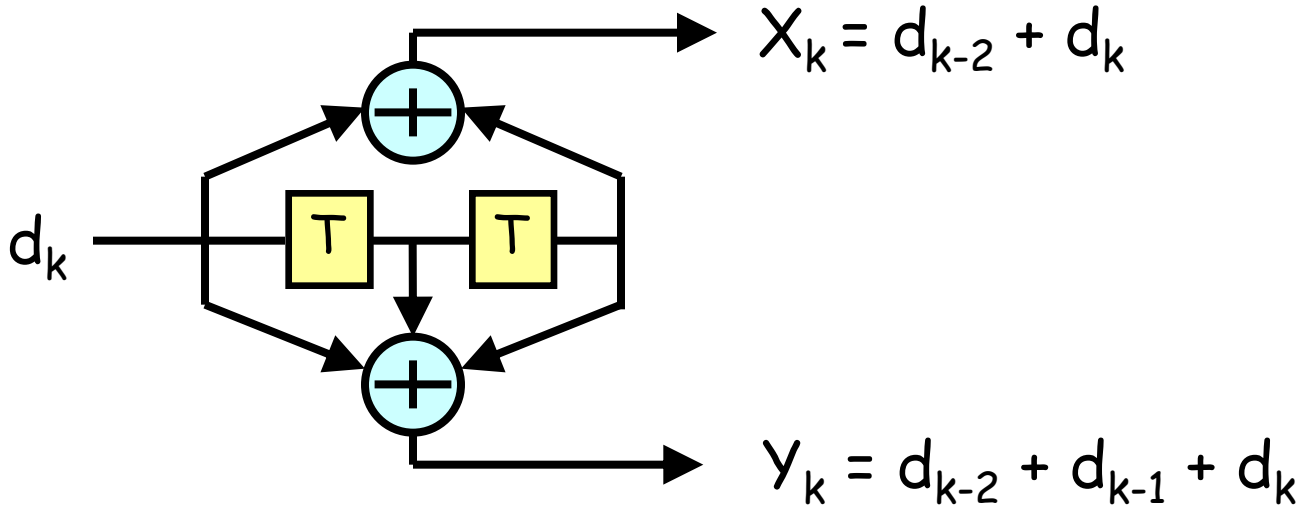


Peter Elias (1923 - 2001)

Convolutional codes

The constraint length K is the number of info bits that are taken into account in the computation of the coded bits. We have K = 3 for our (5, 7) encoder.

Two coded bits: $X_k = d_{k-2} + d_k$ and $Y_k = d_{k-2} + d_{k-1} + d_k$



Convolutional codes

The values of d_{k-2} and d_{k-1} define the current state of the encoder.

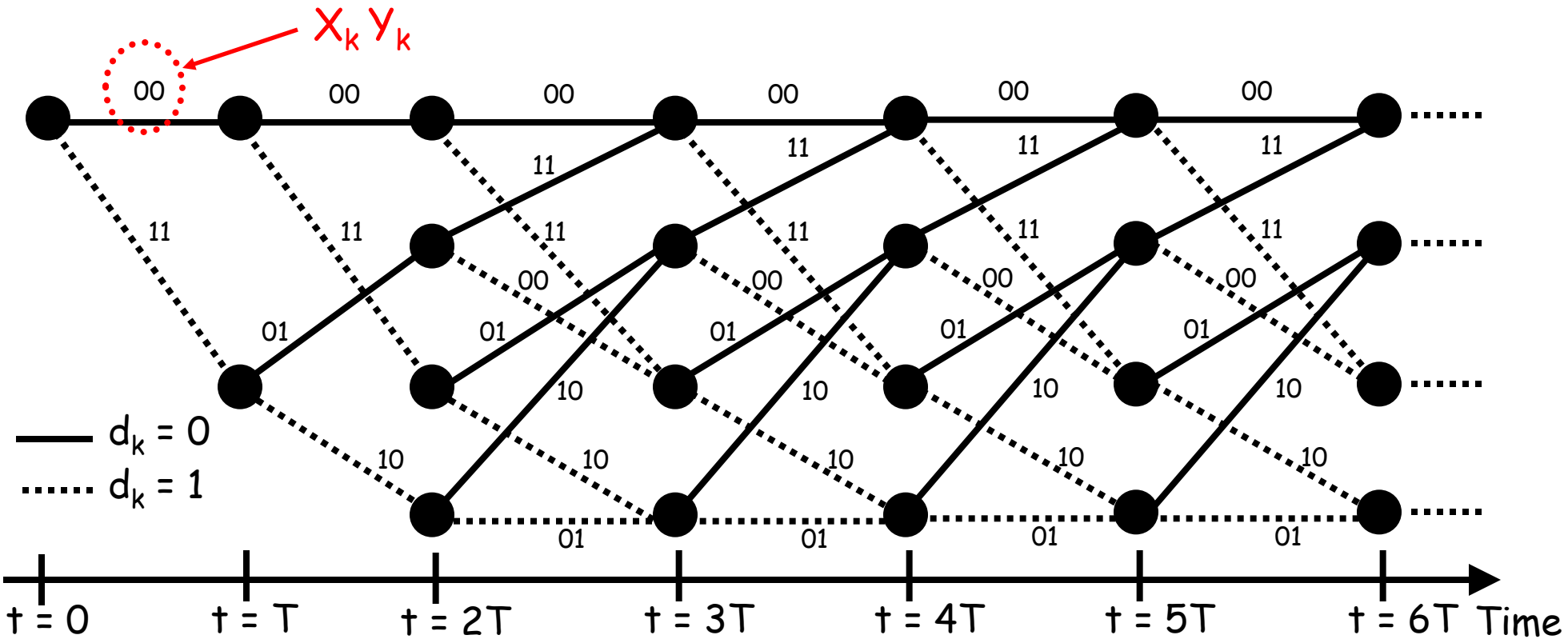
The values of d_{k-1} and d_k define the next state of the encoder.

→There are 4 states for this encoder.

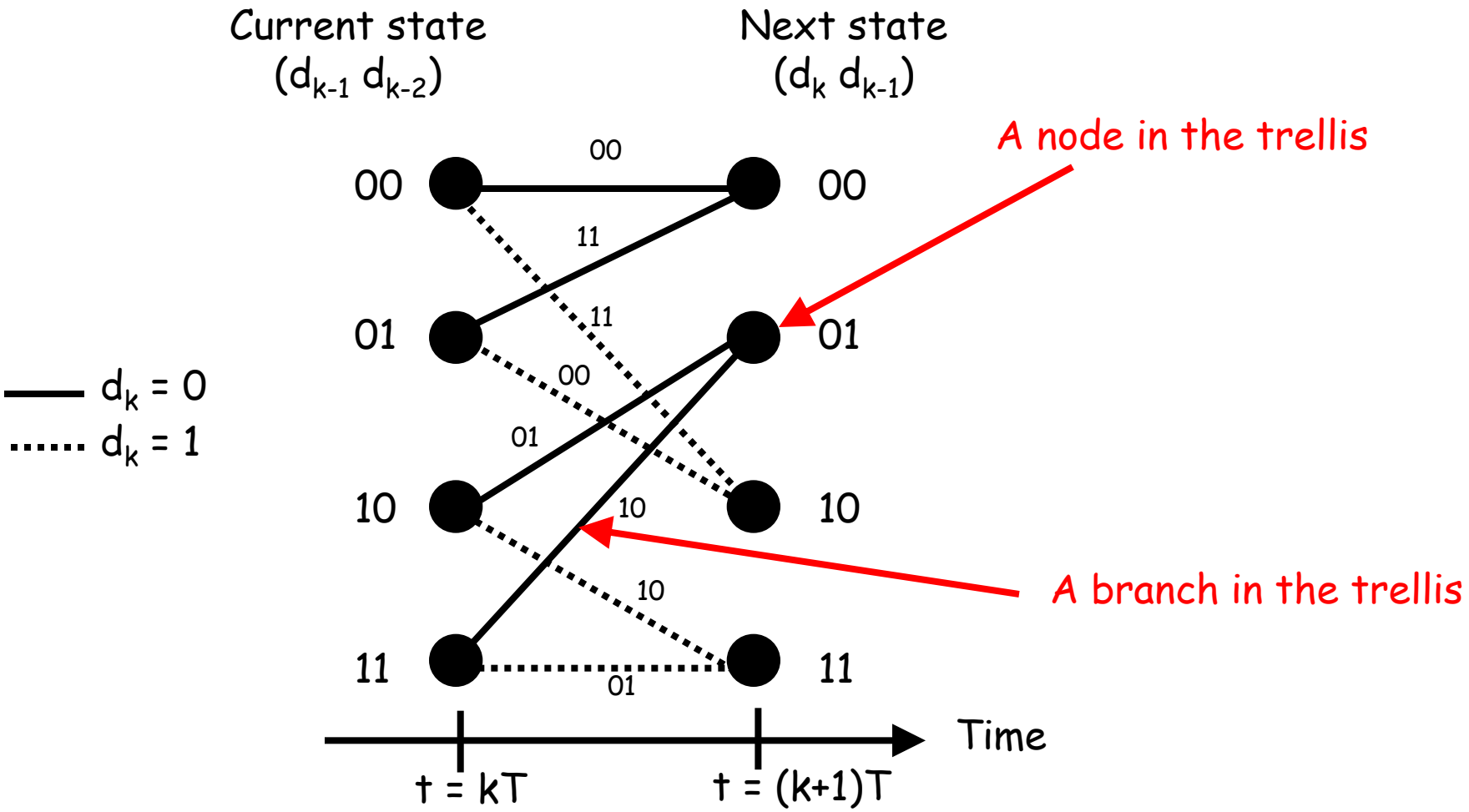
A code with a constraint length K has 2^{K-1} states.

Convolutional codes

The trellis diagram describes the operation of the encoder. The trellis of the (5, 7) encoder is shown below.



Convolutional codes



Convolutional codes

Consider an info sequence $\{d_k\} = \{11010011\}$, which can be seen as a message M with k = 8.

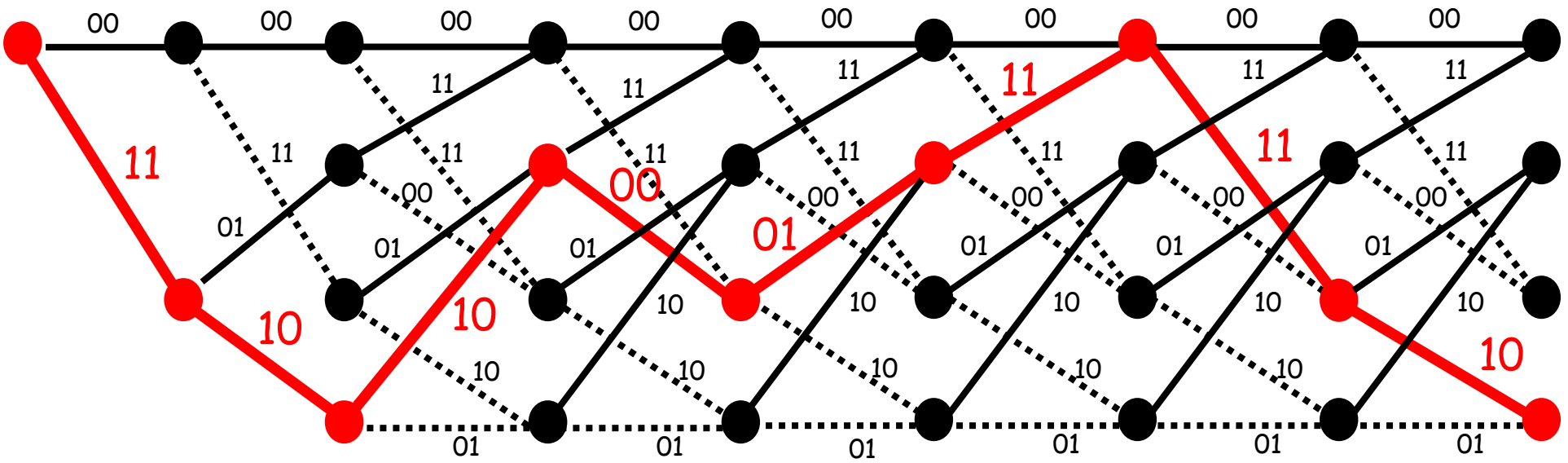
We can also use the following notation:

$$M = (11010011)$$

The corresponding coded sequence $\{X_k\, Y_k\}$ can be seen as a codeword C with n = 16.

This codeword corresponds to a particular path in the trellis.

Convolutional codes



→ $\{X_k\, Y_k\}$ = {11 10 10 00 01 11 11 10}

M = (11010011) → C = (1110100001111110)

A CC can be viewed as a block code for which k can be chosen arbitrarily by the user.

Convolutional codes

Each possible path in the trellis is a codeword.

We can apply the results obtained for block codes to *CCs*.

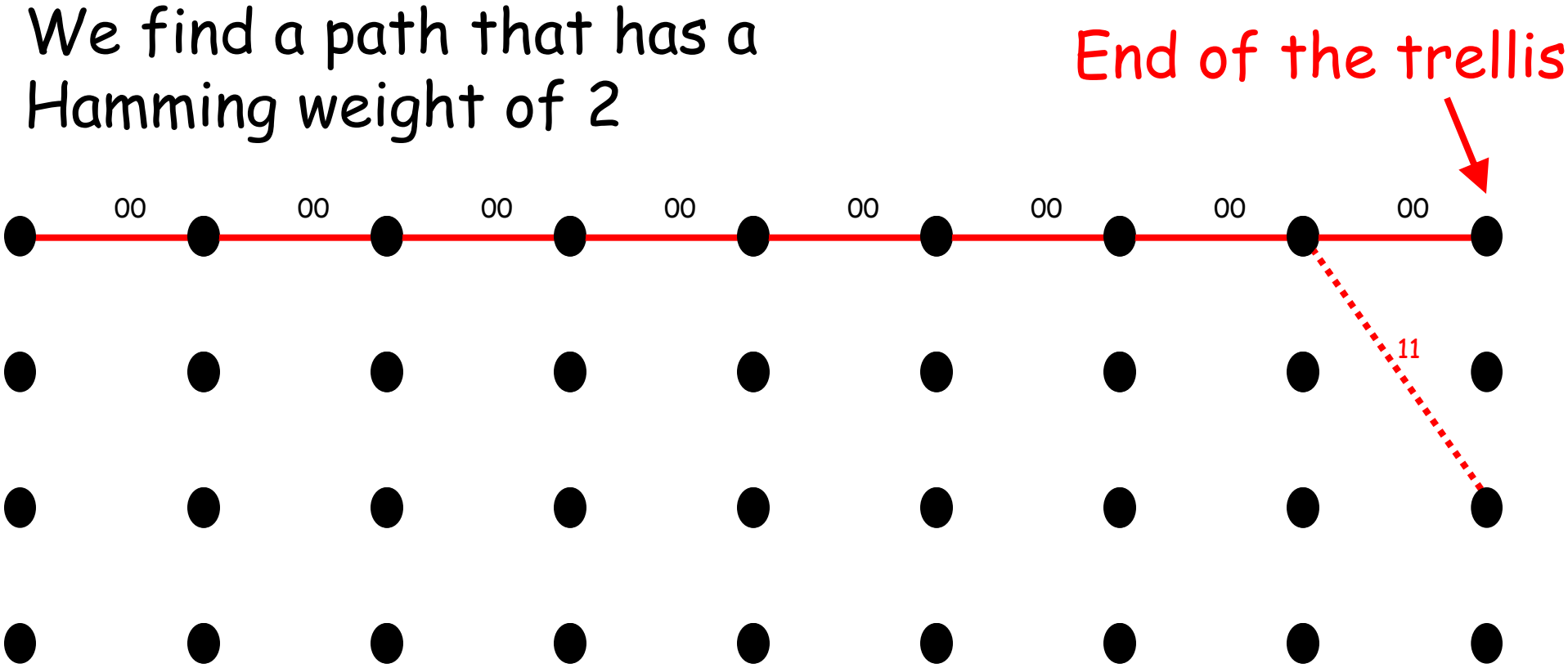
What is the value of d_{min} for the (5, 7) *CC*?

CCs are linear codes: we can assume that the all-zero codeword was transmitted and search for the codeword with minimum Hamming weight.

Consider once again the case where k = 8.

Convolutional codes

We find a path that has a Hamming weight of 2



d_{min} = 2 and number of codewords = 1 (w_{dmin} = 1)

Convolutional codes

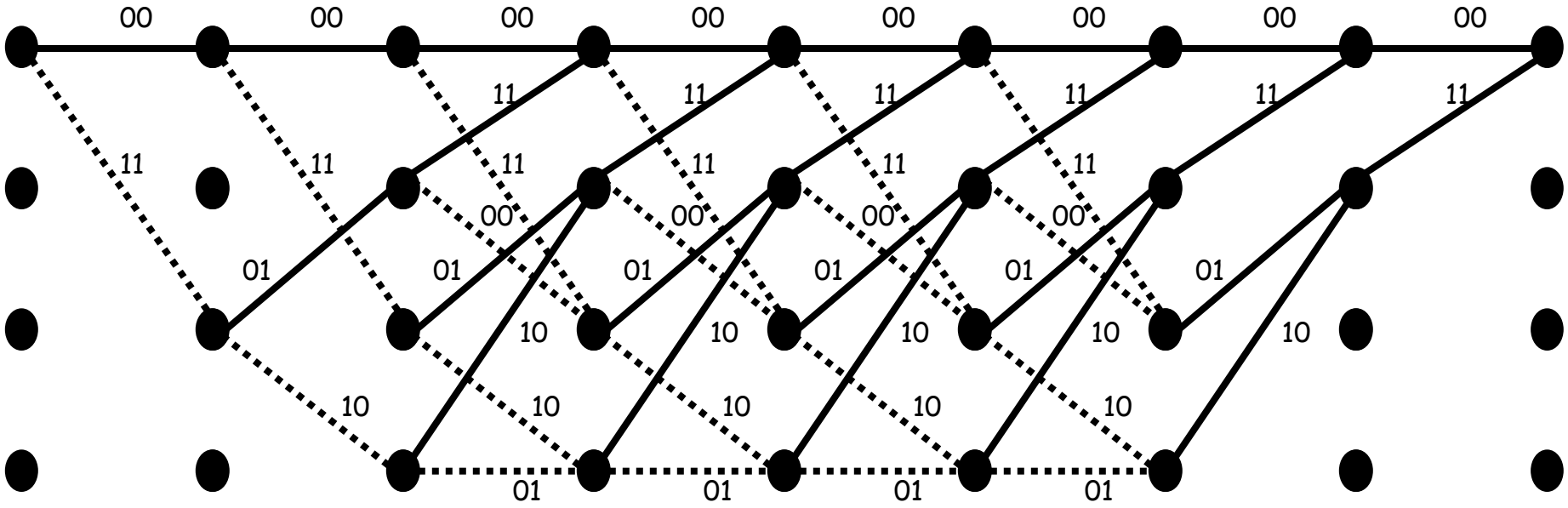
This result means that the asymptotic coding gain of the (5, 7) CC over uncoded BPSK is $10.\log(d_{min} \cdot R_c) = 10.\log(1) = 0$ dB.

Very disappointing result.

How to solve this problem? Make sure that the trellis returns to the zero state at the end of the message.

This can be done by appending two 0s ("tail bits") to the end of the message M. For instance, when k = 8, the resulting trellis is shown on the next slide.

Convolutional codes



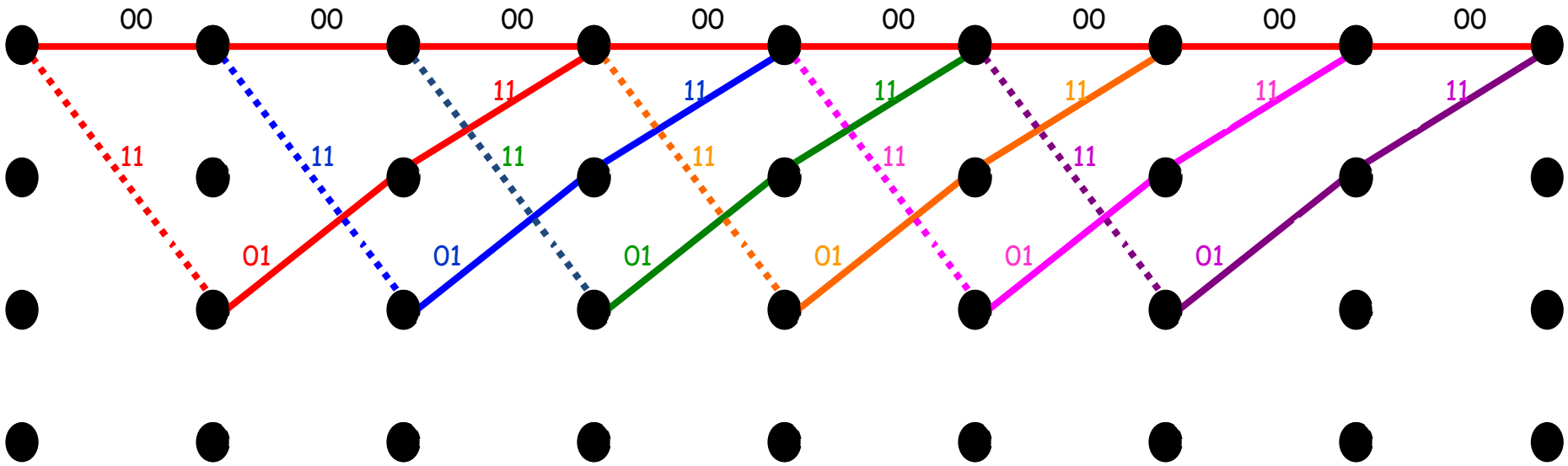
The trellis always starts from the zero state and returns to the zero state at the end of the message.

The use of 2 tail bits reduces the coding rate from 8/16 to 6/16. (However, the rate reduction due to tail bits would be negligible if k >> 2)

Convolutional codes

Does this technique increase d_{min}?

Assume once again $k = 8$ and $n = 16$. We now find $k - 2 = 6$ codewords with a Hamming weight of 5.



$$d_{min} = 5, w_{d_{min}} = k - 2 = 6.$$

Note: If $k \gg 2$, we can write $w_{d_{min}} \approx k$.

Union bound of convolutional codes

Do you remember the union bound expression ?

$$P_{eb} \leq \sum_{d=d_{min}}^{+\infty} e(d) \cdot \mathrm{erfc}\left(\sqrt{dR_c \frac{E_b}{N_0}}\right).$$

If we only consider the first term in the sum (as it is the dominant one at sufficiently high SNRs) and replace the bound by an approximation, we obtain the following expression for our (5, 7) CC:

$$P_{eb} \approx \frac{k-2}{2k} \cdot \mathrm{erfc}\left(\sqrt{\frac{5}{2}\frac{E_b}{N_0}}\right) \approx \frac{1}{2} \cdot \mathrm{erfc}\left(\sqrt{\frac{5}{2}\frac{E_b}{N_0}}\right).$$

Union bound of convolutional codes

What is the procedure to determine the union bound expression for a CC?

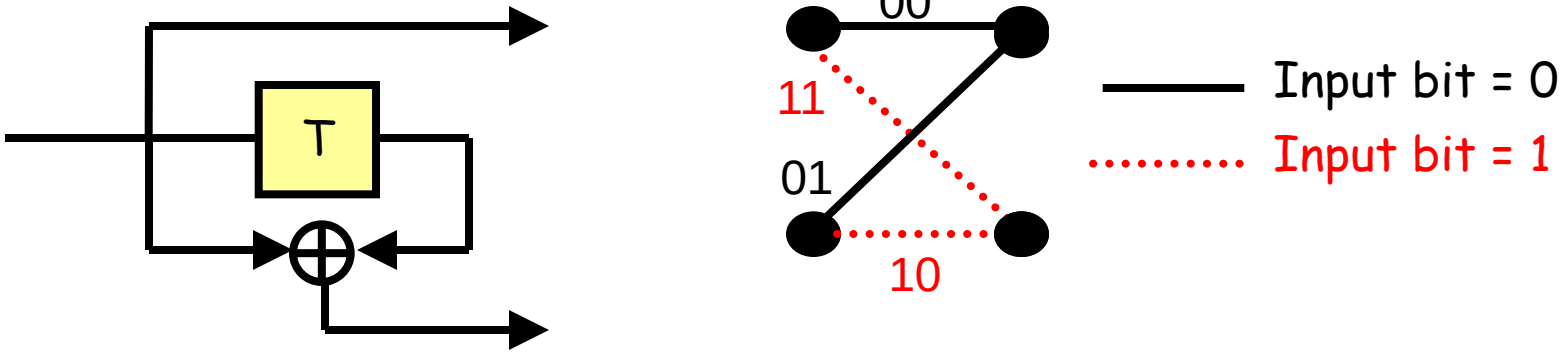
In other words, how can we determine the minimal distance d_{min} as well as the error coefficients $e(d) = \frac{w_d}{2\mathrm{k}}$, for $d = d_{min}, d_{min}+1, d_{min}+2, d_{min}+3$, etc.

In the context of CCs, the distance d_{min} is usually called "free distance" and denoted d_{free}.

We will use the fact that low-weight codewords can only be generated by encoding low-weight messages.

Union bound of convolutional codes

Example: 2-state, rate-1/2, (2, 3) code.

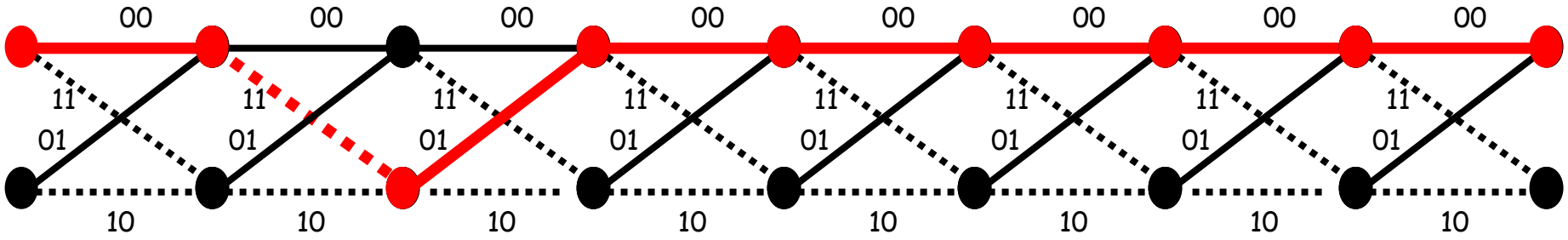


Let us determine the first 4 terms in the union bound for this code.

Low-weight codewords can only be generated by encoding low-weight messages → Consider weigth-1 messages first.

Union bound of convolutional codes

A k-bit message with a Hamming weight of 1 generates a path in the trellis (codeword) as shown below.



Such configuration leads to $d = 3$.

This distance is the minimal Hamming distance between codewords for this code: $d_{free} = 3$.

Union bound of convolutional codes

To determine the first four terms in the union bound, we must thus find the expressions of the error coefficients $e(d=3)$, $e(d=4)$, $e(d=5)$, and $e(d=6)$.

What is the value of $e(d=3)$ associated with a weigth-1 message?

To answer, we must determine the number of possible weight-1 messages in a k-bit message.

Union bound of convolutional codes

This number is given by the number of possible permutations of one element (a '10') in a word of (k-1) elements. ('10' instead of '1' because the error event corresponds to a '1' followed by a '0').

This number is thus equal to the binomial coefficient

$$\binom{k-1}{1} = k - 1.$$

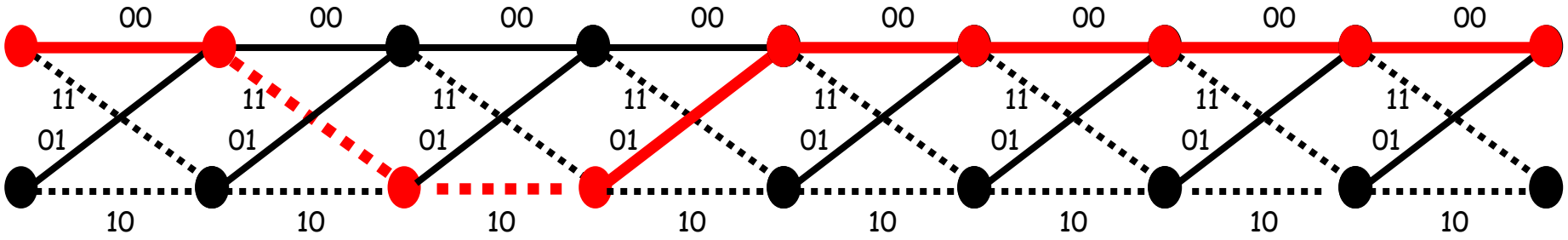
The contribution of weight-1 messages to the union bound is thus represented by the error coefficient

$e(d = 3) = \frac{k-1}{2\mathrm{k}} \approx \frac{1}{2}$.

Union bound of convolutional codes

Consider now weigth-2 messages for which we have two possibilities: (1) two successive 1s, (2) two 1s separated by at least one 0.

The configuration with successive 1s generates a path in the trellis (i.e., a codeword) as shown below.



Such configuration leads to d = 4.

Union bound of convolutional codes

The corresponding value of $\mathrm{e}(d=4)$ is the number of possible weight-2 messages with successive 1s in a k-bit message.

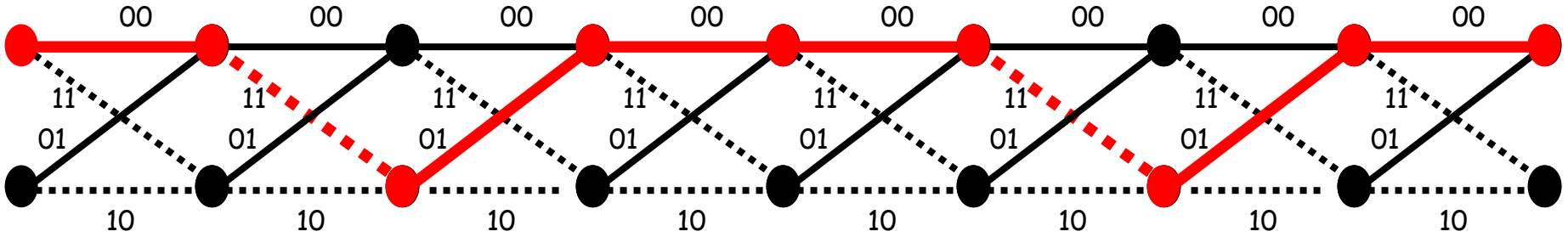
This number is given by the number of possible permutations of one element (a '110') in a word of (k-2) elements: $\binom{k-2}{1} = k-2.$

The contribution of these messages to the union bound is thus represented by the error coefficient

$e(d=4) = \frac{2(k-2)}{2\mathrm{k}} \approx 1.$

Union bound of convolutional codes

The configuration in which both 1s are separated by at least one 0 generates a codeword as shown below.



Such configuration leads to d = 6. The corresponding error coefficient $\mathrm{e}(d=6)$ is equal to the number of possible weight-2 messages with separated 1s in a k-bit message.

Union bound of convolutional codes

This number is given by the number of possible permutations of two elements (two '10's) in a word of (k-2) elements: $\binom{k-2}{2} = \frac{(k-2)(k-3)}{2} \approx \frac{k^2}{2}$.

The contribution of these particular messages to the union bound is thus represented by the error coefficient $e(d=6) \approx \frac{2k^2}{4\mathrm{k}} = \frac{k}{2}$.

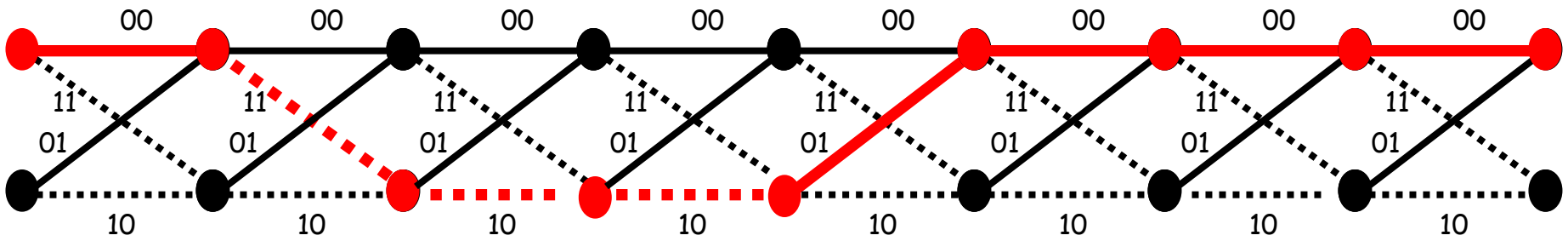
This error coefficient can unfortunately be very large since it is proportional to k.

And the procedure can be further extended to weight-3, 4, 5… messages until we are satisfied.

Union bound of convolutional codes

For weight-3 messages, there are 3 possibilities.

(1) Three successive 1s: $d = 5$, $e(d = 5) = \frac{3\binom{k-3}{1}}{2k} = \frac{3(k-3)}{2k} \approx \frac{3}{2}$.



(2) Two successive 1s and one isolated 1: $d = 7$.

(3) Three isolated 1s: $d = 9$.

Union bound of convolutional codes

For weight-4 messages, there are 4 possibilities.

(1) Four successive 1s: $d = 6$, $e(d = 6) = \frac{4\binom{k-4}{1}}{2k} = \frac{4(k-4)}{2k} \approx 2$.

(2) Two successive 1s separated from two successive 1s by at least one 0: $d = 8$.

(3) Three successive 1s separated from the 4th 1 by at least one 0: $d = 8$.

(4) Four isolated 1s: $d = 12$.

Union bound of convolutional codes

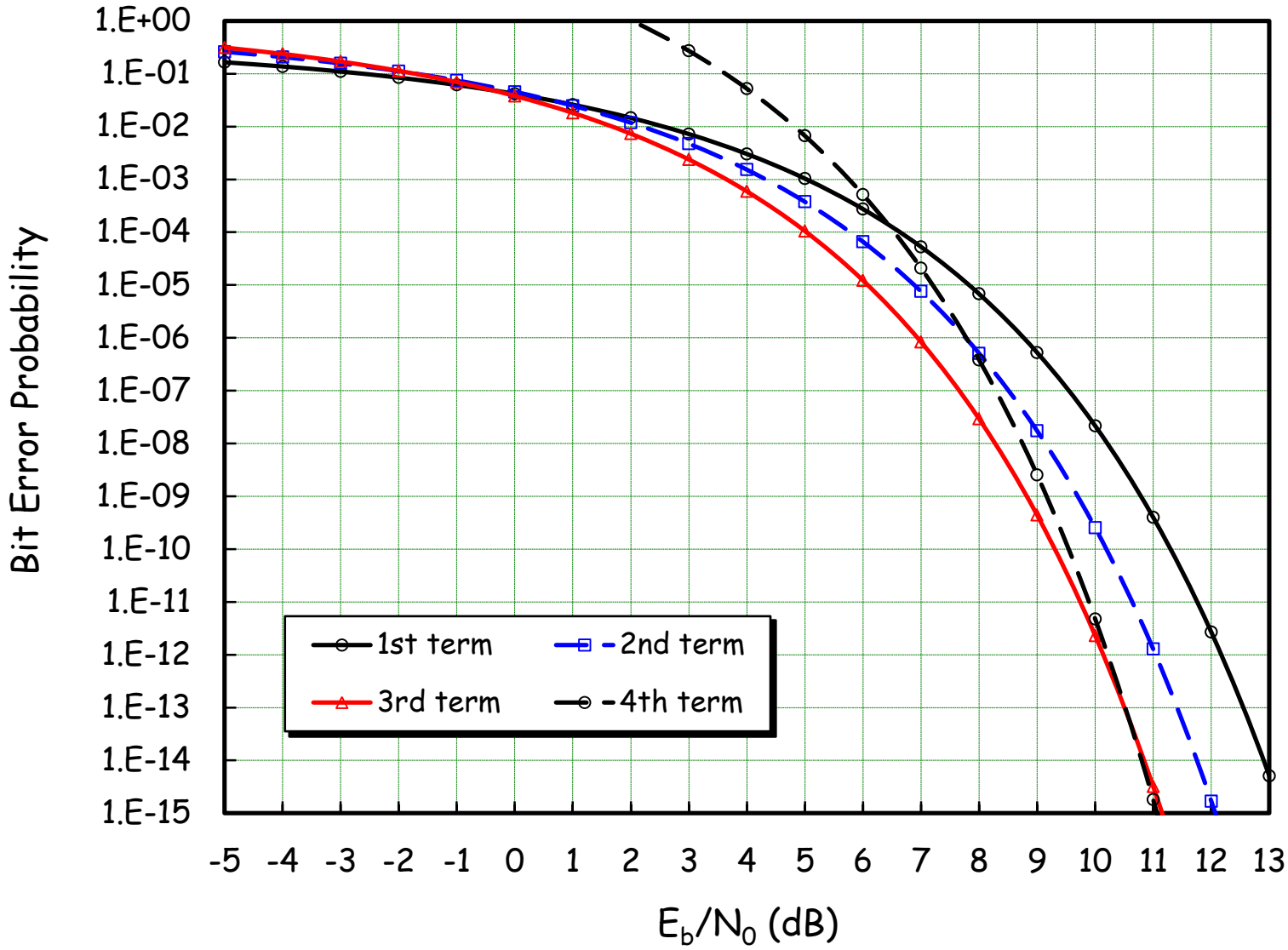
$$P_{eb} \le e(d_{free}) \cdot \operatorname{erfc}\left(\sqrt{d_{free} R_c \frac{E_b}{N_0}}\right)$$
$$+ \; e(d_{free}+1) \cdot \operatorname{erfc}\left(\sqrt{(d_{free}+1) R_c \frac{E_b}{N_0}}\right)$$
$$+ \; e(d_{free}+2) \cdot \operatorname{erfc}\left(\sqrt{(d_{free}+2) R_c \frac{E_b}{N_0}}\right)$$
$$+ \; e(d_{free}+3) \cdot \operatorname{erfc}\left(\sqrt{(d_{free}+3) R_c \frac{E_b}{N_0}}\right) + \cdots$$

Union bound of convolutional codes

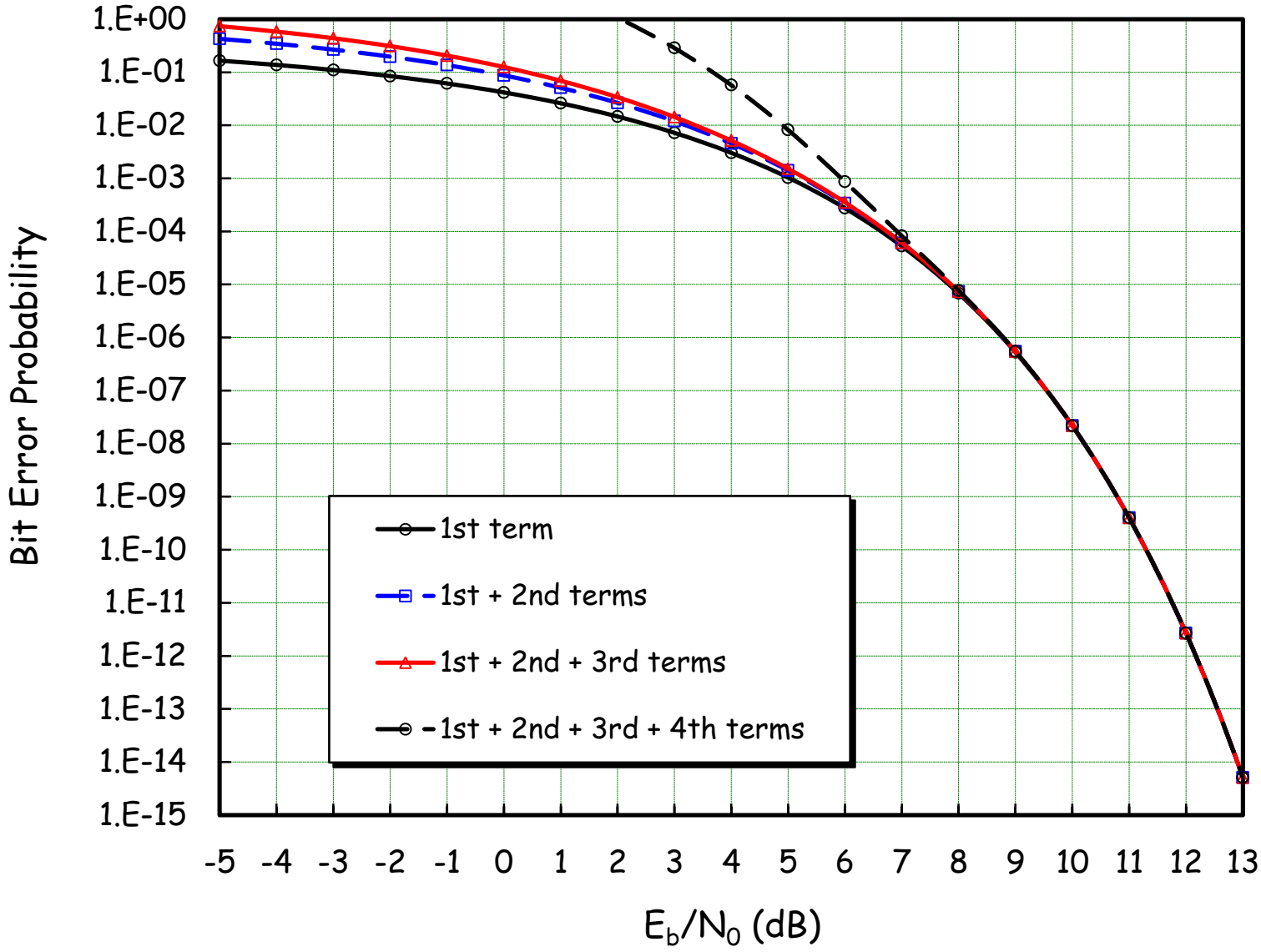
$$P_{eb} \leq \frac{1}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{3}{2}\frac{E_b}{N_0}}\right) + \frac{2}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{4}{2}\frac{E_b}{N_0}}\right)$$
$$+ \frac{3}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{5}{2}\frac{E_b}{N_0}}\right) + \frac{k+4}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{6}{2}\frac{E_b}{N_0}}\right) + \cdots$$

The next three slides illustrates some interesting results obtained for this (2, 3) CC when the message length is k = 1000.

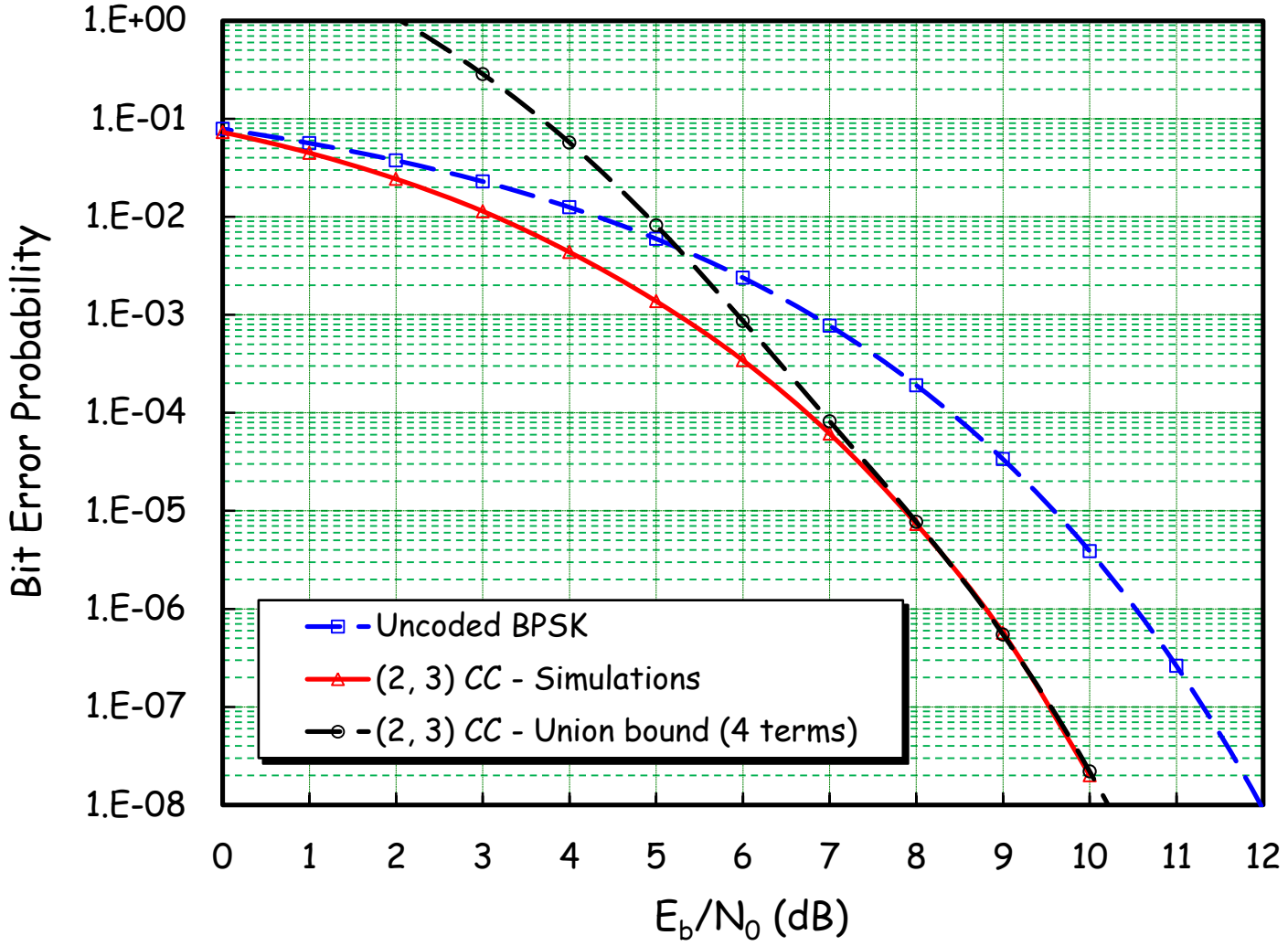
The first term becomes the dominant one in the union bound for SNR values greater than 7-8 dB.



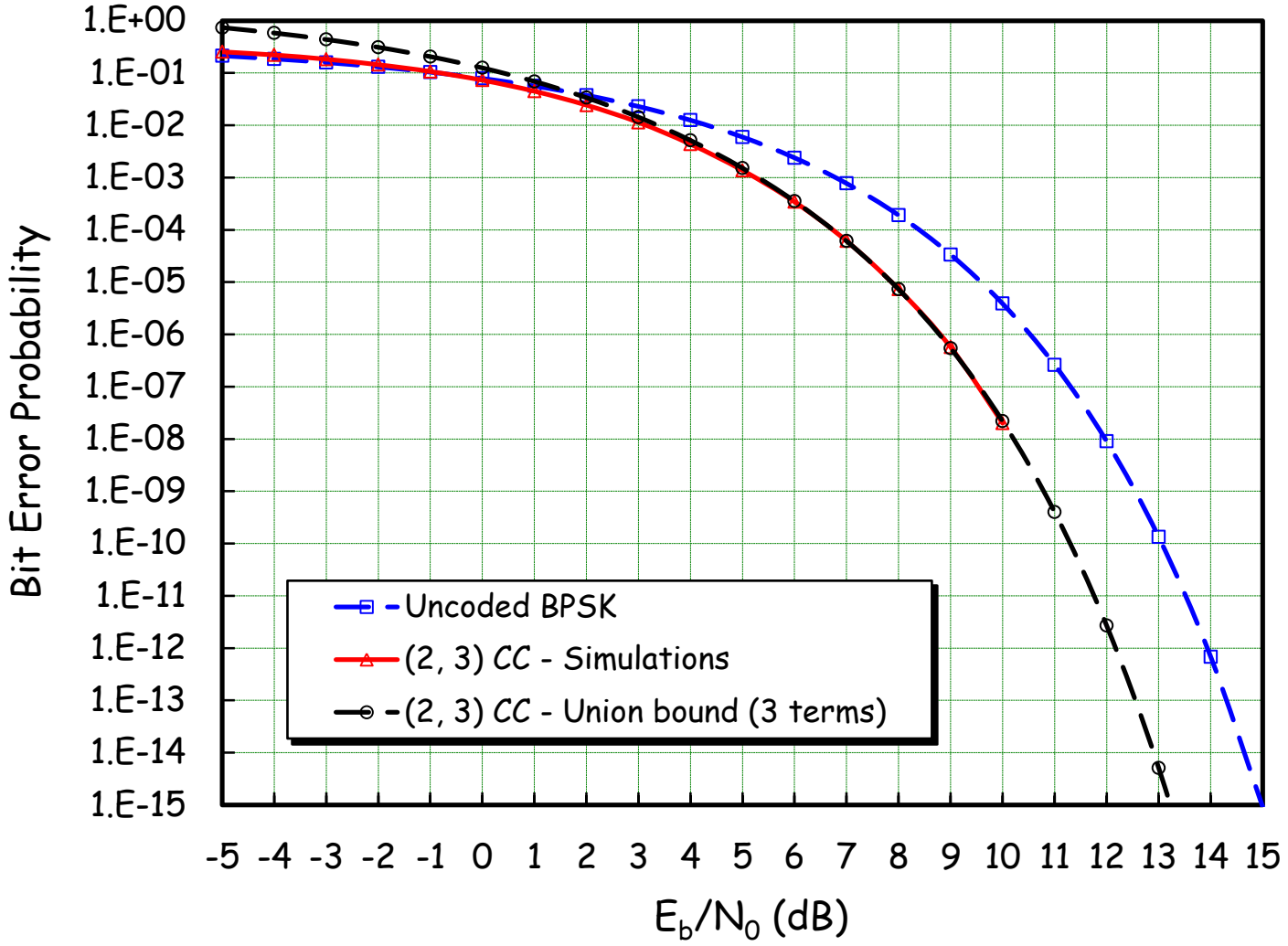
The first term provides a good approximation of the union bound for SNR values greater than 7-8 dB.



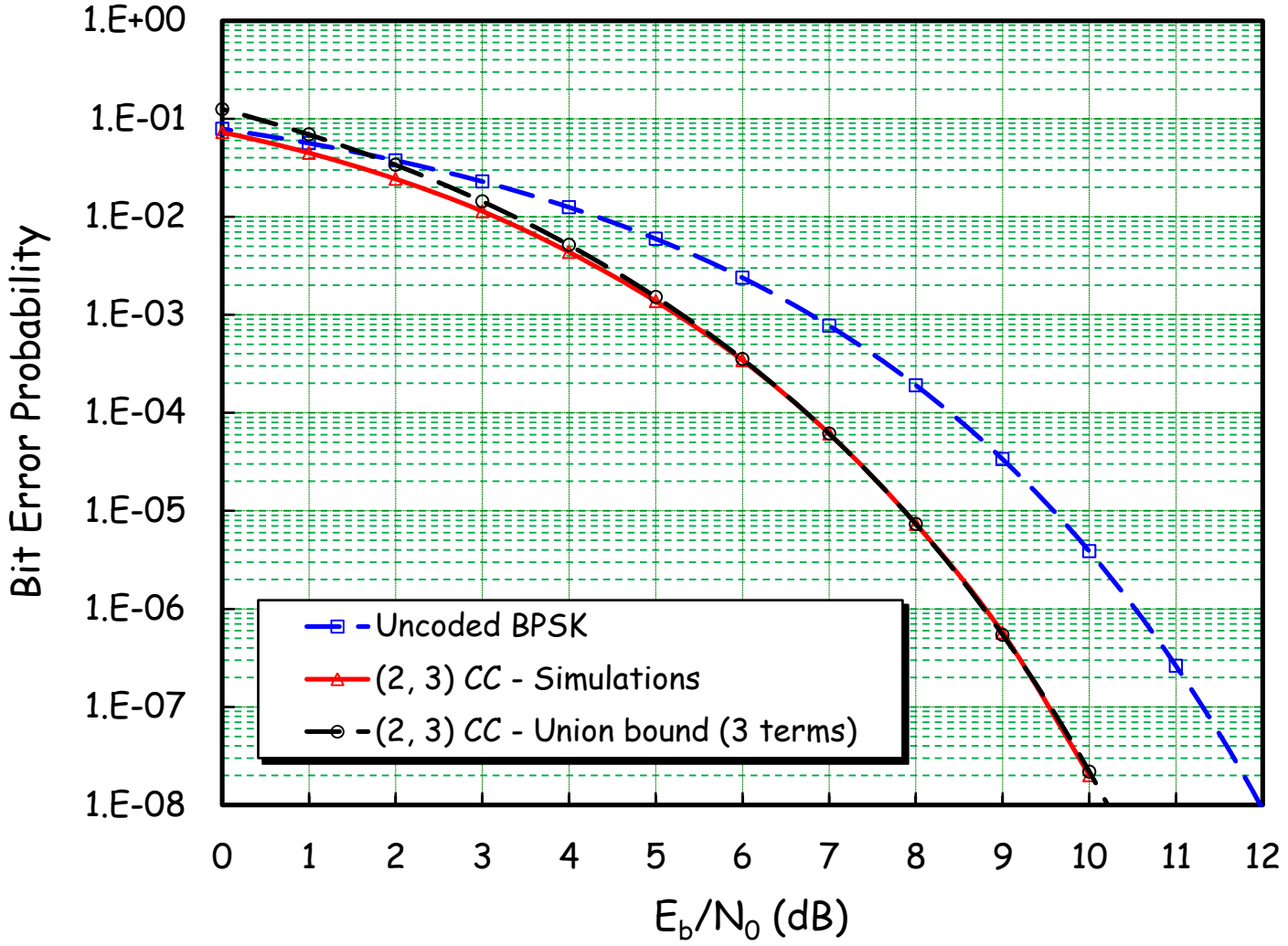
The union bound using 4 terms is an excellent approximation of the actual error performance for SNR values greater than 7-8 dB.



Relying on 3, instead of 4, terms provides a much better approximation of the actual error performance for SNR values below 7 dB.



There is probably no need to be too zealous when deriving the union bound for a CC. Here, the first 3 terms are sufficient, and the 4th term is one too many.



Convolutional codes

At high SNRs, the dominant term in the union bound is the first term, i.e. the term associated with the free distance d_{free}.

In addition, the union bound becomes actually very tight as $\frac{E_b}{N_0} \to +\infty$. We can thus write

$$P_{eb} \approx e\left(d_{free}\right) \cdot \operatorname{erfc}\left(\sqrt{d_{free} R_c \frac{E_b}{N_0}}\right) \text{ as } \frac{E_b}{N_0} \to +\infty.$$

The asymptotic coding gain over uncoded BPSK is given by $G \approx 10 \cdot \log_{10}\left(d_{free} \cdot R_c\right)$ dB.

In the case of the (2, 3) code, we find $G \sim 1.76$ dB.

Convolutional codes

$$P_{eb} \approx e\left(d_{free}\right)\cdot \mathrm{erfc}\left(\sqrt{d_{free} R_c \frac{E_b}{N_0}}\right) \text{ as } \frac{E_b}{N_0} \to +\infty.$$

However, "$\frac{E_b}{N_0} \to +\infty$" can sometimes refer to a very high SNR that is not of practical interest.

This equation might thus not always be accurate at SNRs of practical interest (corresponding to $P_{eb} = 10^{-4} - 10^{-7}$). Including the next few terms associated with the distances $d_{free} + 1$, $d_{free} + 2$, $d_{free} + 3$, etc, can be necessary to obtain a more accurate expression of the bit error probability.

Convolutional codes

The union bound results have as usual been obtained by assuming the use of an ML decoding algorithm to decode CCs.

Does such algorithm even exist?

Yes, it does and is known as the "Viterbi's algorithm".

We have also seen that, for distances $d \geq 2d_{free}$, determining the error coefficient values can become a complicated task because we must then also account for the paths that leave the zero state and come back to it more than once.

Convolutional codes

Because of this, distances $d \geq 2d_{free}$ are associated with large error coefficient values $e(d)$ that increase with the message length k.

In practice, we do not however need to worry too much about distances $d \geq 2d_{free}$ and their corresponding error coefficients because they play a minor role in the union bound.

To determine the error coefficients values, we can focus on the paths that leave the zero state and come back to it only once.

Convolutional codes

If we only take into account the paths that leave the zero state and come back to it once, i.e. we choose to ignore distances $d \geq 2d_{free}$, then the process for finding the union bound expression can actually be simplified.

In this case, for any distance $d < 2d_{free}$, the parameter w_d is given by $w_d \approx k \cdot N_d$, where N_d denotes the total Hamming weight of the input sequences associated with the coded sequences that:
(1) leave the zero state at time t = 0;
(2) are at distance d from the all-zero.

Convolutional codes

Then, the union bound expression for a *CC* becomes

$$P_{eb} \leq \sum_{d=d_{min}}^{+\infty} \frac{k \cdot N_d}{2k} \cdot \mathrm{erfc}\left(\sqrt{dR_c \frac{E_b}{N_0}}\right),$$

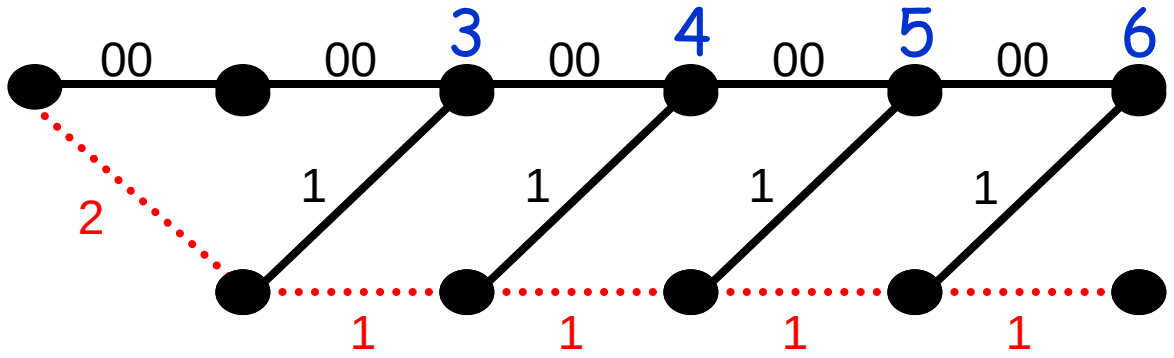
which is equivalent to

$$P_{eb} \leq \sum_{d=d_{min}}^{+\infty} \frac{N_d}{2} \cdot \mathrm{erfc}\left(\sqrt{dR_c \frac{E_b}{N_0}}\right).$$

The first few terms of the union bound for a *CC* can be found by determining the values of the error coefficients now expressed as $e(d) = \frac{N_d}{2}$, for $d = d_{free}, d_{free} + 1, d_{free} + 2, d_{free} + 3$, etc.

Convolutional codes

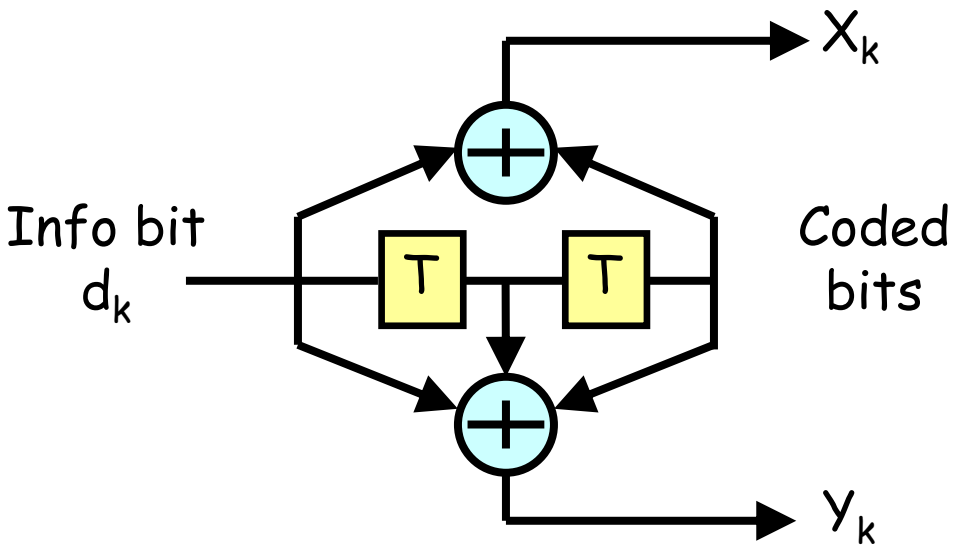
For our (2, 3) CC, we can work on the trellis below to find the first 3 terms in the union bound.



We see that $d_{free} = 3$, $N_{d_{free}} = 1$, $N_{d_{free}+1} = 2$, $N_{d_{free}+2} = 3$, leading to $e(d = 3) = \frac{1}{2}$, $e(d = 4) = \frac{2}{2}$ and $e(d = 5) = \frac{3}{2}$. This simple technique however produces erroneous results for all distances $d \geq 6$.

The best convolutional codes

4-state (K = 3), rate-1/2, (5, 7) code

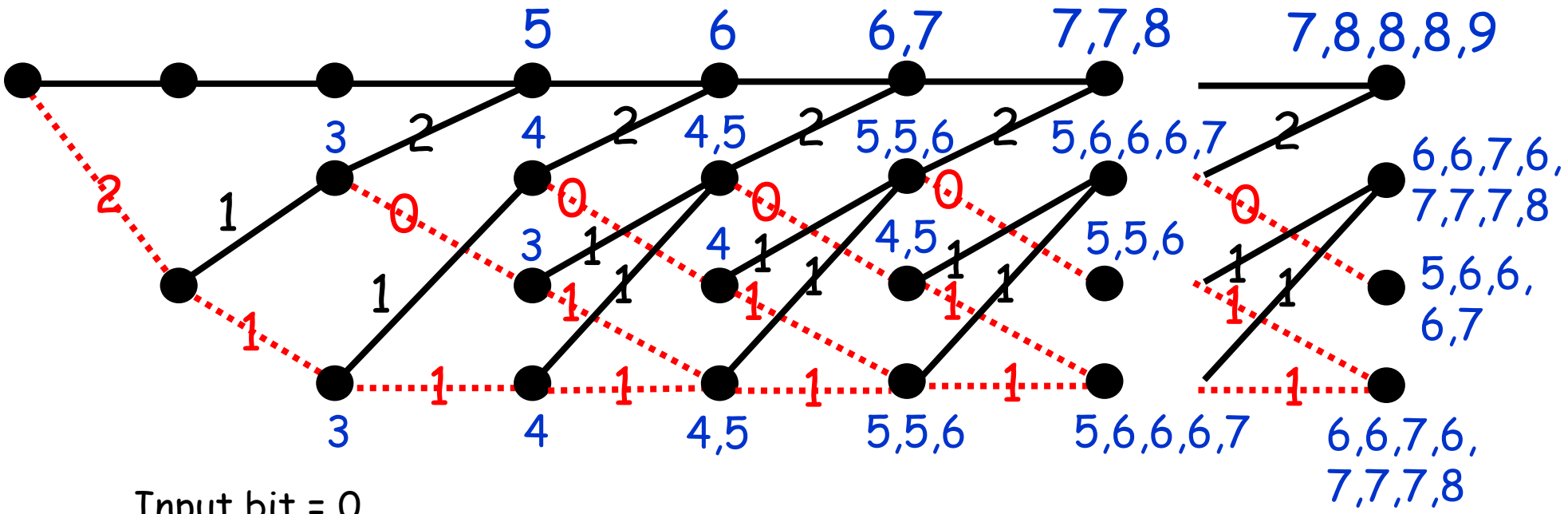


We find $d_{free} = 5$, $e(d_{free}) = \frac{1}{2}$, $e(d_{free} + 1) = \frac{4}{2}$, and $e(d_{free} + 2) = \frac{12}{2}$.

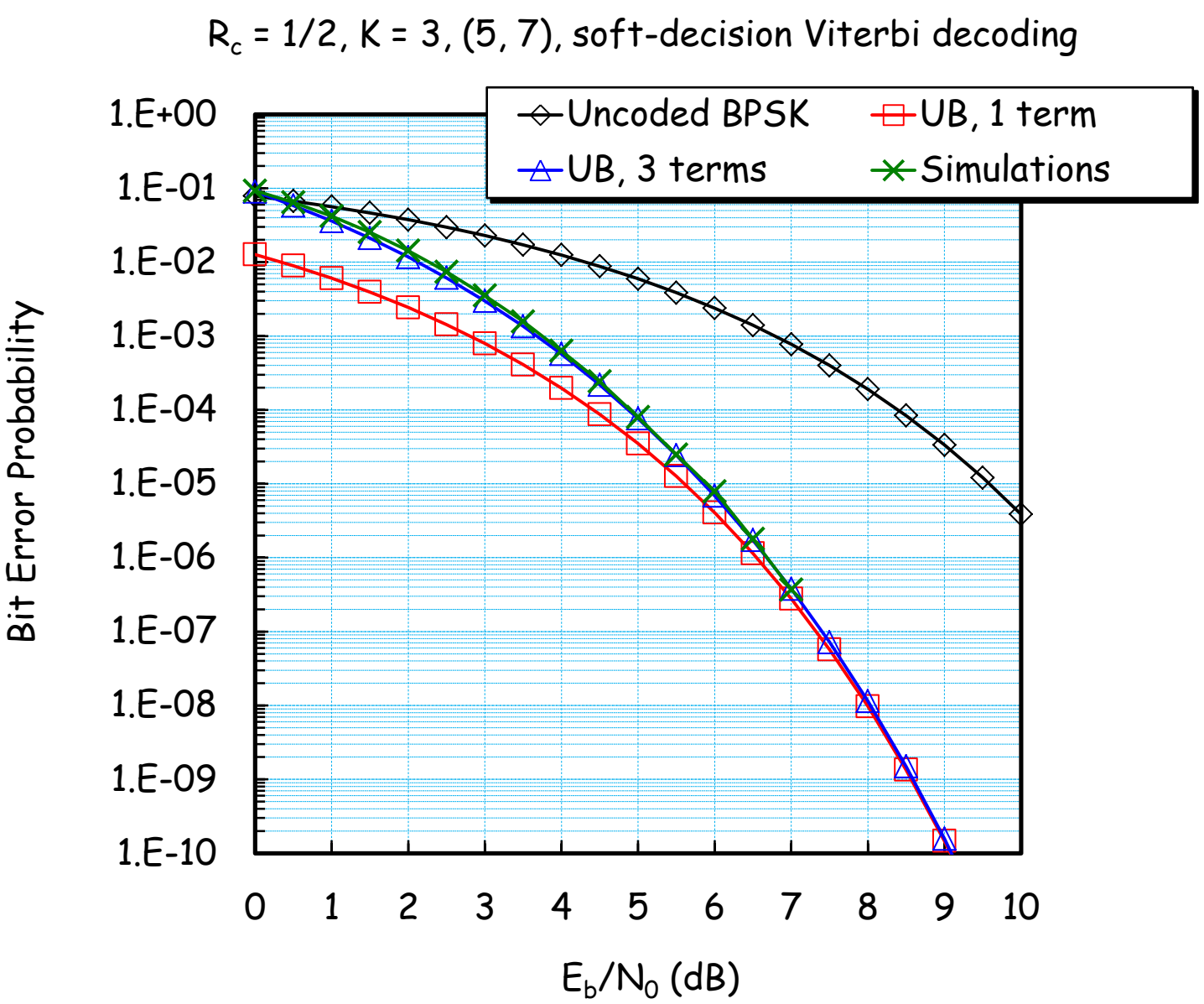
Asymptotic coding gain $G \sim$ 3.98 dB.

The best convolutional codes

4-state (K = 3), rate-1/2, (5, 7) code

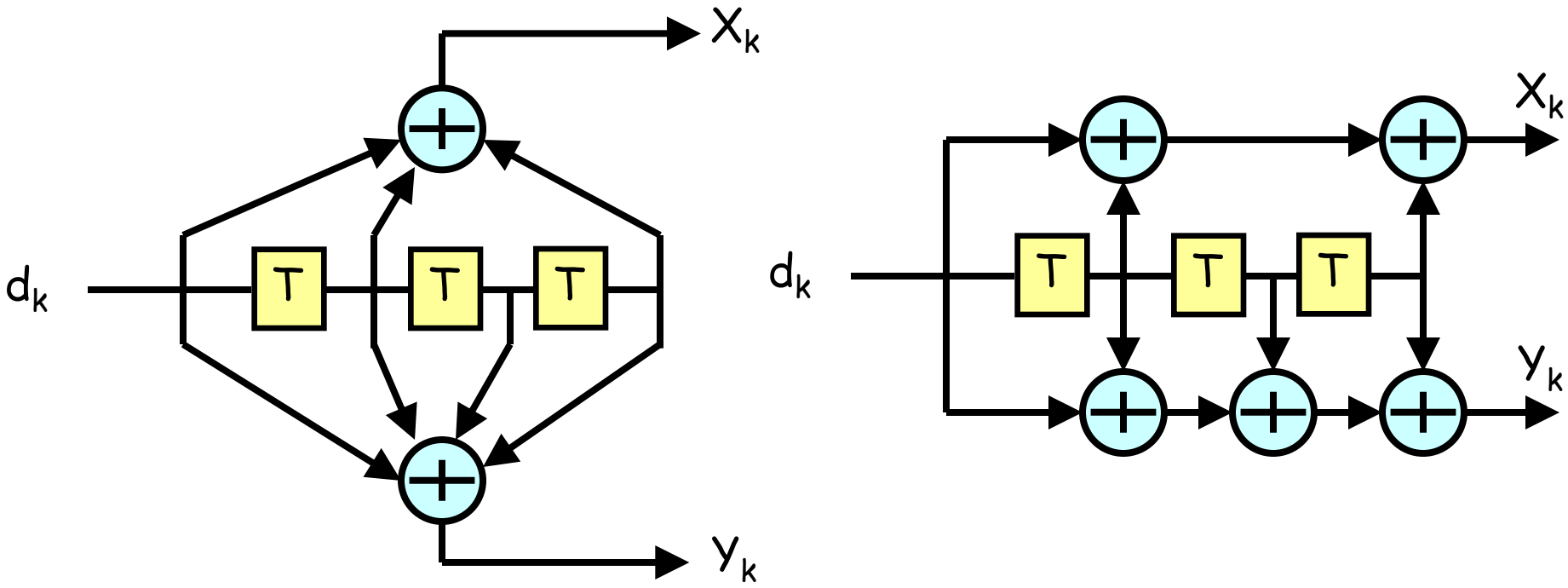


We find $N_{d=5} = 1$, $N_{d=6} = 2 + 2 = 4$, and $N_{d=7} = 3 + 3 + 3 + 3 = 12$.

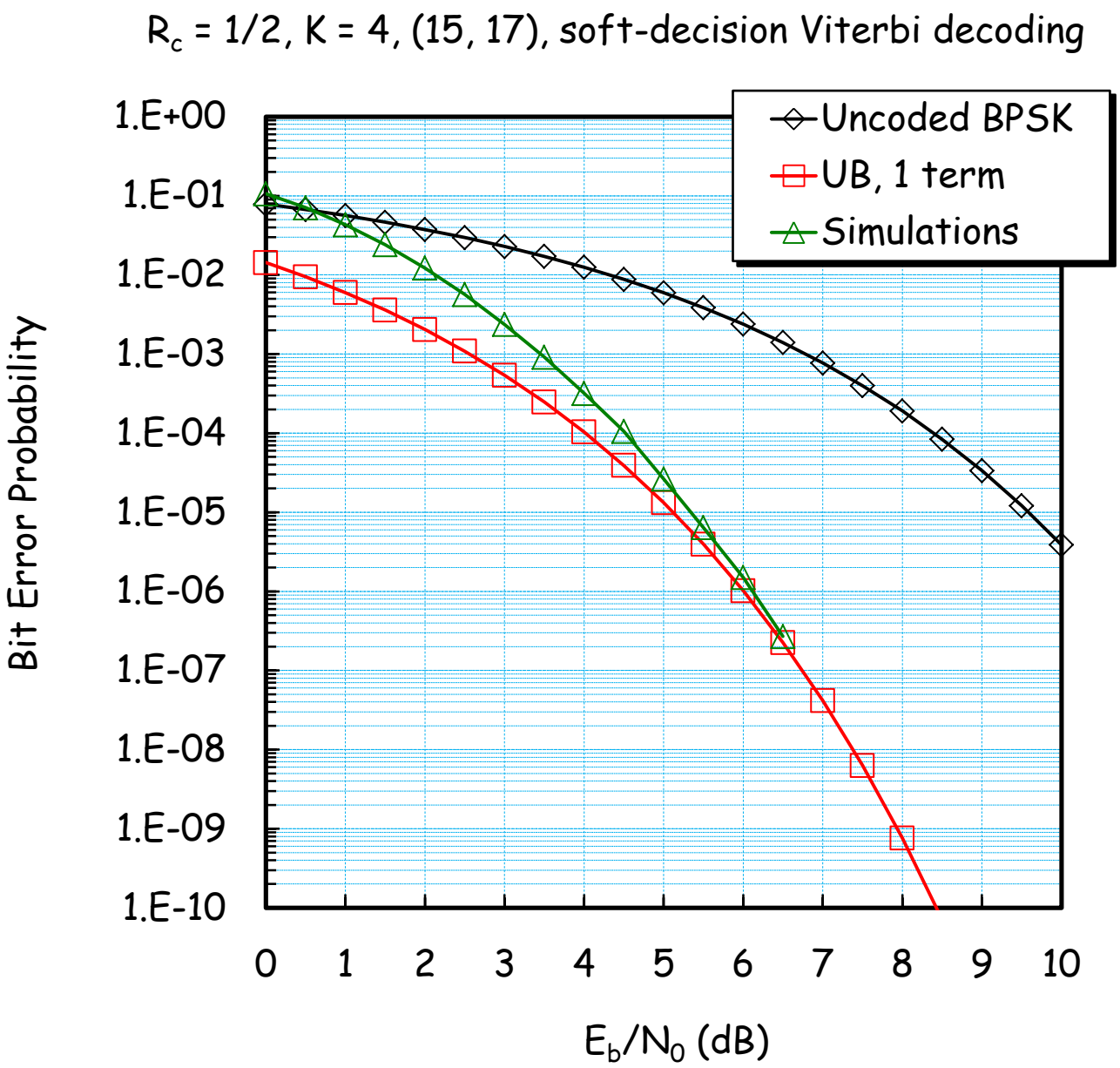

R_c = 1/2, K = 3, (5, 7), soft-decision Viterbi decoding
Uncoded BPSK
UB, 1 term
UB, 3 terms
Simulations
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
1.E-09
1.E-10
0
1
2
3
4
5
6
7
8
9
10
E_b/N_0 (dB)

The best convolutional codes

8-state (K = 4), rate-1/2, (15, 17) code

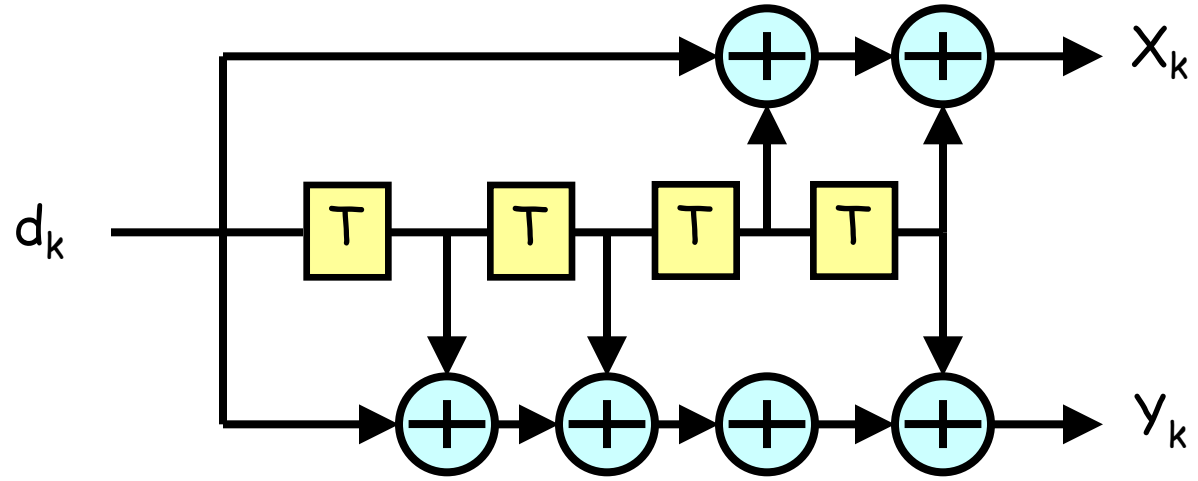


$$d_{free} = 6,\ e(d_{free}) = \frac{2}{2} \rightarrow G \sim 4.77\ \text{dB}.$$

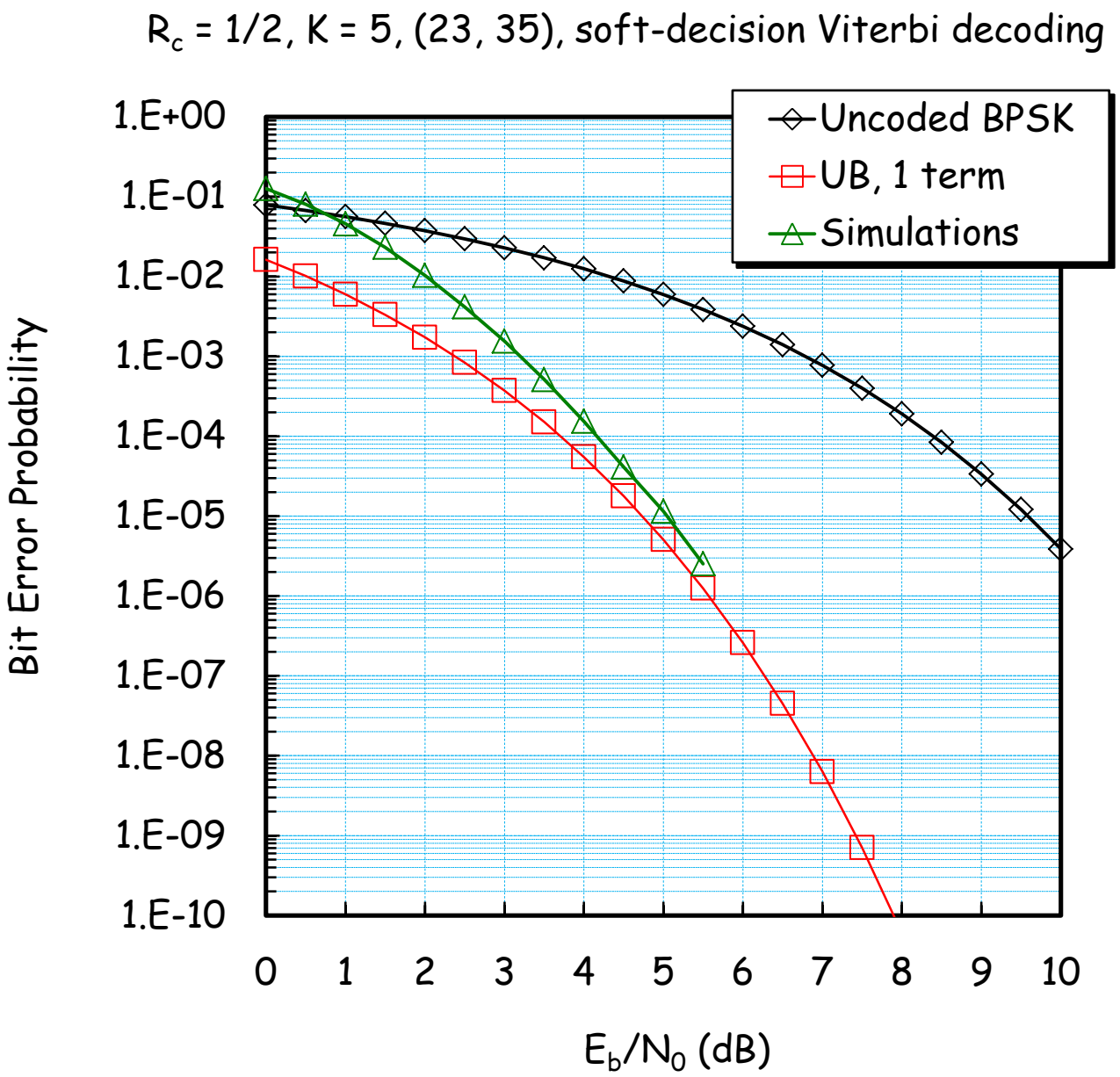

R_c = 1/2, K = 4, (15, 17), soft-decision Viterbi decoding
Uncoded BPSK
UB, 1 term
Simulations
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
1.E-09
1.E-10
0
1
2
3
4
5
6
7
8
9
10
E_b/N_0 (dB)

The best convolutional codes

16-state (K = 5), rate-1/2, (23, 35) code

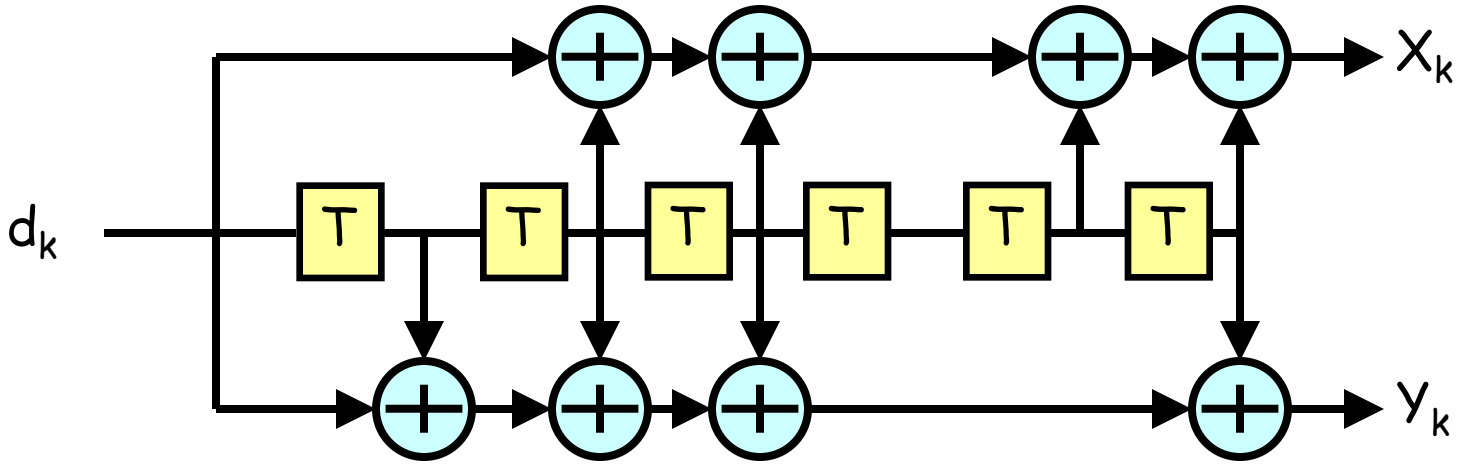


$d_{free} = 7, e(d_{free}) = \frac{4}{2} \rightarrow G \sim$ 5.44 dB.

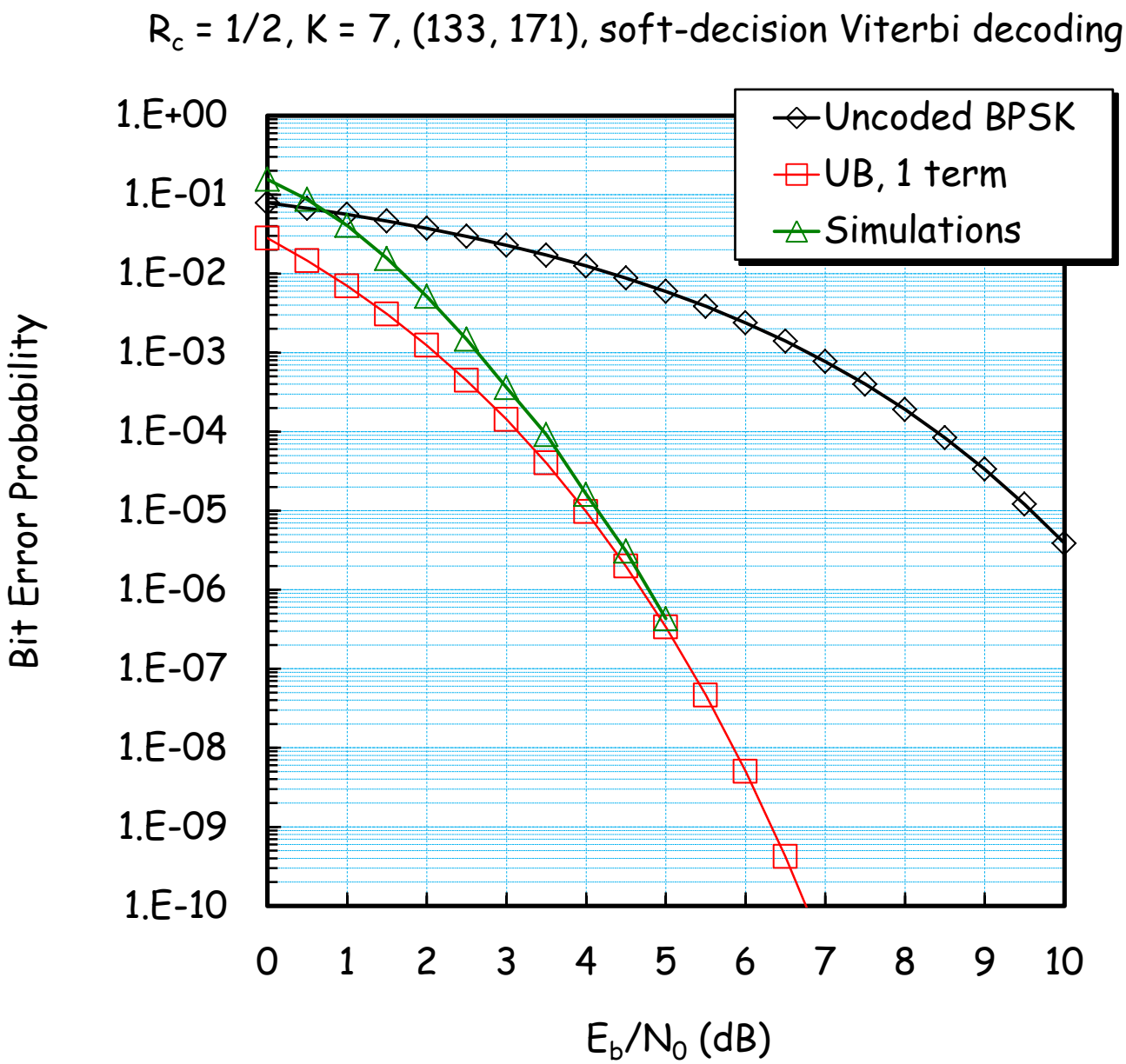

R_c = 1/2, K = 5, (23, 35), soft-decision Viterbi decoding
Uncoded BPSK
UB, 1 term
Simulations
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
1.E-09
1.E-10
0
1
2
3
4
5
6
7
8
9
10
E_b/N_0 (dB)

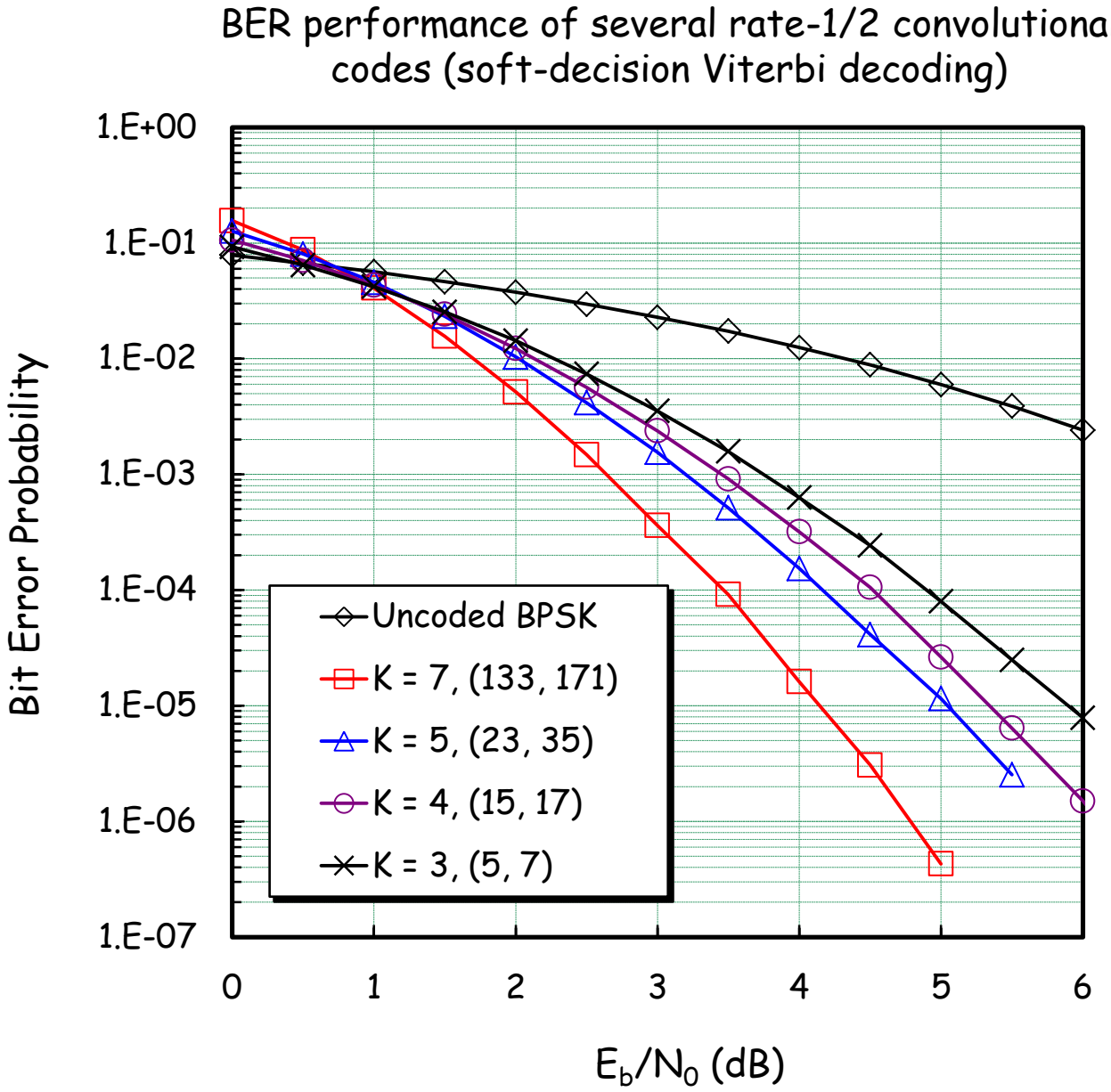
The best convolutional codes

64-state (K = 7), rate-1/2, (133, 171) code



$d_{free} = 10, e(d_{free}) = \frac{36}{2} \rightarrow G \sim$ 6.99 dB.


R_c = 1/2, K = 7, (133, 171), soft-decision Viterbi decoding
Uncoded BPSK
UB, 1 term
Simulations
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
1.E-08
1.E-09
1.E-10
0
1
2
3
4
5
6
7
8
9
10
E_b/N_0 (dB)

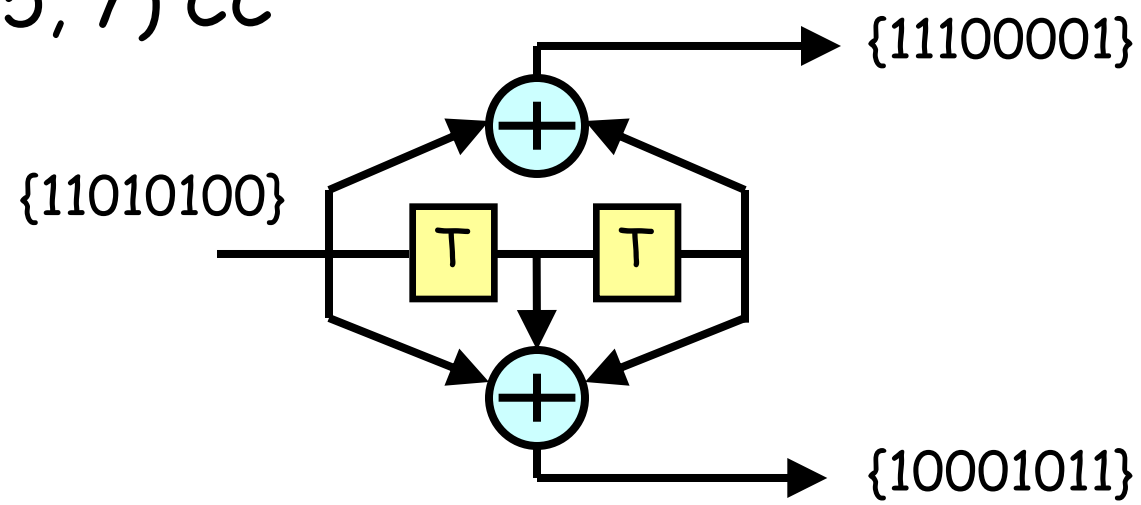

BER performance of several rate-1/2 convolutional codes (soft-decision Viterbi decoding)
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
0
1
2
3
4
5
6
E_b/N_0 (dB)
Uncoded BPSK
K = 7, (133, 171)
K = 5, (23, 35)
K = 4, (15, 17)
K = 3, (5, 7)

Punctured convolutional codes

What about other coding rates?

The most popular technique to generate CCs with coding rates $R_c > \frac{1}{2}$ consists of periodically deleting coded bits at the output of a rate-1/2 encoder.

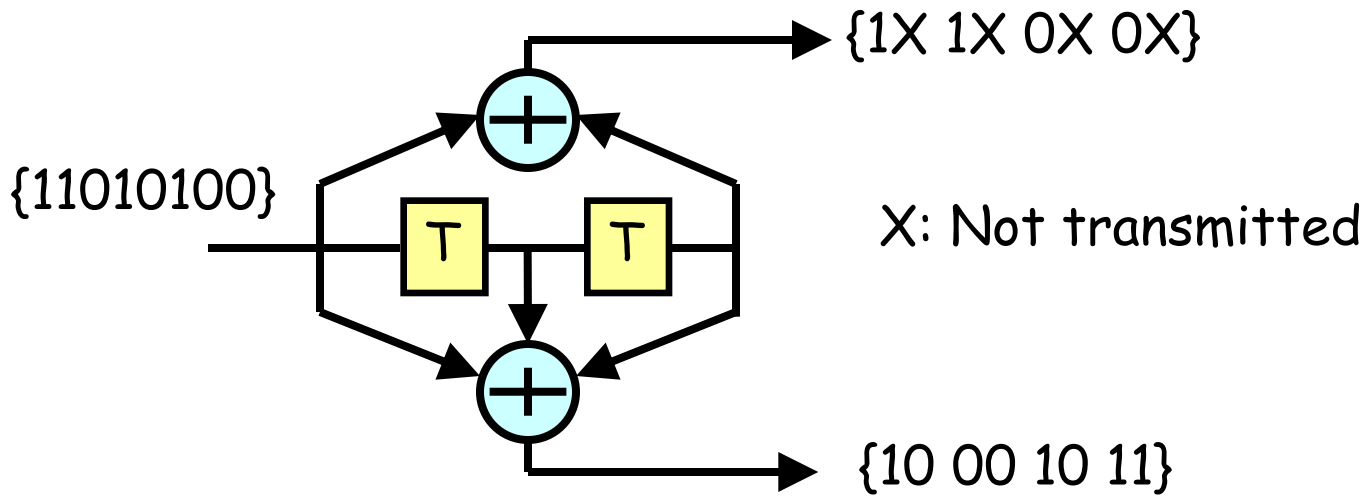
Ex: Rate-1/2, (5, 7) CC



Punctured convolutional codes

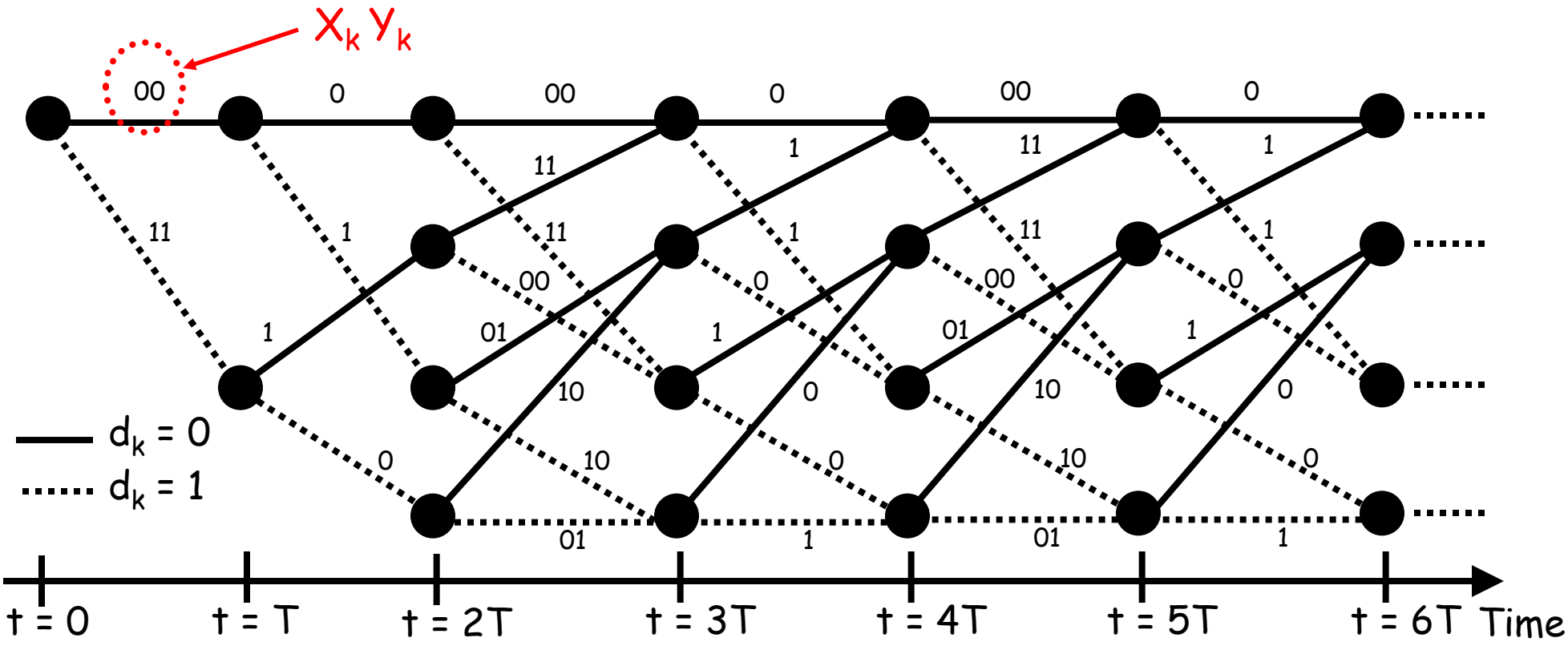
A 4-state, rate-2/3 CC can be generated from such code using, for instance, the puncturing pattern

$$P = \begin{bmatrix} 10 \\ 11 \end{bmatrix}$$



Punctured convolutional codes

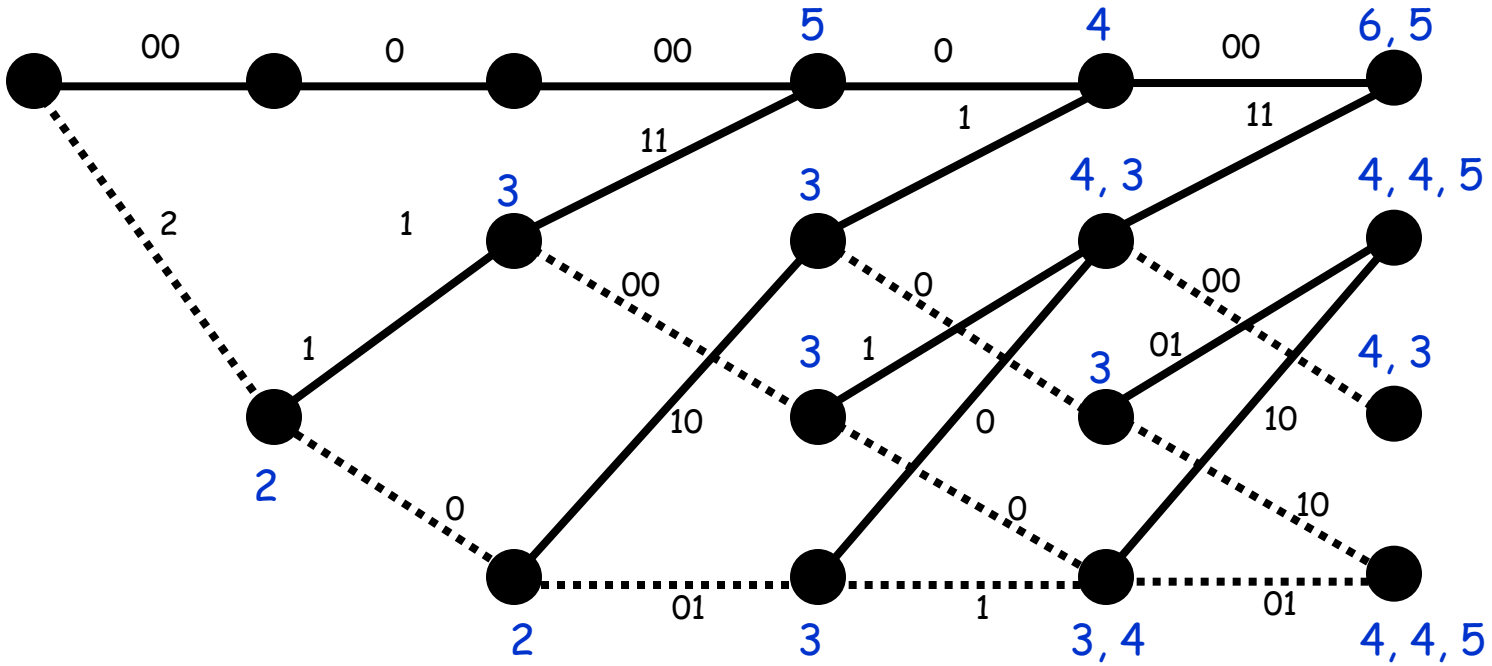
Trellis of the rate-2/3 code thus obtained:



Punctured convolutional codes

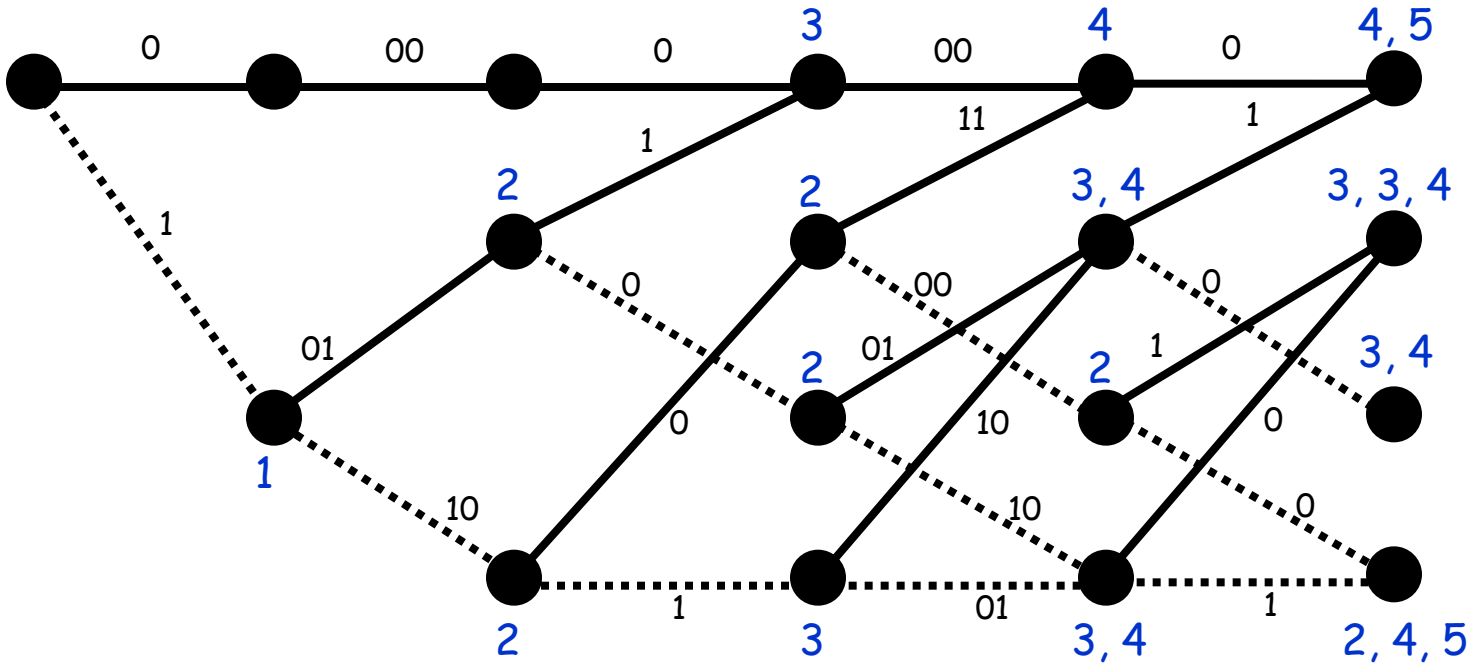
Two cases to consider in order to evaluate d_{free}:

(1) The erroneous codeword leaves the all-zero codeword during the "non-punctured" time-slot.



Punctured convolutional codes

(2) The erroneous codeword leaves the all-zero codeword during the "punctured" time-slot.



Take the worst-case scenario: $d_{free} = 3$.

Puncturing a code decreases its free distance.

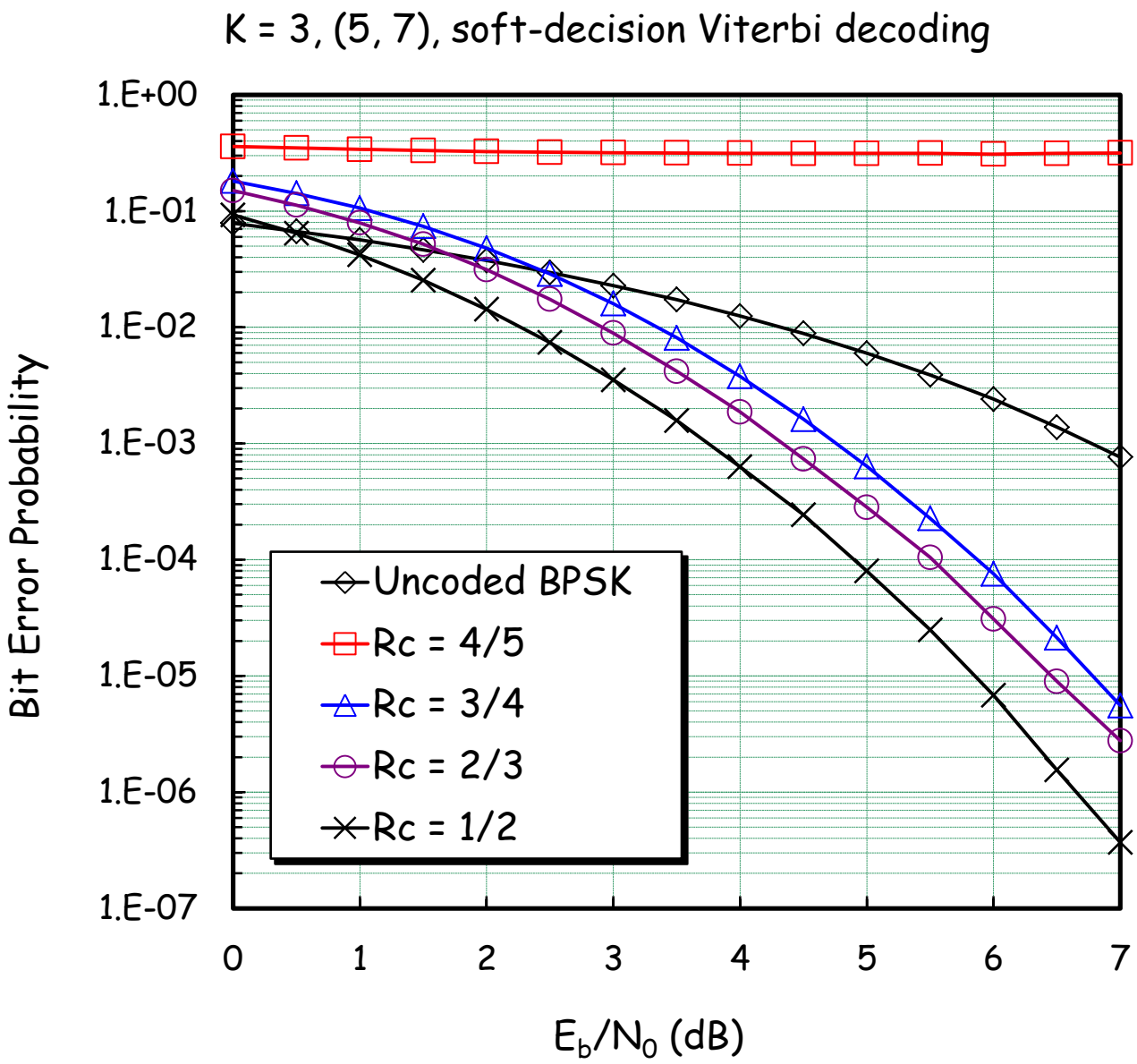
Punctured convolutional codes

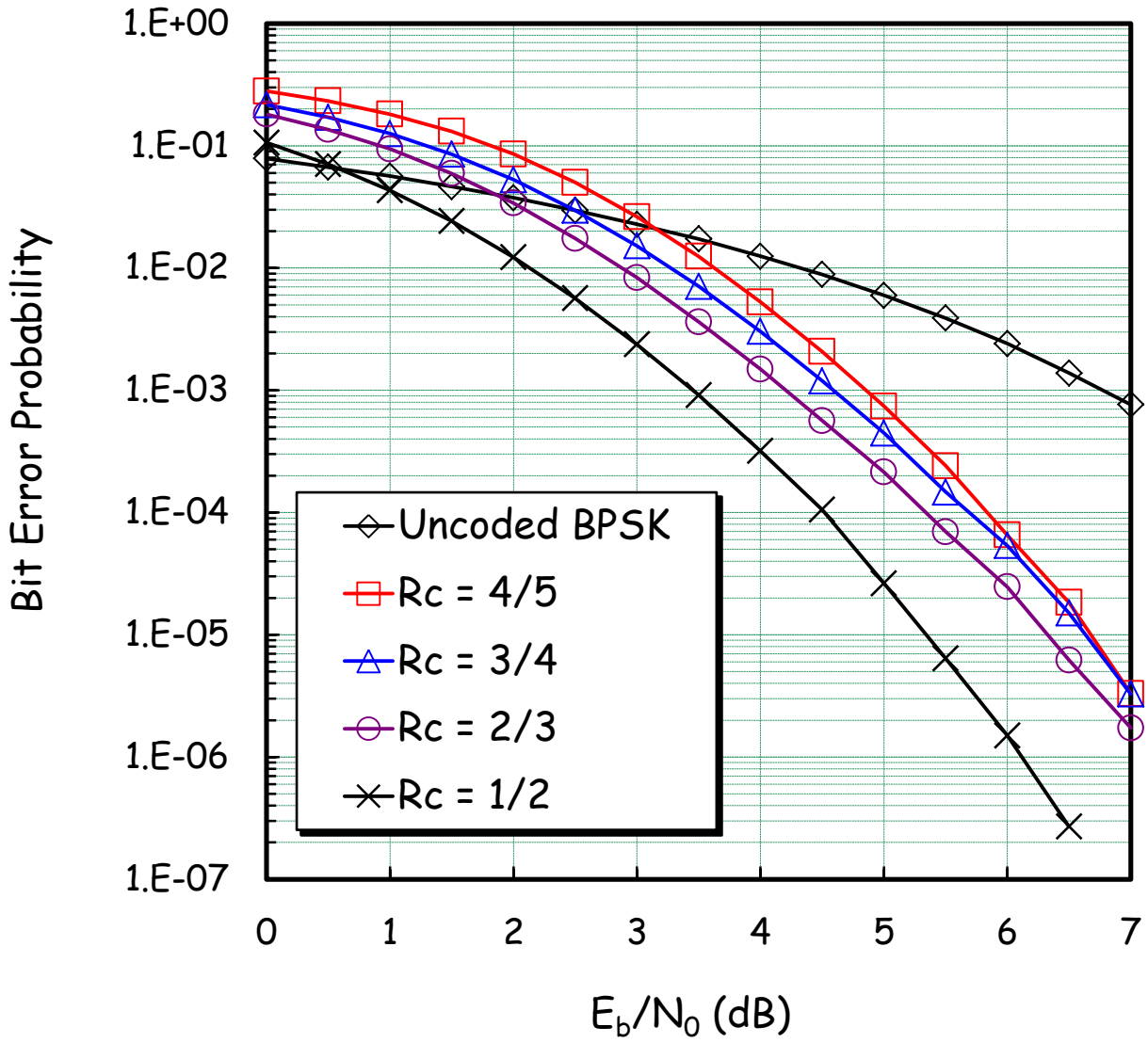
A few examples of punctured codes:

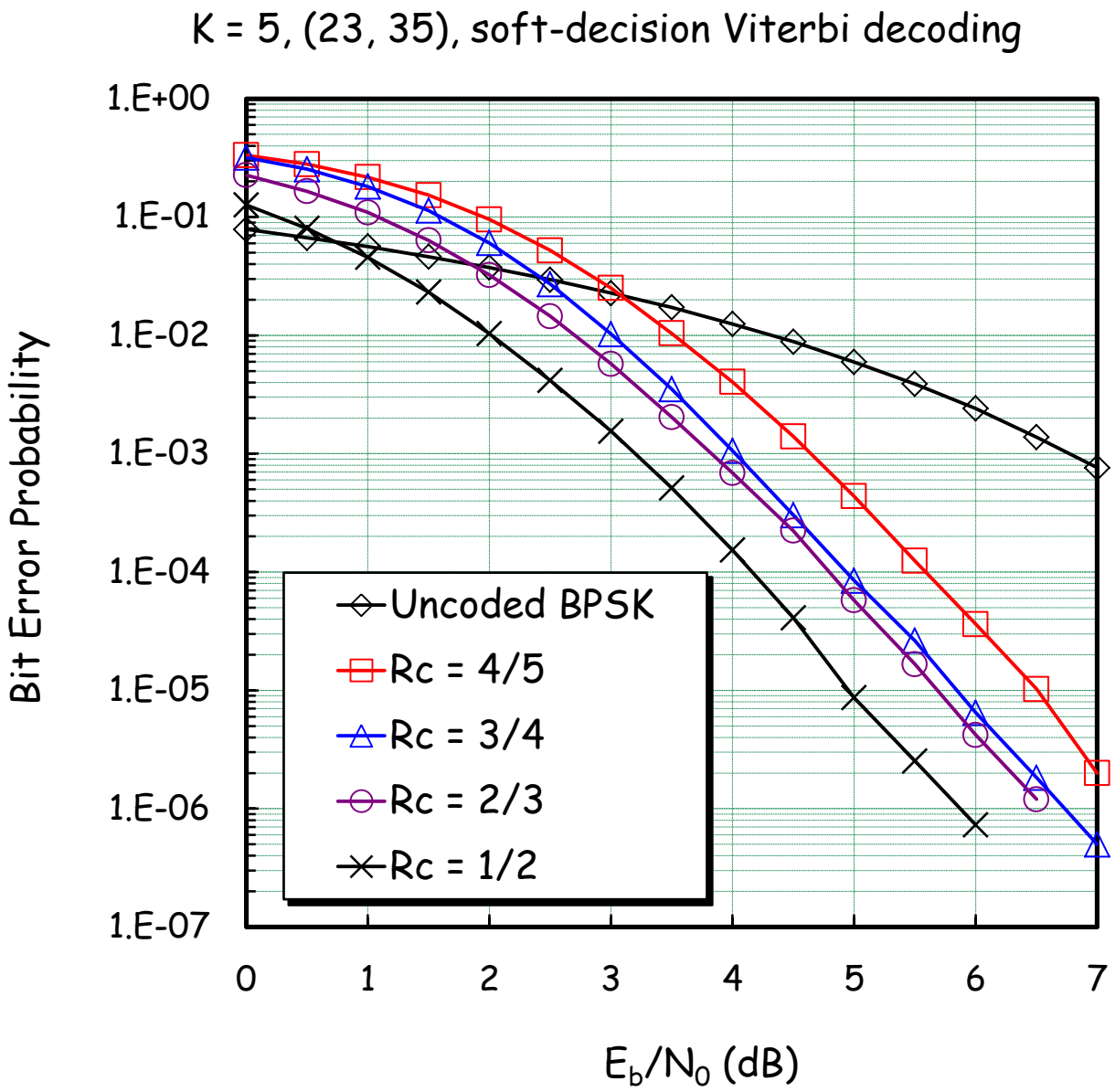
Rate-2/3 CC obtained using $P = \begin{bmatrix} 1110 \\ 1011 \end{bmatrix}$

Rate-3/4 CC obtained using $P = \begin{bmatrix} 110 \\ 101 \end{bmatrix}$

Rate-4/5 CC obtained using $P = \begin{bmatrix} 10101101 \\ 11011010 \end{bmatrix}$


K = 3, (5, 7), soft-decision Viterbi decoding
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
0
1
2
3
4
5
6
7
Eb/N0 (dB)
Uncoded BPSK
Rc = 4/5
Rc = 3/4
Rc = 2/3
Rc = 1/2

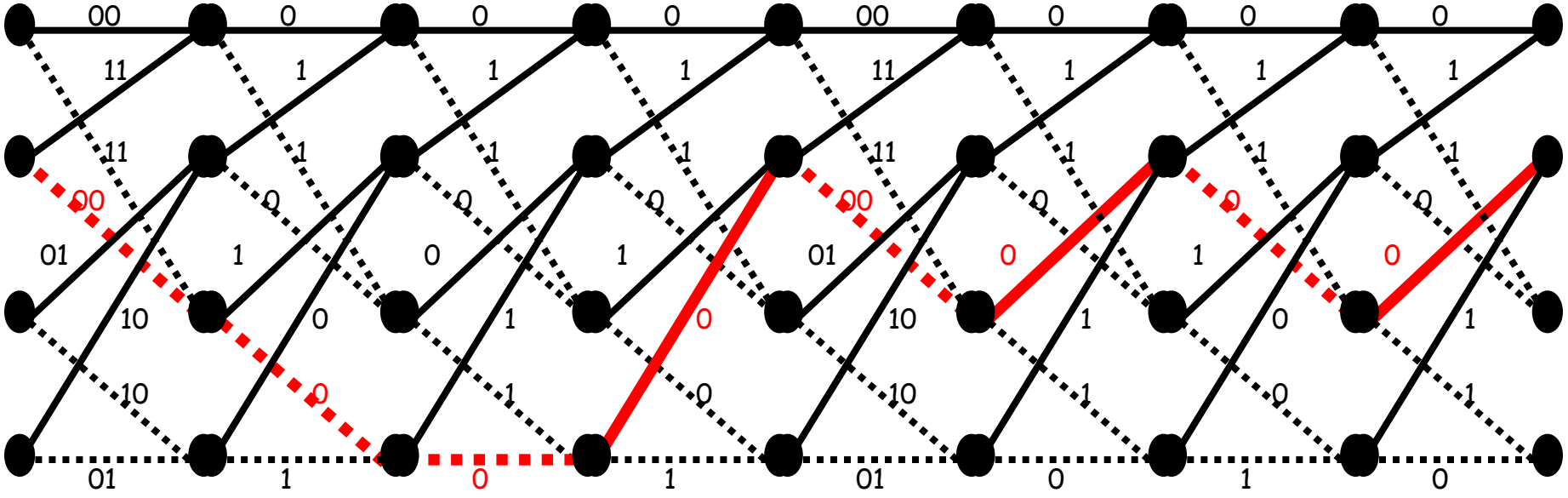

K = 4, (15, 17), soft-decision Viterbi decoding
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
0
1
2
3
4
5
6
7
Eb/N0 (dB)
Uncoded BPSK
Rc = 4/5
Rc = 3/4
Rc = 2/3
Rc = 1/2


K = 5, (23, 35), soft-decision Viterbi decoding
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
0
1
2
3
4
5
6
7
Eb/N0 (dB)
Uncoded BPSK
Rc = 4/5
Rc = 3/4
Rc = 2/3
Rc = 1/2

Punctured convolutional codes

What is wrong with the (5, 7), rate-4/5 code ?

Consider a trellis section over the length of the puncturing pattern $P = \begin{bmatrix} 10101101 \\ 11011010 \end{bmatrix}$



Punctured convolutional codes

Through the trellis, there is a weigth-0 path.

In other words, a message with infinite weight can generate a codeword with finite weight. This should never happen with a convolutional encoder.

The code is said to be catastrophic.

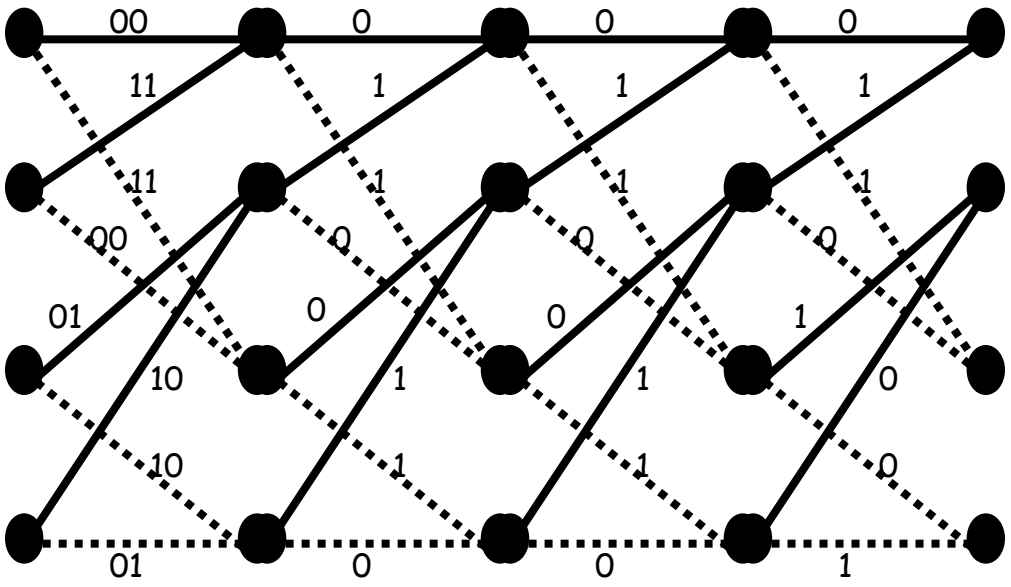
Try the following puncturing pattern instead:

$$P = \begin{bmatrix} 1110 \\ 1001 \end{bmatrix}$$

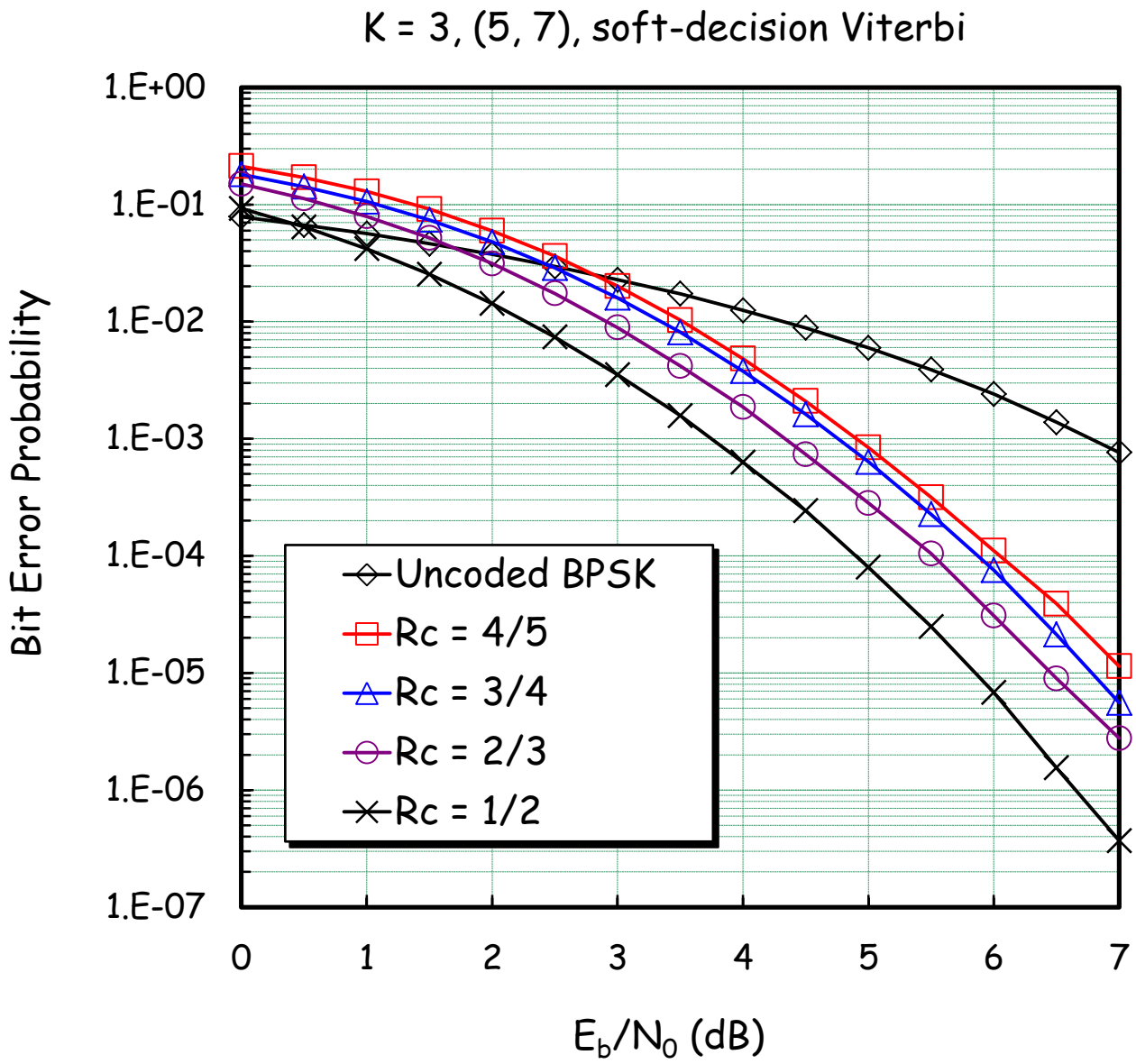
Punctured convolutional codes

There is no longer any weigth-0 path through the trellis.

$$P = \begin{bmatrix} 1110 \\ 1001 \end{bmatrix}$$



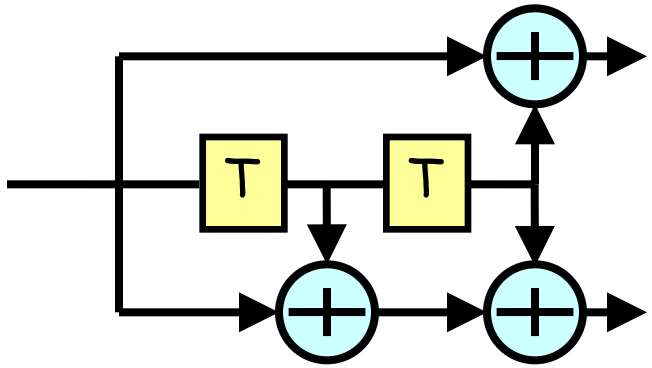
Design the puncturing pattern carefully to guarantee good error performance.


K = 3, (5, 7), soft-decision Viterbi
Bit Error Probability
1.E+00
1.E-01
1.E-02
1.E-03
1.E-04
1.E-05
1.E-06
1.E-07
0
1
2
3
4
5
6
7
Eb/N0 (dB)
Uncoded BPSK
Rc = 4/5
Rc = 3/4
Rc = 2/3
Rc = 1/2

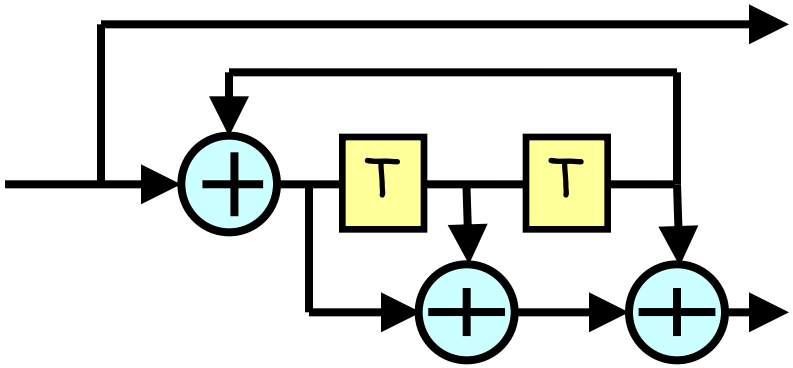
Recursive Systematic Convolutional codes

RSC codes are convolutional codes with interesting properties.

Non-recursive convolutional (NRC) encoder



Equivalent RSC encoder



RSC codes are important because they are used in the design of near-capacity codes called turbo codes.

Recursive Systematic Convolutional codes

Systematic codes, whether recursive or not, cannot be catastrophic.

Systematic codes, whether recursive or not, often perform better than non-systematic codes at low SNR. This is an interesting property, especially when trying to operate close to the capacity limit…

Non-recursive systematic codes are often outperformed at high SNR by their non-systematic equivalent codes, due to their smaller free distance (in general).

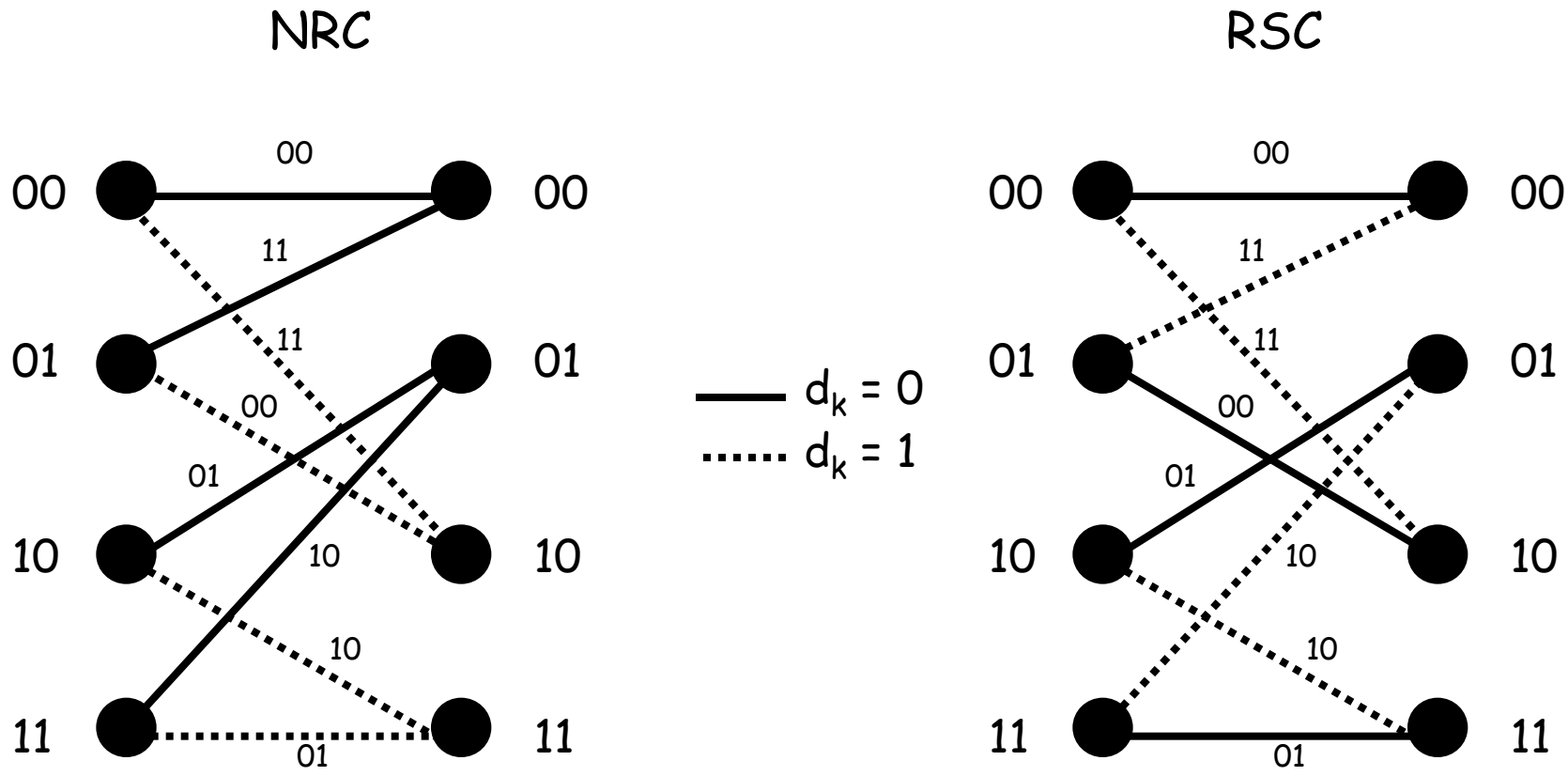
Recursive Systematic Convolutional codes

This is however not true for RSC codes because these codes, despite being systematic, are as good as non-systematic codes at high SNR.

The reason is simply that a NRC code and its RSC equivalent have the same free distance.

In summary, RSC codes combine the advantages of both the systematic codes (which is not surprising since they are systematic) and the non-systematic ones.

Recursive Systematic Convolutional codes



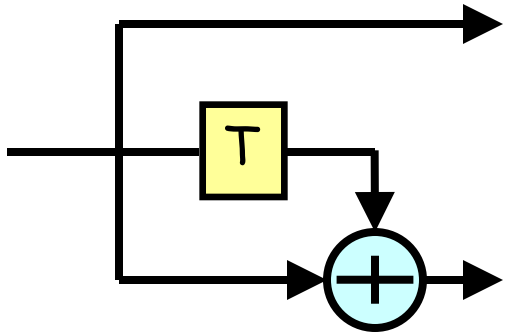
Identical free distance $d_{free} = 5$

Recursive Systematic Convolutional codes

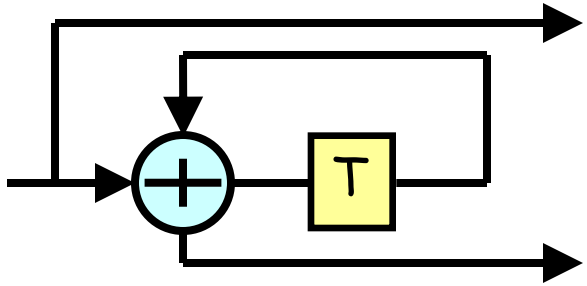
Let us investigate the fundamental difference between a NRC code and its equivalent RSC code.

We focus on the (2, 3) code as it is easier to handle than the (5, 7) code.

(2, 3) NRC encoder

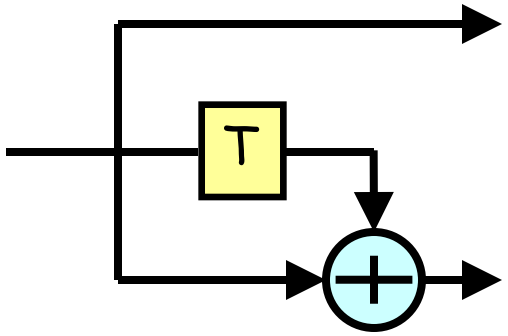


(2, 3) RSC code

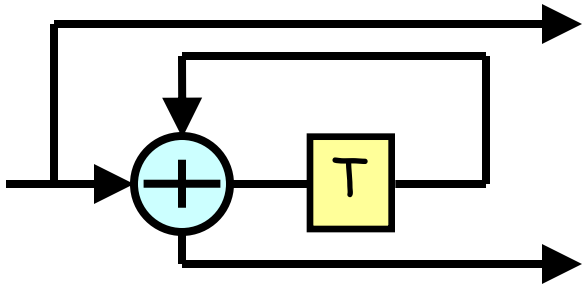


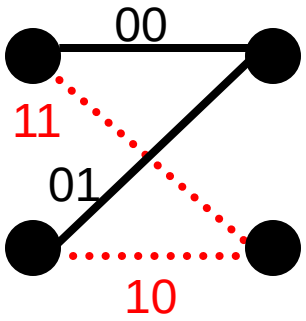
Recursive Systematic Convolutional codes

(2, 3) NRC encoder



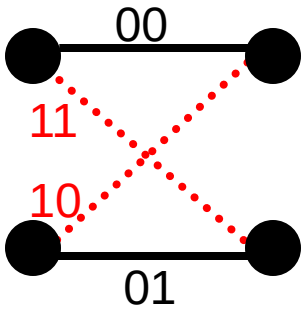
(2, 3) RSC code





Input bit = 0

Input bit = 1



Recursive Systematic Convolutional codes

The trellis of both codes must be terminated in the zero state.

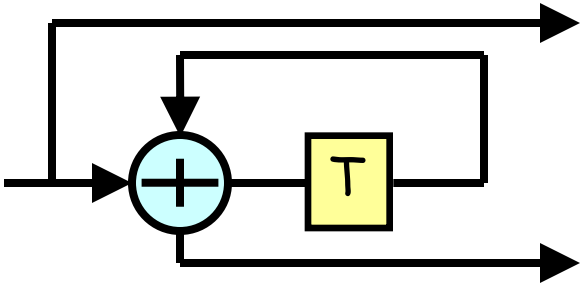
Appending a tail bit at the end of a k-bit message results in a negligible data rate loss since $k \gg 1$ in practice.

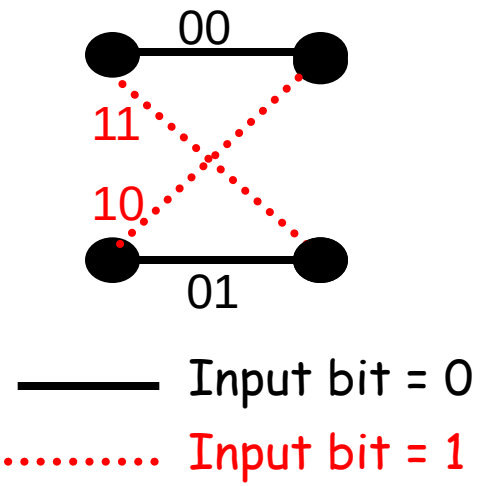
For both encoders, a single tail bit is appended at the end of the message M.

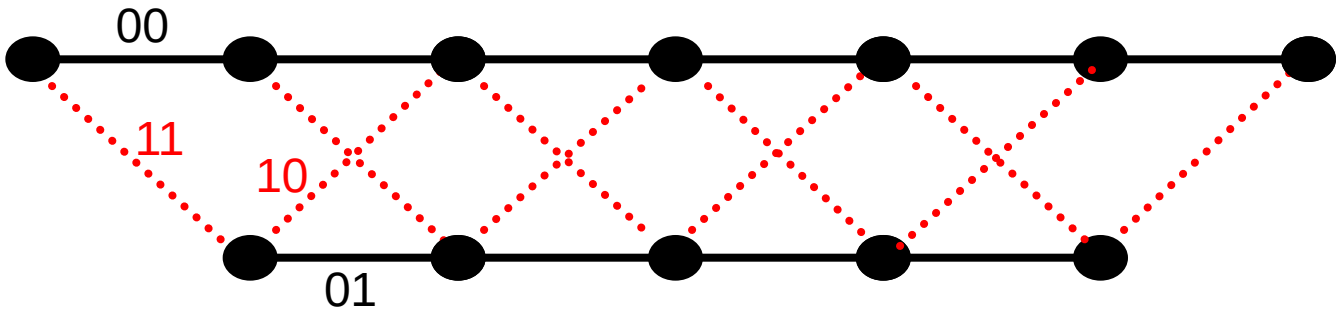
(2, 3) NRC: this tail bit is always equal to 0.

(2, 3) RSC: this tail bit is not necessarily equal to 0.

Recursive Systematic Convolutional codes







M = (100001) → a tail bit equal to 1 is used to return to the zero state.

Recursive Systematic Convolutional codes

In this example, a second 1 is needed in the message to return to the zero state.

As long as the first 1 is followed by 0s, the encoder does not return to the zero state.

Messages with weight-1, 3, 5, 7, etc. cannot exist at the (2, 3) RSC encoder input.

In a terminated NRC encoder, any weight would be acceptable in the message as NRC encoders are terminated by appending a string of 0s as tail bits.

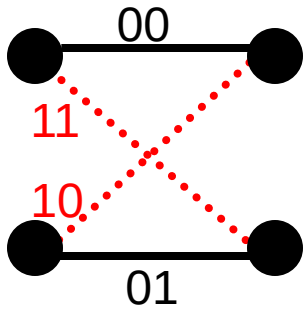
Recursive Systematic Convolutional codes

Consider the simple example of a weight-2 message.

If the two 1s are separated by L 0s, the corresponding codeword is at distance d(L) = 3 + L, L $\in$ {0,…, k-2}, from the all-zero codeword.

Ex: M = (…0 1 0 0 0 0 0 0 0 1 0…) →

C = (…00 11 01 01 01 01 01 01 01 10 00…)



Weight-2 messages can generate codewords at large distance from the all-zero codeword, provided that both 1s are far away from each other.

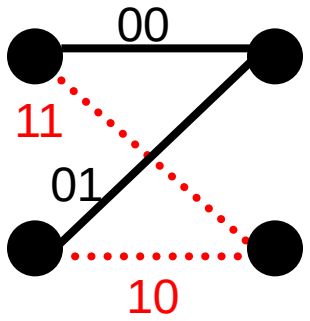
Recursive Systematic Convolutional codes

With an RSC code, a small-weight message often generates a large-weight codeword because, in a long small-weight message, the few 1s are more likely to be isolated rather than grouped together…

An RSC encoder generates low-weight codewords only when the 1s are grouped together.

A NRC code does not exhibit such characteristic as, with such an encoder, a small-weight message always generates a small-weight codeword, regardless of the location of the 1s in the message, as illustrated by the following example.

Recursive Systematic Convolutional codes



Ex: M = (…0 1 0 0 0 0 0 0 0 1 0…) →

C = (…00 11 01 00 00 00 00 00 00 11 01…)

If the two 1s are separated by L 0s, the corresponding codeword is at distance

- d(L) = 4 for L = 0,
- d(L) = 6 for L > 0,

from the all-zero codeword.

Recursive Systematic Convolutional codes

The fundamental difference between NRC and RSC codes is now very clear.

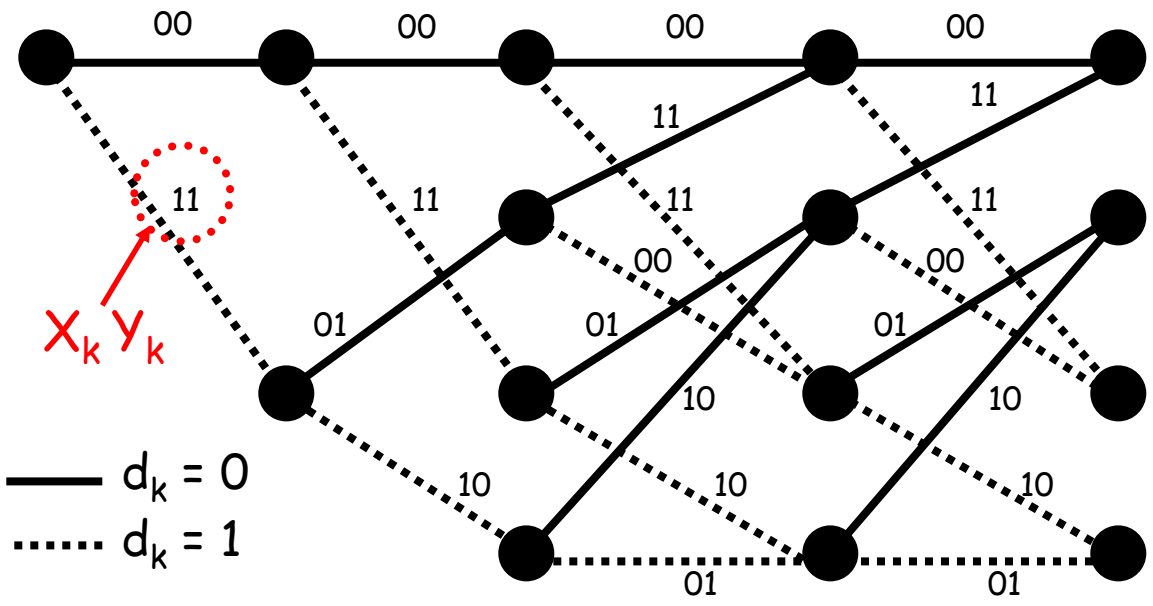
NRC: Low-weight messages always produce low-weight codewords.

RSC: Low-weight messages often produce large-weight codewords. They generate low-weight codewords only when the 1s are grouped in the message.

The latter property partially explains the near-capacity performance of turbo codes which are built by concatenating two RSC codes.

Viterbi decoding algorithm (1967)

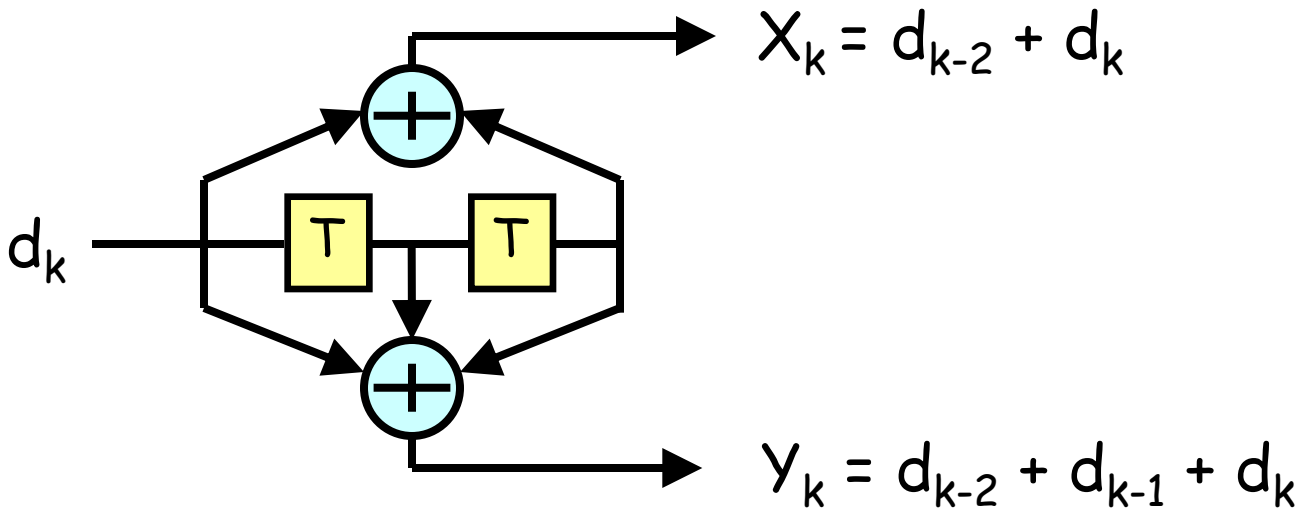
1967: Andrew J Viterbi introduces a practical ML decoding algorithm, initially intended for CCs (the celebrated "Viterbi algorithm").



Trellis representation of the $R_c = \frac{1}{2}$, K = 3, (5, 7) CC, used for Viterbi decoding

Viterbi decoding algorithm

To explain the Viterbi decoding algorithm, let us consider the 4-state rate-1/2 CC previously studied.



Assume the info sequence $\{d_k\} = \{110100\}$, including two tail bits used to return to state 00 at the end of the encoding process.

Viterbi decoding algorithm

The Viterbi decoding algorithm exploits the trellis structure of convolutional codes in order to achieve ML decoding with reasonable complexity, no matter how long the message is.

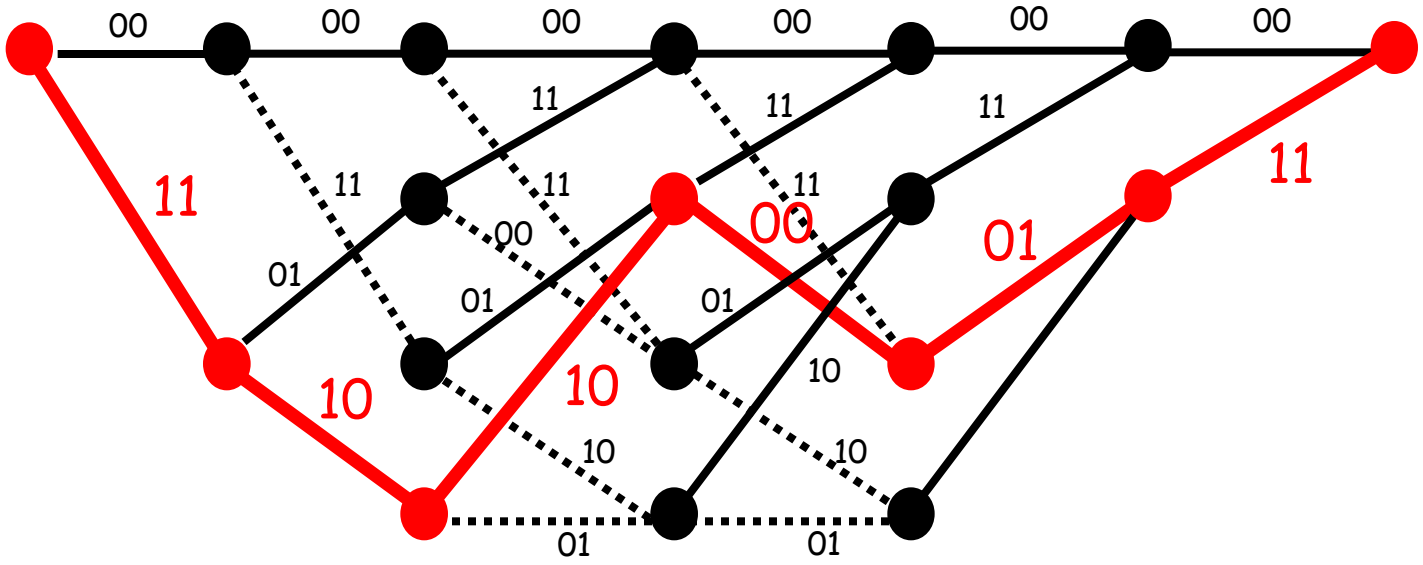
This algorithm has been used for decades in many applications beyond the field of error-correcting codes, almost every time the operation of a given system could be described by a trellis.

Let us come back to our example.

Viterbi decoding algorithm

The coded sequence corresponding to the message M $\{d_k\}$ = {110100} is $\{X_k Y_k\}$ = {11 10 10 00 01 11}.

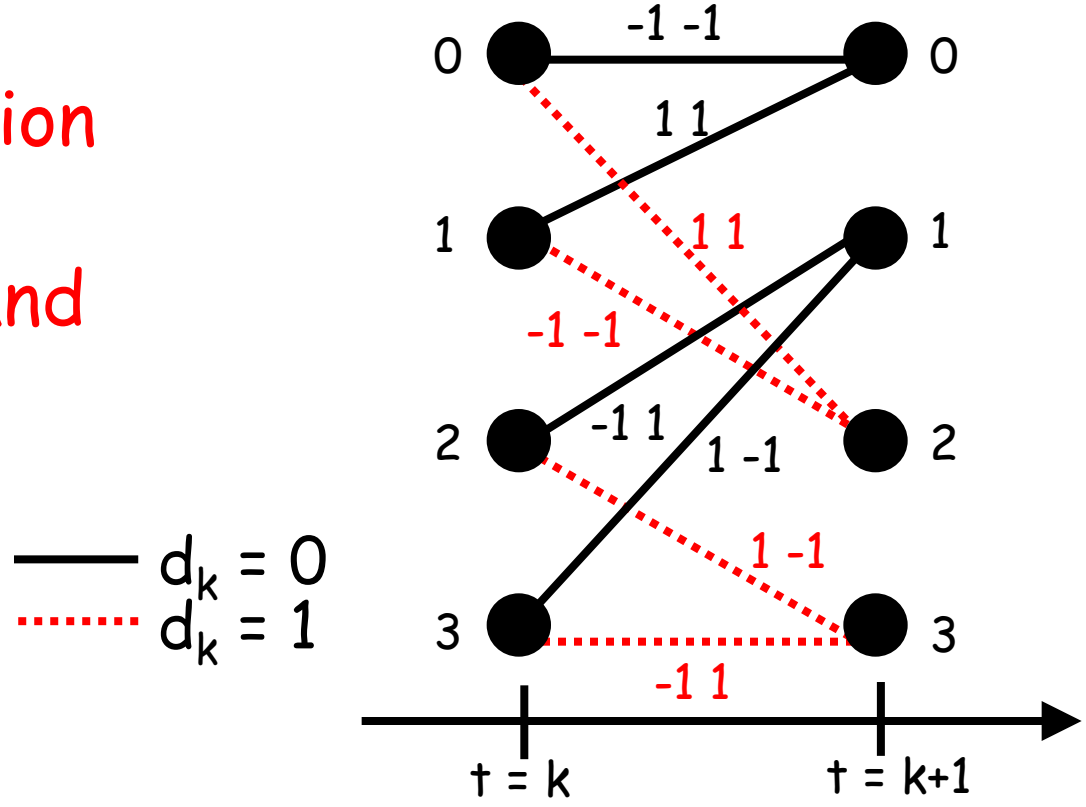
This codeword is represented by the red path in the trellis of the code.



Viterbi decoding algorithm

We use the soft-decision Viterbi decoding algorithm, i.e. assume transmission over a BPSK, AWGN channel.

The BPSK modulation scheme is defined such that 0 → -1 and 1 → +1.



Viterbi decoding algorithm

The Viterbi algorithm is an ML decoding algorithm: it searches, among all possible codewords/paths in the trellis, for the sequence (codeword) which is at minimal Euclidean distance from the received vector $R = (r_0, r_1, r_2, \ldots, r_{n-1})$ of channel samples.

In other words, the Viterbi algorithm searches for the codeword which minimizes the term $\sum_{l=0}^{n-1}\left(r_l - c_{i,l}\right)^2$, where r_l is the $(l+1)$-th received channel sample and $c_{i,l}$ is the $(l+1)$-th bit in a particular codeword C_i.

Here, we have $n = 12$ since $k = 6$.

Viterbi decoding algorithm

Equivalently, the algorithm searches for the sequence C_i which maximizes the correlation term $\gamma(R, C_i) = \sum_{l=0}^{n-1} r_l \cdot c_{i,l}$.

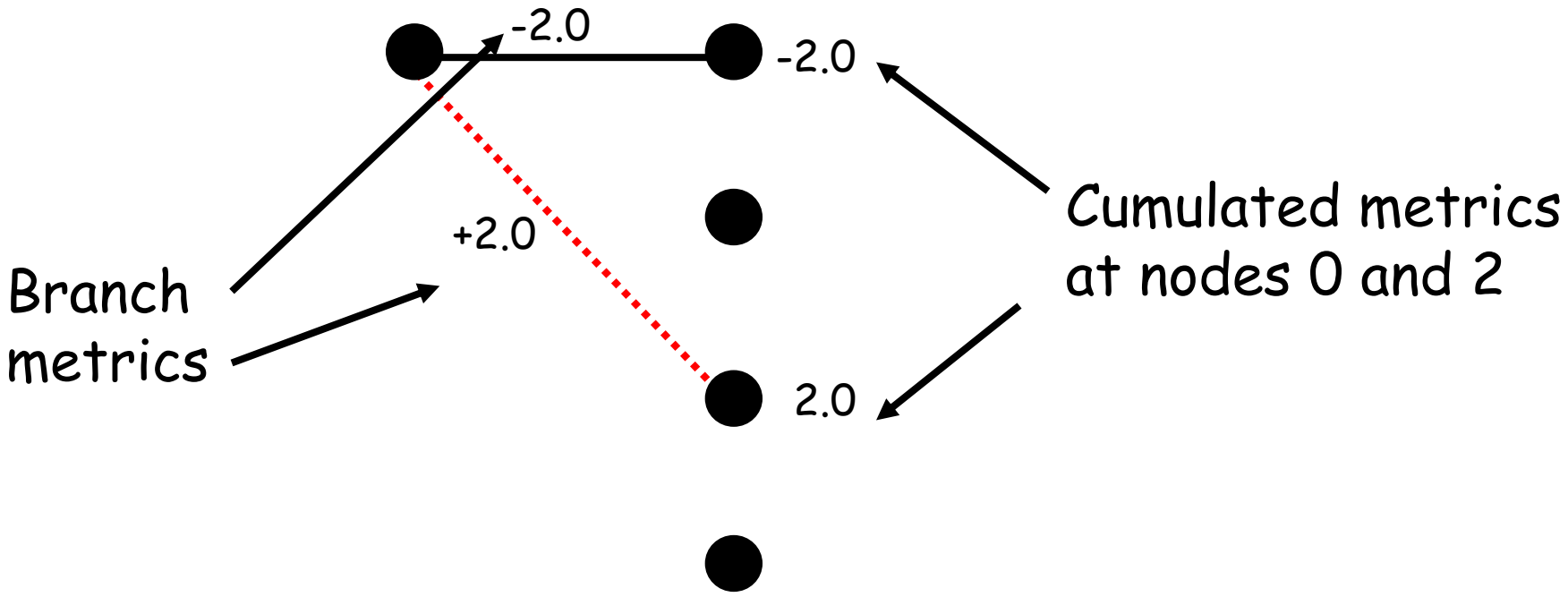
Assume that the channel output is
$R = (+1.3, +0.7, +0.2, -0.2, +0.9, -0.9, -2.2, +0.2, \ldots$
$\ldots -2.5, -0.5, +0.1, +0.3)$.

If we used a decision block before decoding, it would correspond to the received binary sequence {11 10 10 01 00 11}. There would be two errors in this sequence.

Two transmission errors

Viterbi decoding algorithm

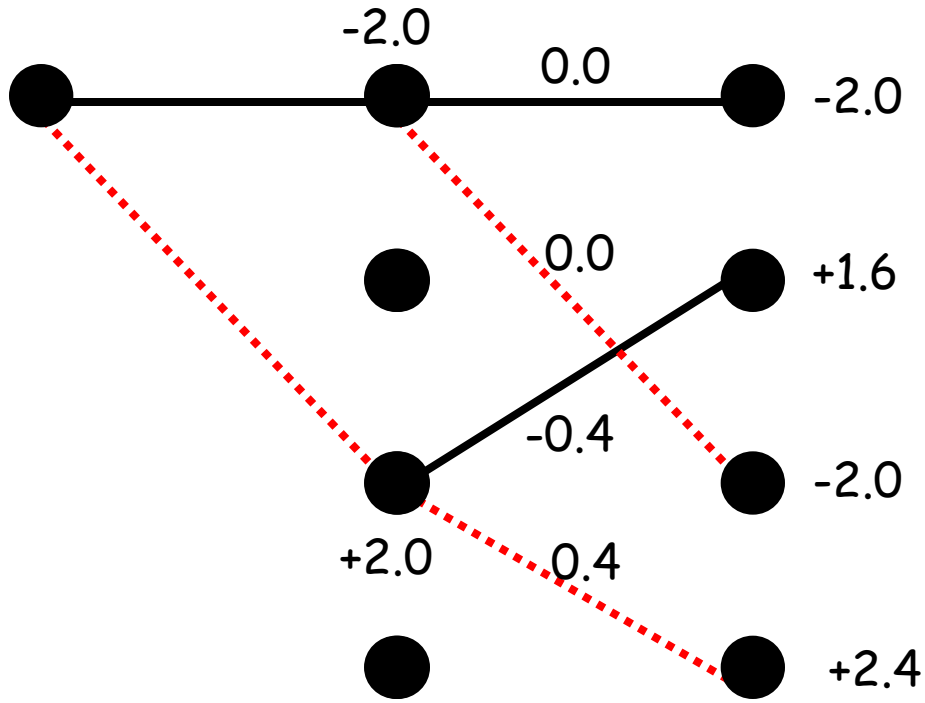
We start in the zero state with the 1st received pair $\{r_0 = +1.3, r_1 = +0.7\}$. We compute the term $(r_0.c_{i,0} + r_1.c_{i,1})$ for the two possible branches.



Viterbi decoding algorithm

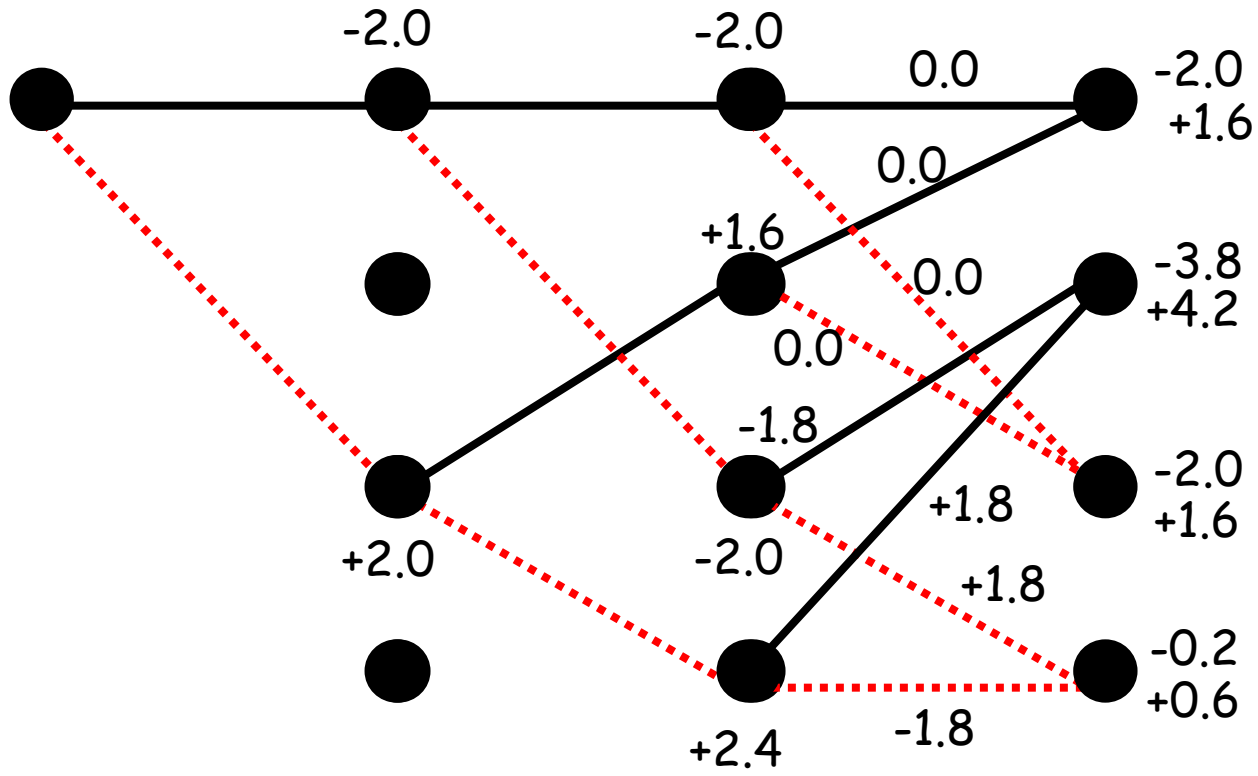
We now consider the 2nd pair $\{r_2 = +0.2, r_3 = -0.2\}$. We compute the term $(r_2.c_{i,2} + r_3.c_{i,3})$ for the four possible branches (4 branch metrics).

We compute the cumulated metric for each node by adding the branch metric to the cumulated metric of the previous node.



Viterbi decoding algorithm

We now consider the 3rd pair $\{r_4 = +0.9, r_5 = -0.9\}$. We compute the term $(r_4.c_{i,4} + r_5.c_{i,5})$ for the eight possible branches (8 branch metrics).



Viterbi decoding algorithm

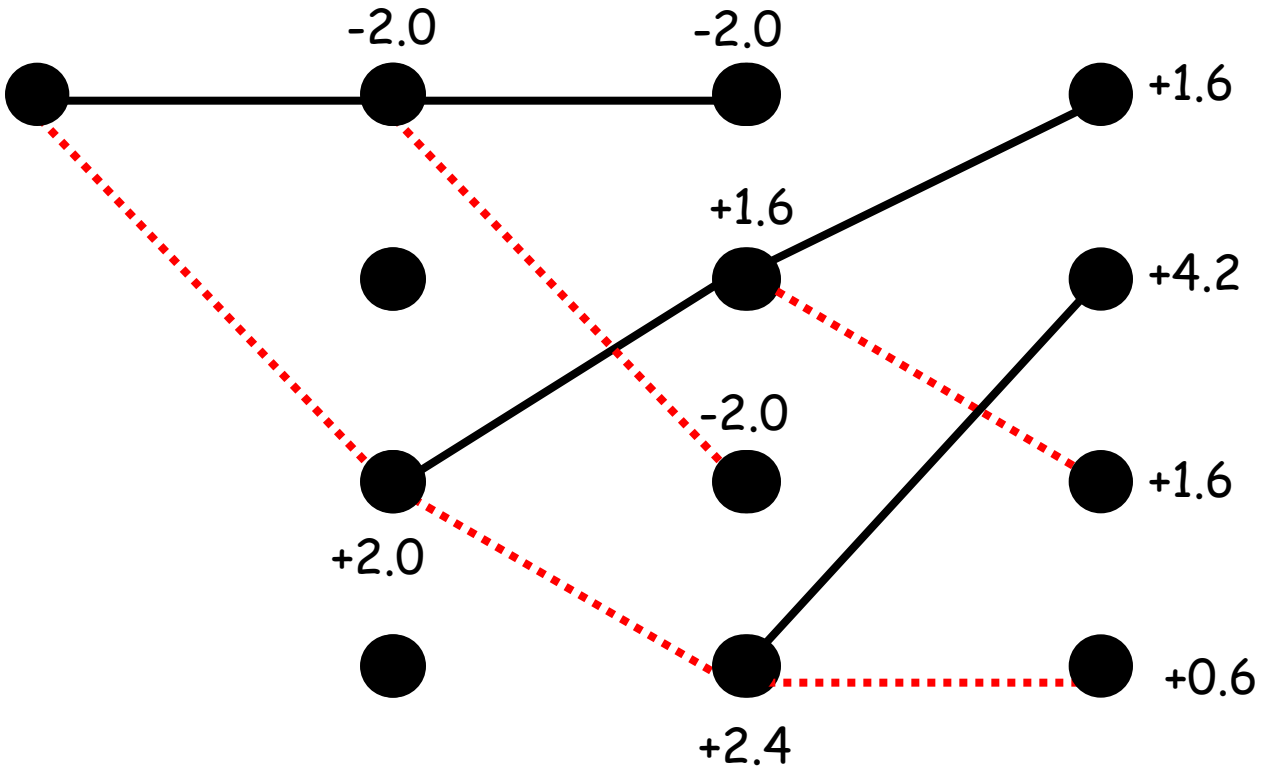
There are two different paths leading to each node.

We only need to keep one of them because we know that, no matter what happens next, one of these paths cannot be the selected path at the end of the process.

Out of the eight paths obtained so far, we can already discard four of them. By doing so, we prevent the number of possible paths from growing exponentially at each decoding step.

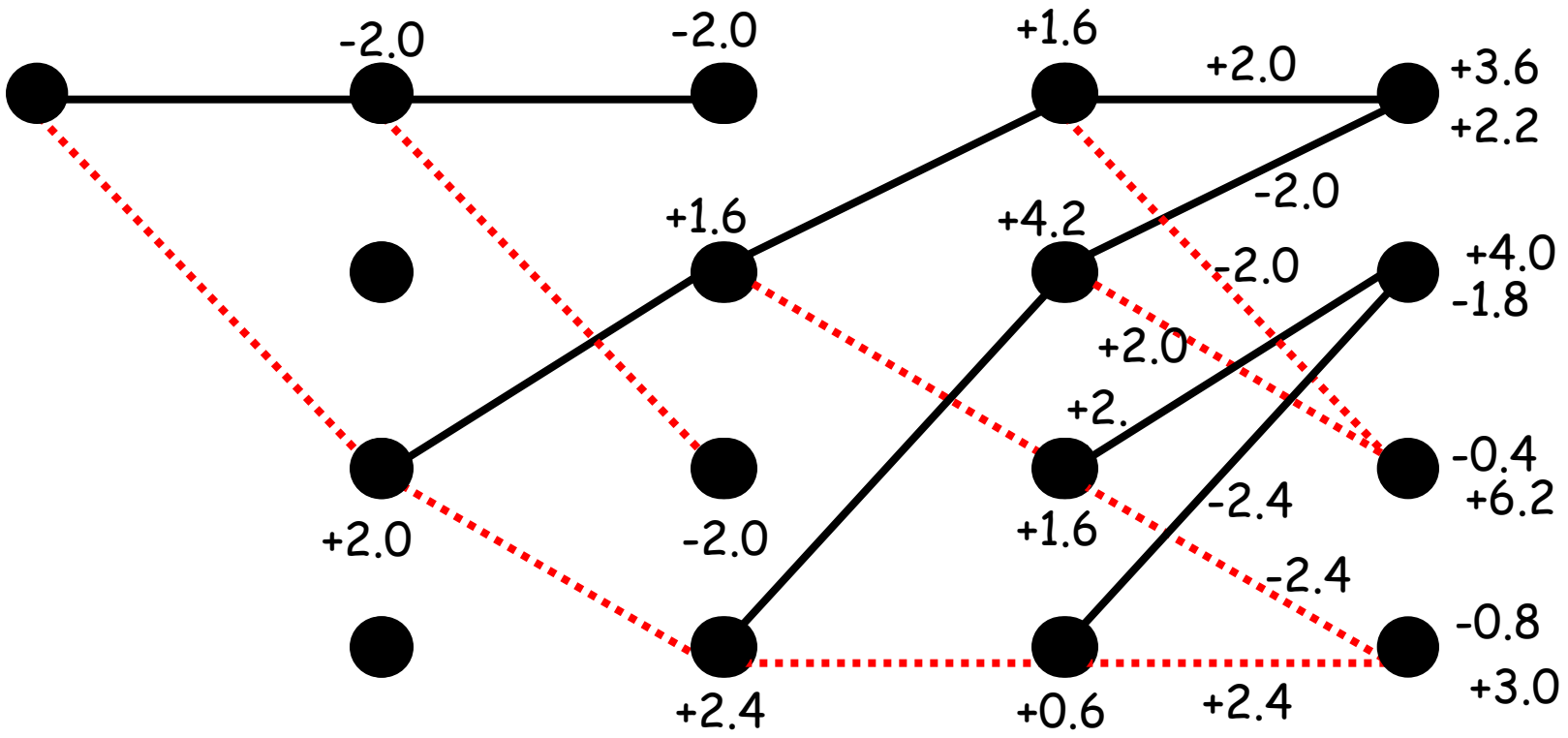
Viterbi decoding algorithm

For each node, we only keep the branch with the highest cumulated metric, and we repeat the process until the whole received sequence is processed.



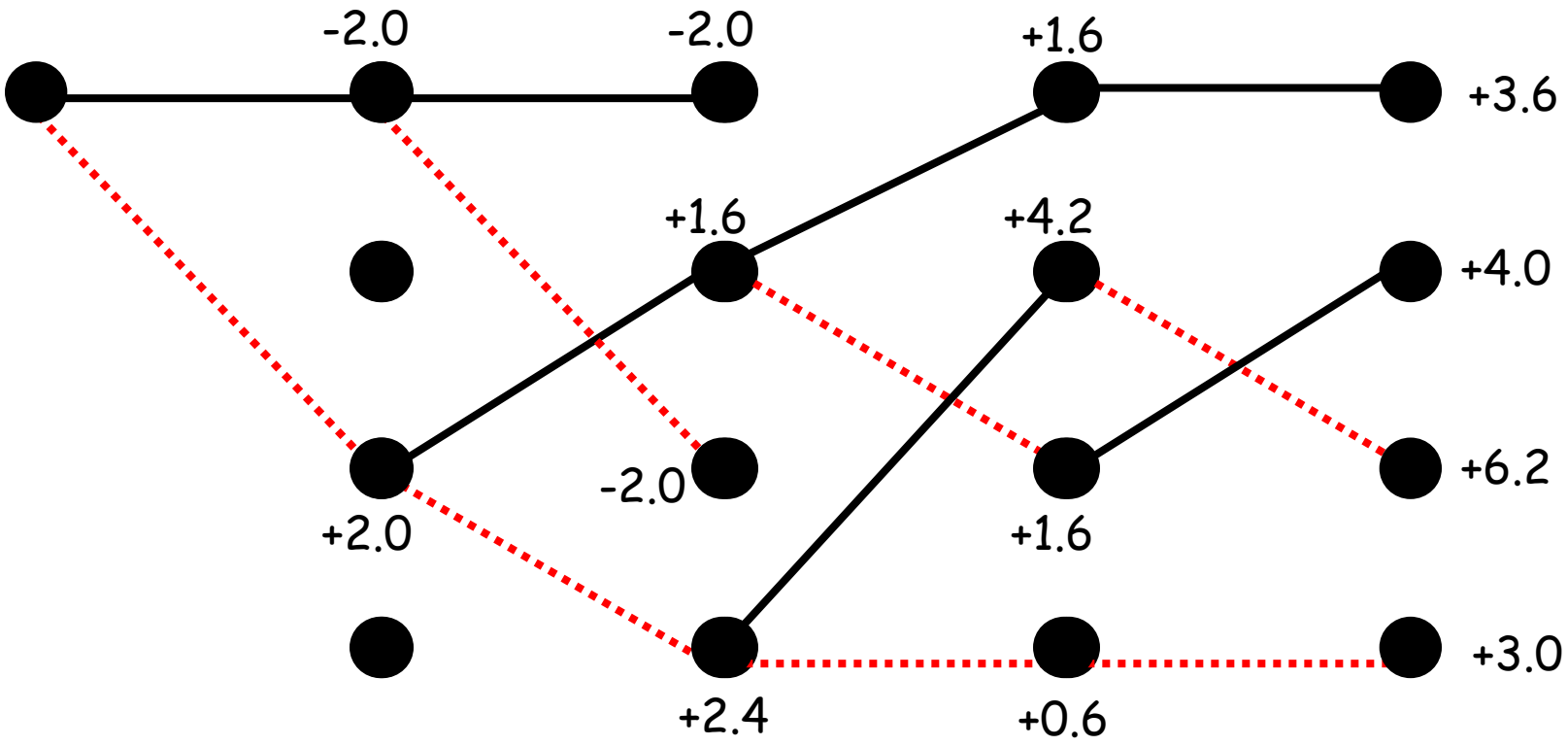
Viterbi decoding algorithm

We now consider the 4th pair $\{r_6 = -2.2, r_7 = 0.2\}$.



Viterbi decoding algorithm

At every decoding step, half of the possible codewords are discarded so that the number of possible codewords remains equal to four.



Viterbi decoding algorithm

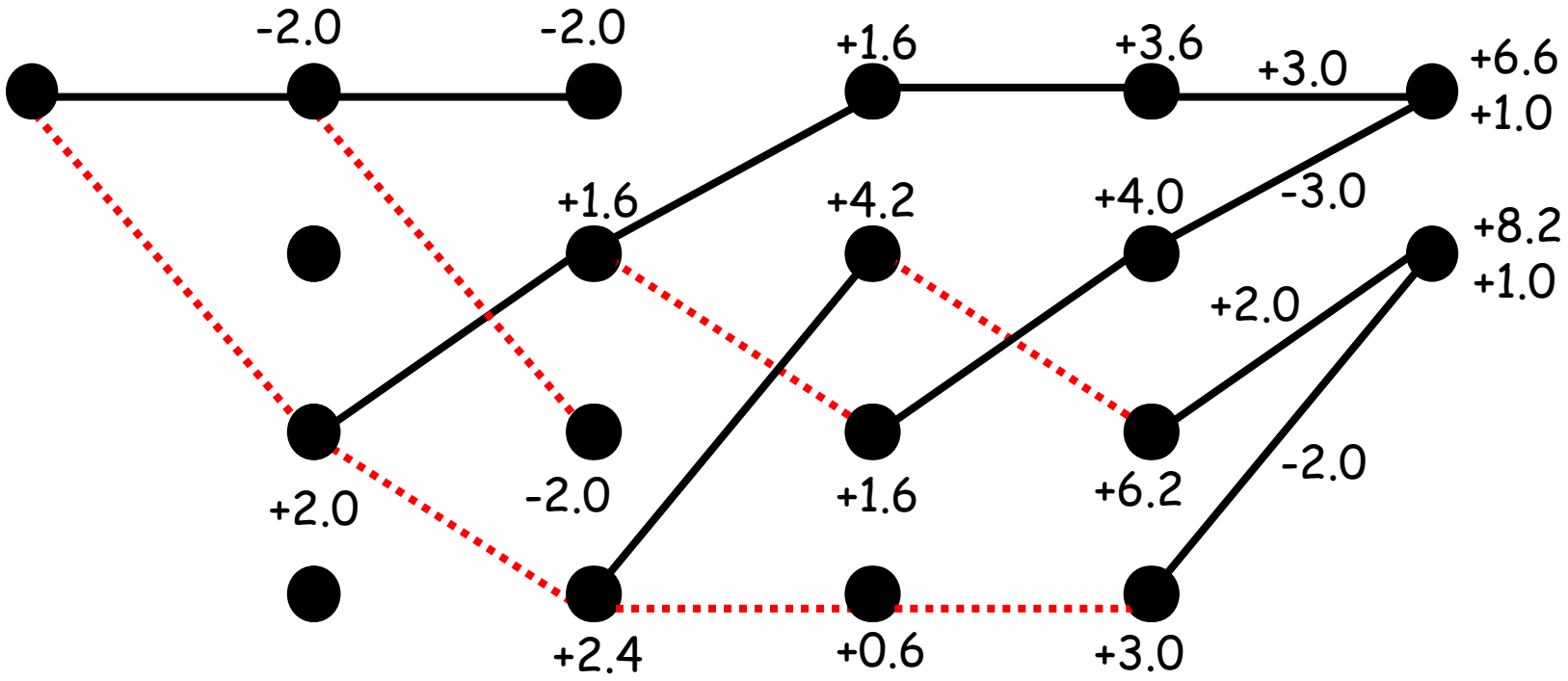
The Viterbi algorithm is an ML decoding algorithm and its use therefore leads to optimal error performance at the decoder output.

The number of codewords to consider never becomes prohibitive since, at every step of the decoding process, it is possible to eliminate half of the remaining codewords with the certainty that the ML codeword is being kept.

This procedure is made possible only thanks to the trellis structure of the encoder.

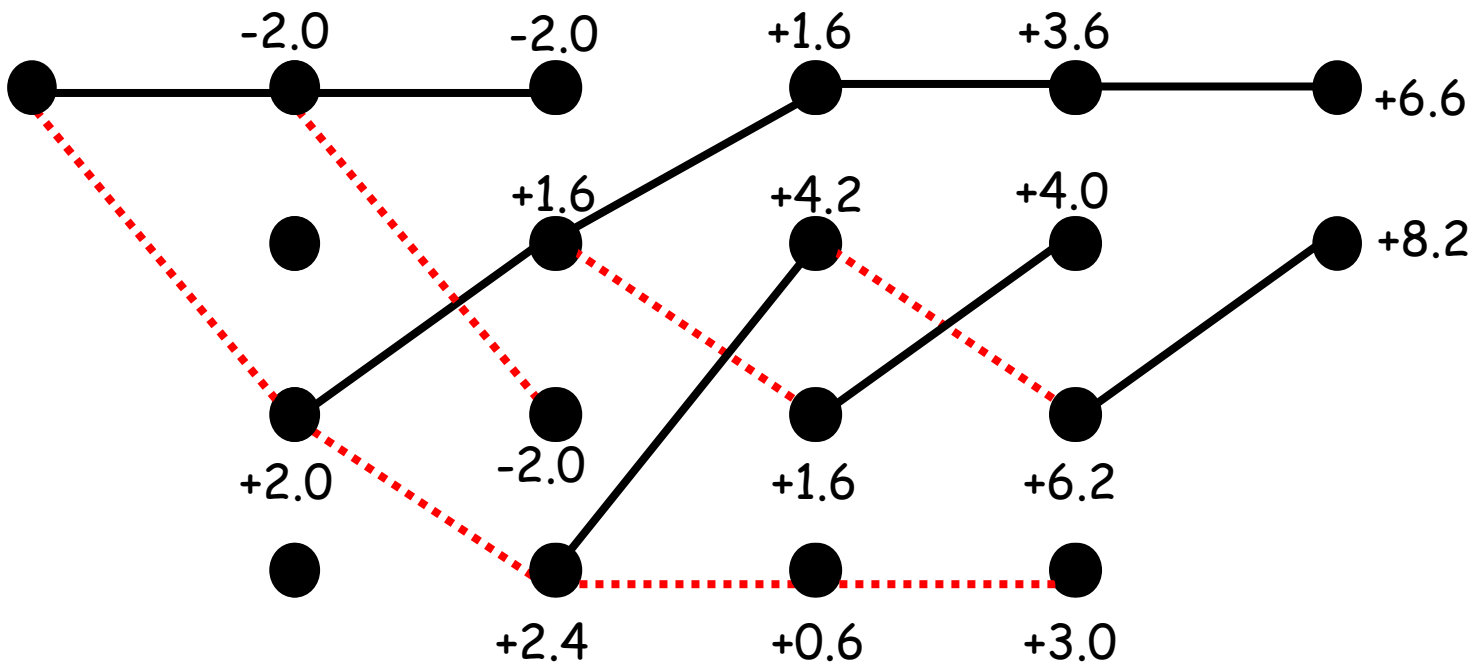
Viterbi decoding algorithm

We consider the 5th pair $\{r_8 = -2.5, r_9 = -0.5\}$.



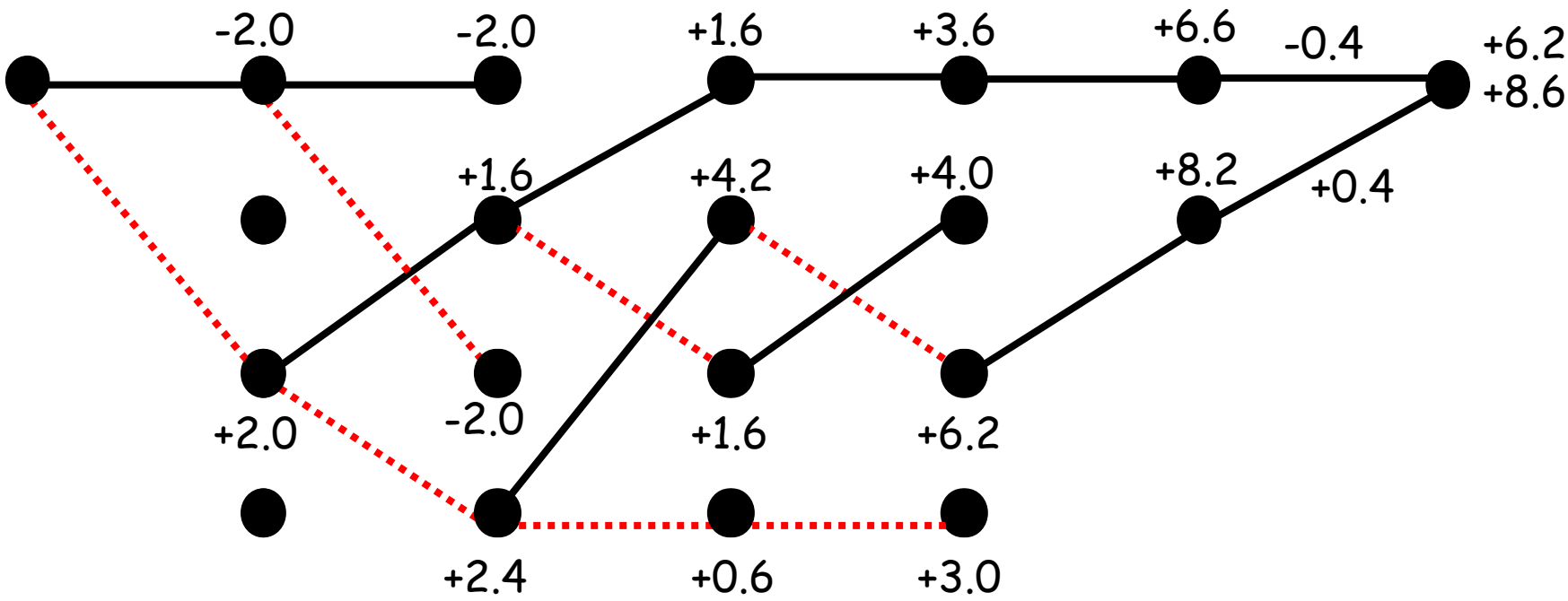
Viterbi decoding algorithm

We are now left with only two possible codewords, with one more final step to go.



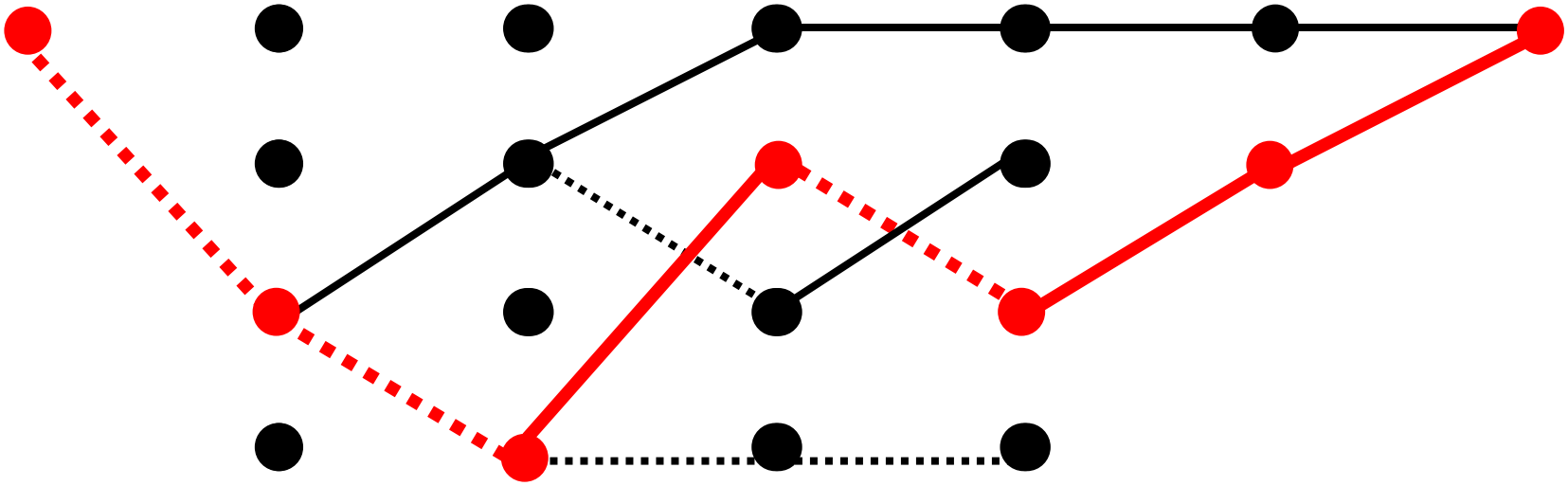
Viterbi decoding algorithm

We consider the 6th pair $\{r_{10} = +0.1, r_{11} = +0.3\}$.



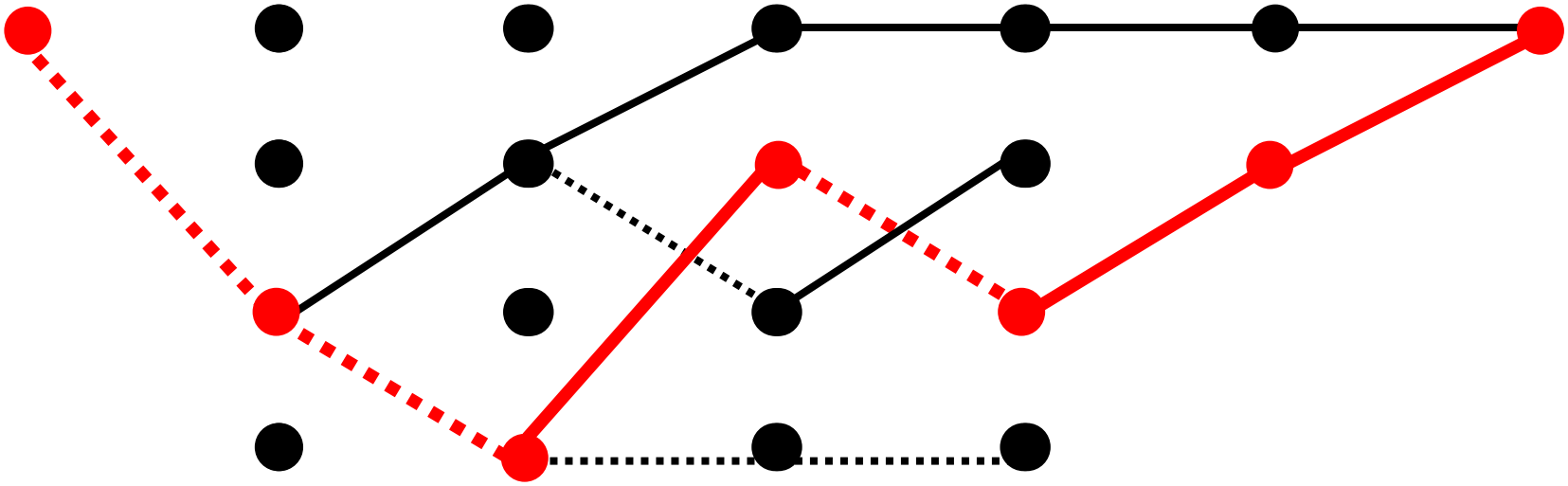
Viterbi decoding algorithm

We finally select the path/sequence/codeword having the largest cumulated metric among the last two remaining paths. The winner codeword is shown in red below.



Viterbi decoding algorithm

The selected path is associated with the decoded sequences C′ = (11 10 10 00 01 11) and M′ = {110100}.



Viterbi decoding algorithm

In this example, despite both (soft) transmission errors, we are able to determine the right path in the trellis, i.e. recover the transmitted coded sequence {11 10 10 00 01 11}, which corresponds to the info sequence {110100}.

The Viterbi algorithm can be used for decoding any code whose operation is described using a trellis.

It can also process hard decisions (bits) in cases where the channel does not generate soft decisions.

Trellis-coded modulation (1982)

1982: Gottfried Ungerboeck invents trellis-coded modulation, a coding approach suitable for bandwidth-limited channels → Modems of the 80s, initially.

The redundancy needed for correcting errors is achieved by employing more constellation signal points.

Unlike traditional coding techniques, TCM coding does not result in bandwidth expansion.

Trellis-coded modulation

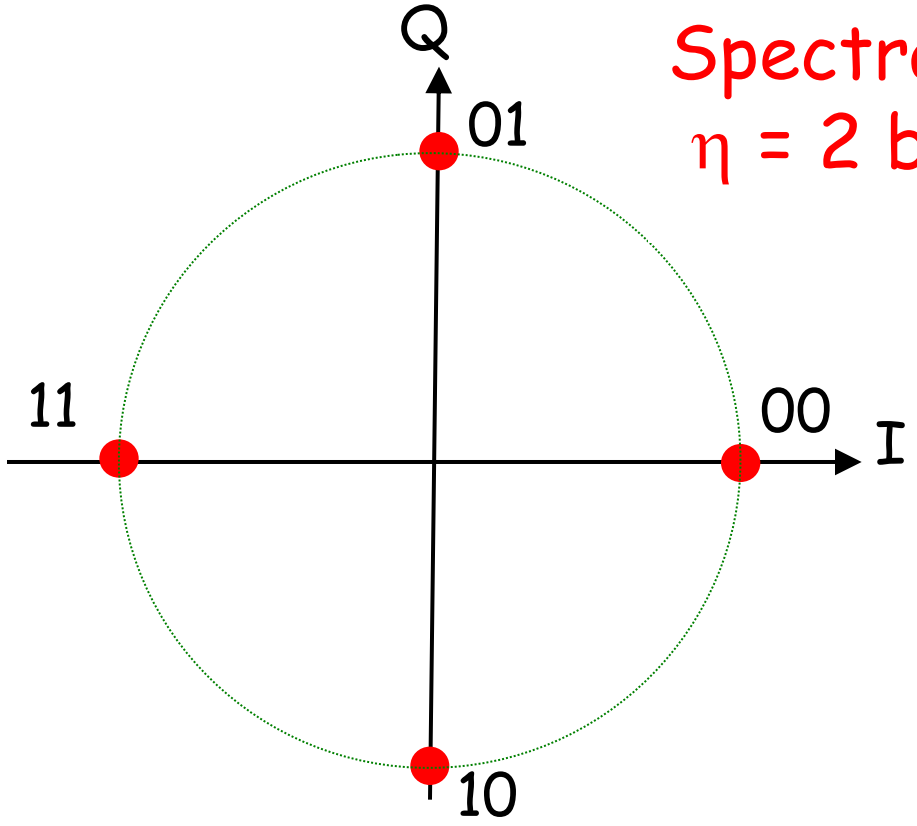
Traditional channel coding techniques expand the bandwidth by a factor n/k (for a fixed info bit rate), which is not acceptable in applications where bandwidth is a precious resource.

For these applications, we need to obtain a coding gain without sacrificing bandwidth.

Ex: If a spectral efficiency of 2 bits/s/Hz is needed, we can use
(1) Uncoded QPSK or
(2) 8PSK + rate-2/3, 4-state convolutional code.

Will (2) outperform (1) in terms of P_{eb}?

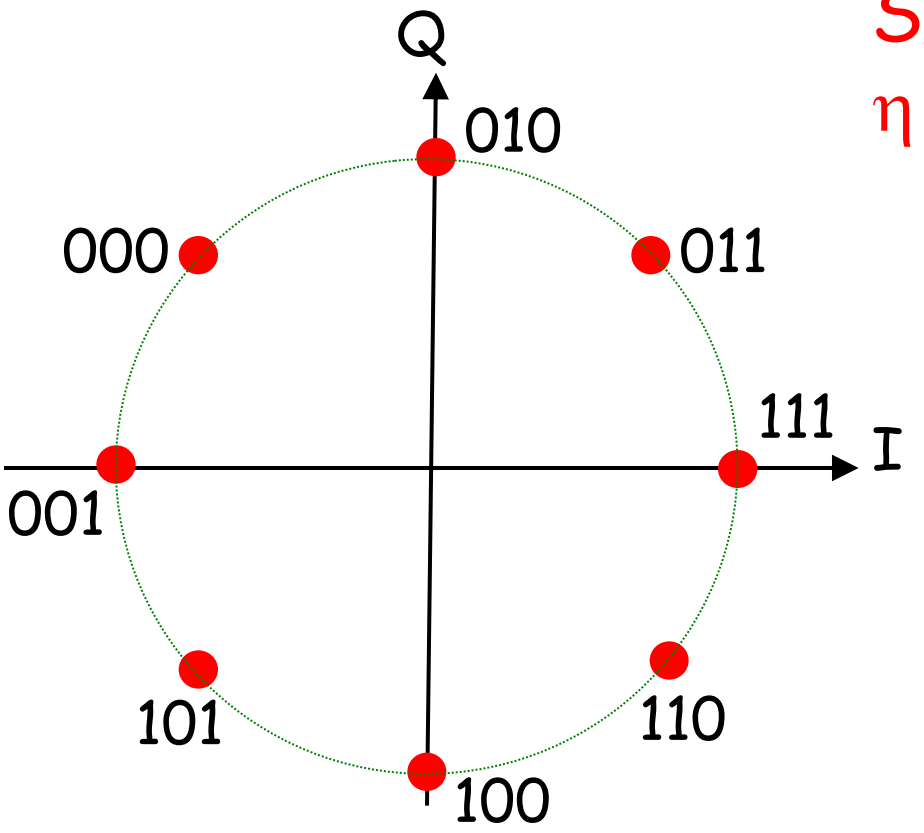
Trellis-coded modulation – An example



Spectral efficiency
η = 2 bits/sec/Hz

QPSK (Gray mapping)

Trellis-coded modulation – An example

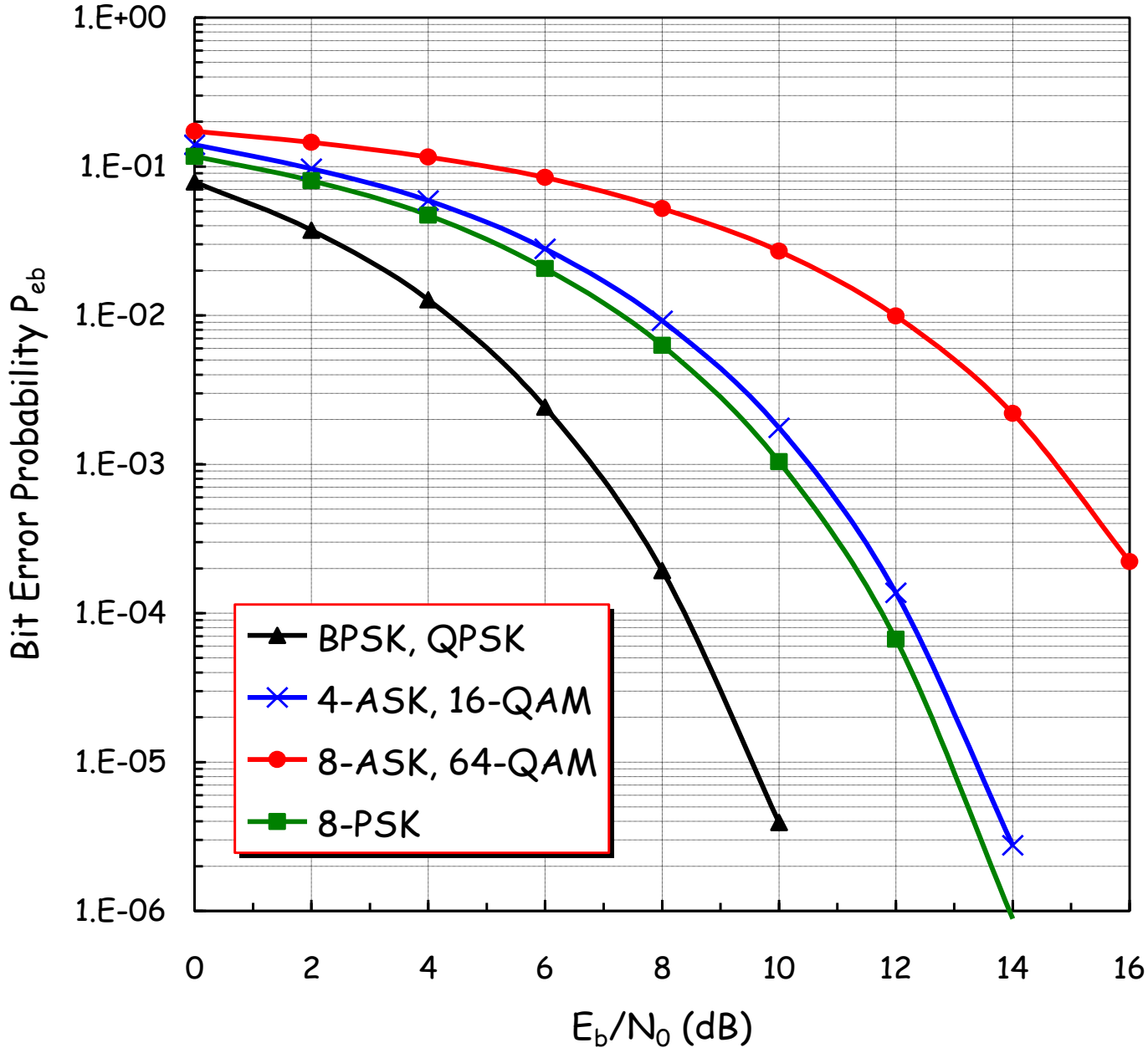


8PSK (Gray mapping)

Spectral efficiency
η = 3 bits/sec/Hz

If the 3rd bit is used as a redundant bit, then the spectral efficiency is reduced to 2 bits/sec/Hz

Do not forget that QPSK is more robust than 8PSK in the absence of coding.



Trellis-coded modulation – An example

Symbol error probability of an uncoded modulation scheme over AWGN channel, at high SNR:

$$P_{es} \approx \frac{N}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{d_0^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

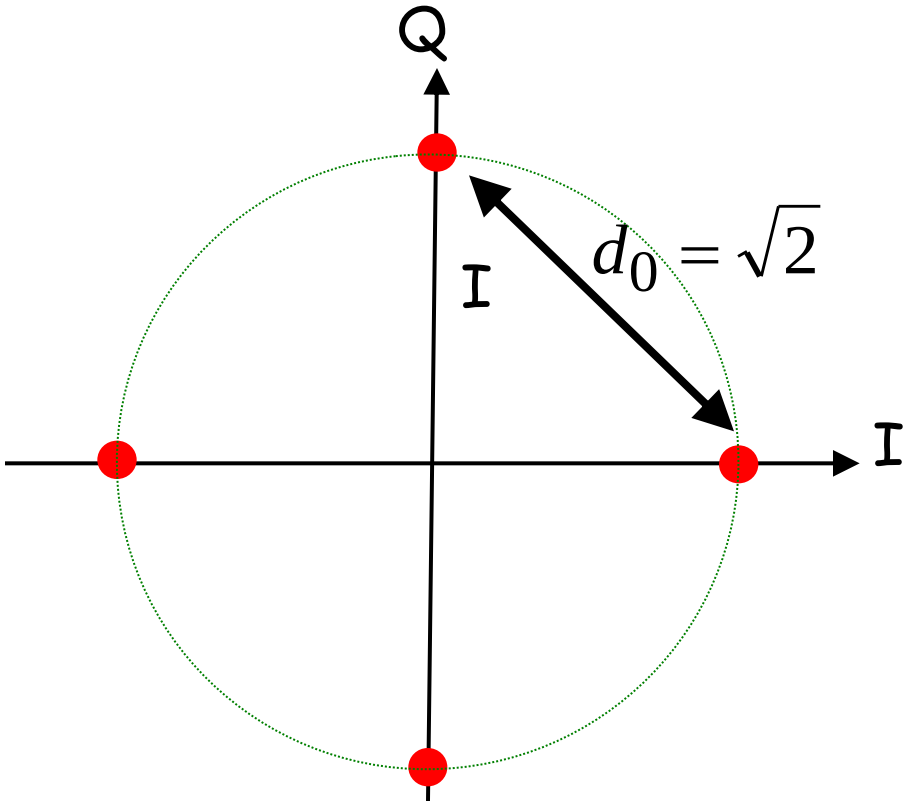
where

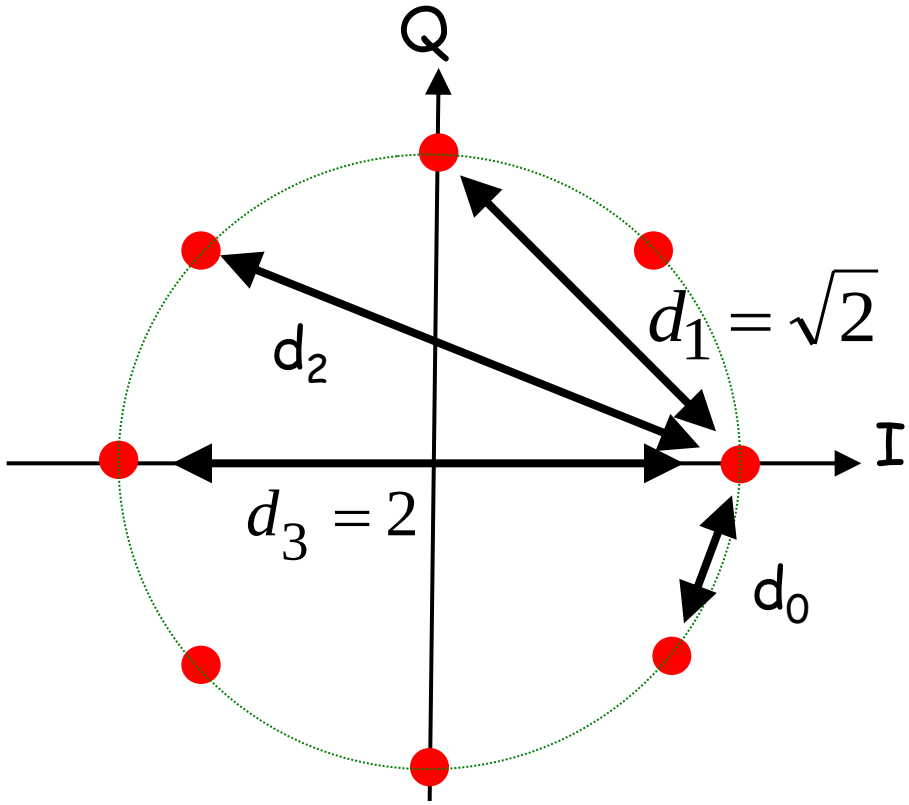
N is the average number of nearest neighbour signal points in the constellation,

E_s/N_0 is the SNR per transmitted signal,

d_0 is the Euclidean distance between two nearest-neighbor signal points in the constellation, under the constraint of unit average energy for these signal points.

Trellis-coded modulation – An example

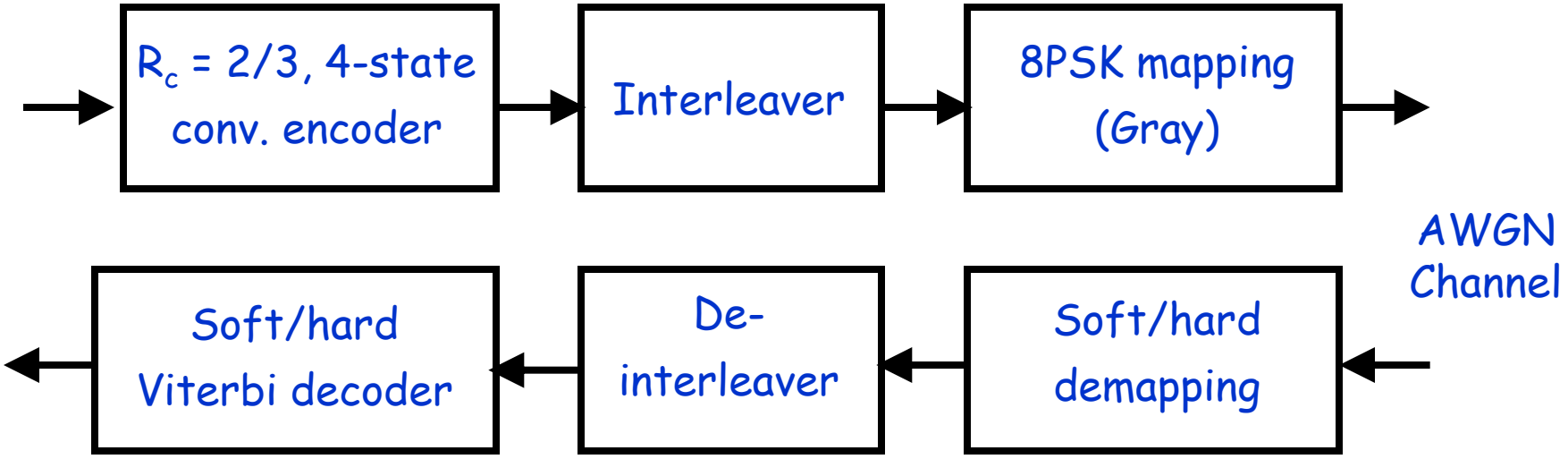




$$d_0 = 2 \cdot \sin\left(\frac{\pi}{8}\right) \qquad d_2 = 2 \cdot \cos\left(\frac{\pi}{8}\right)$$

Trellis-coded modulation – An example

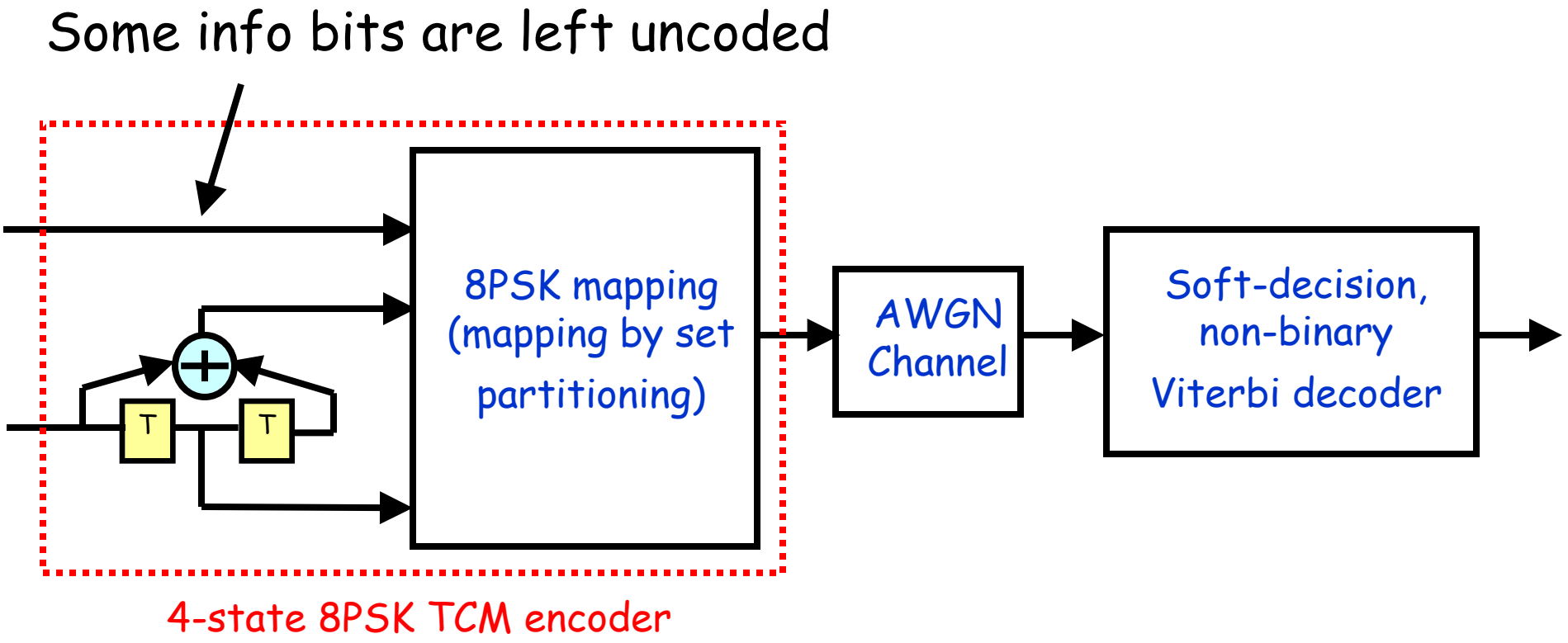
Before 1982: The coding and modulation were seen as two separate entities.



Coding and modulation are optimized separately. When soft-demapping is employed, such approach is known as "bit-interleaved coded modulation" (BICM). Coding gains over uncoded scheme were disappointing.

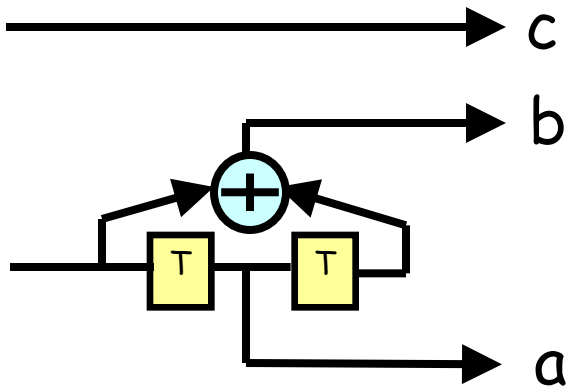
Trellis-coded modulation – An example

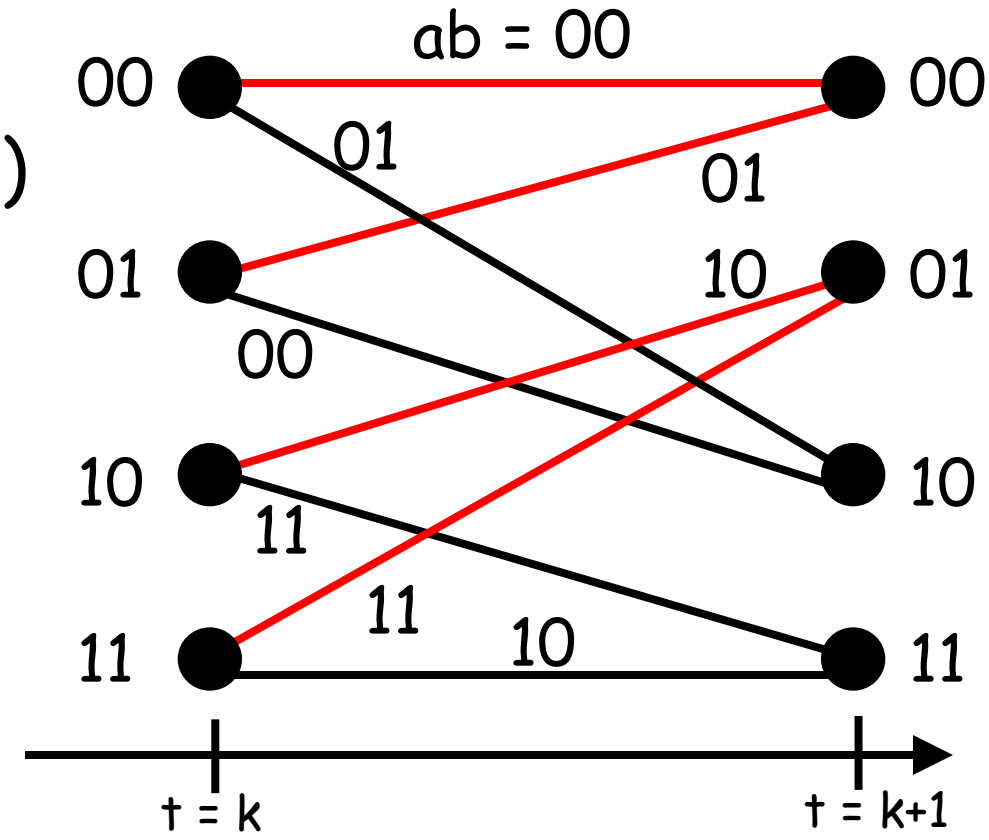
In TCM, coding and modulation are jointly optimized.



Trellis-coded modulation – An example

Binary encoder
(bit c is not encoded)

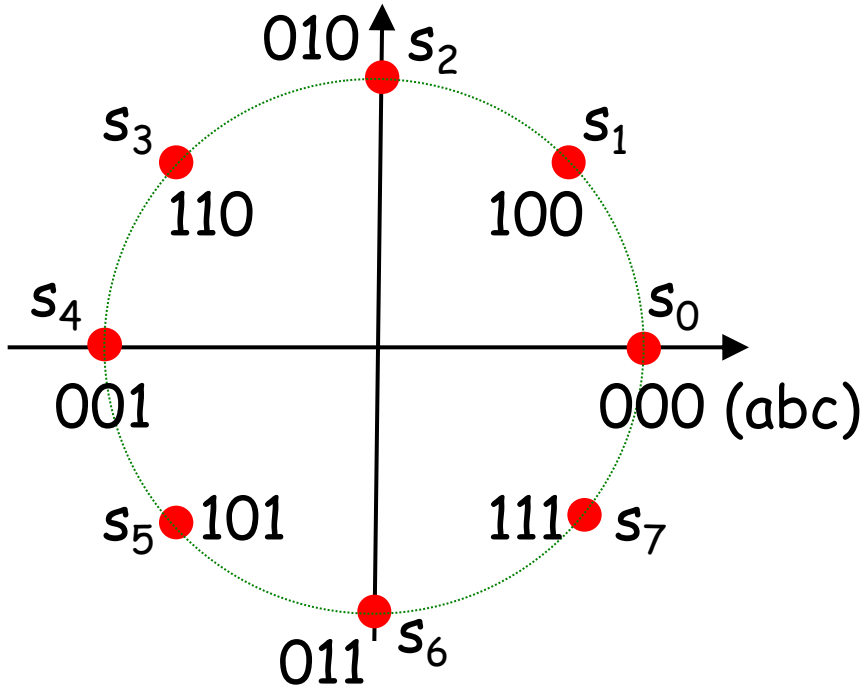




This binary encoder is not optimal (d_{free} = 3 only).

Trellis-coded modulation – An example

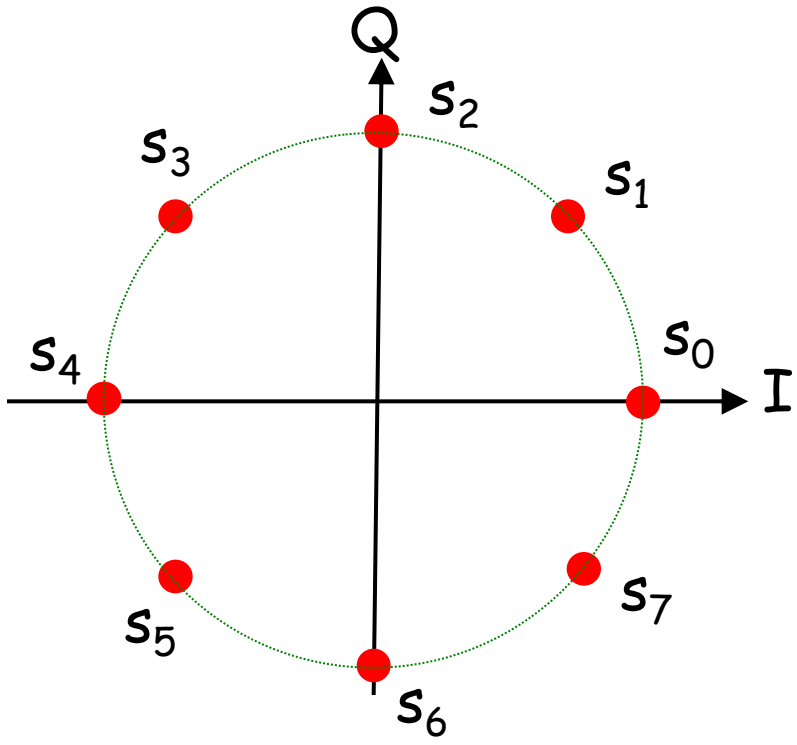
Mapping (called mapping by set partitioning)

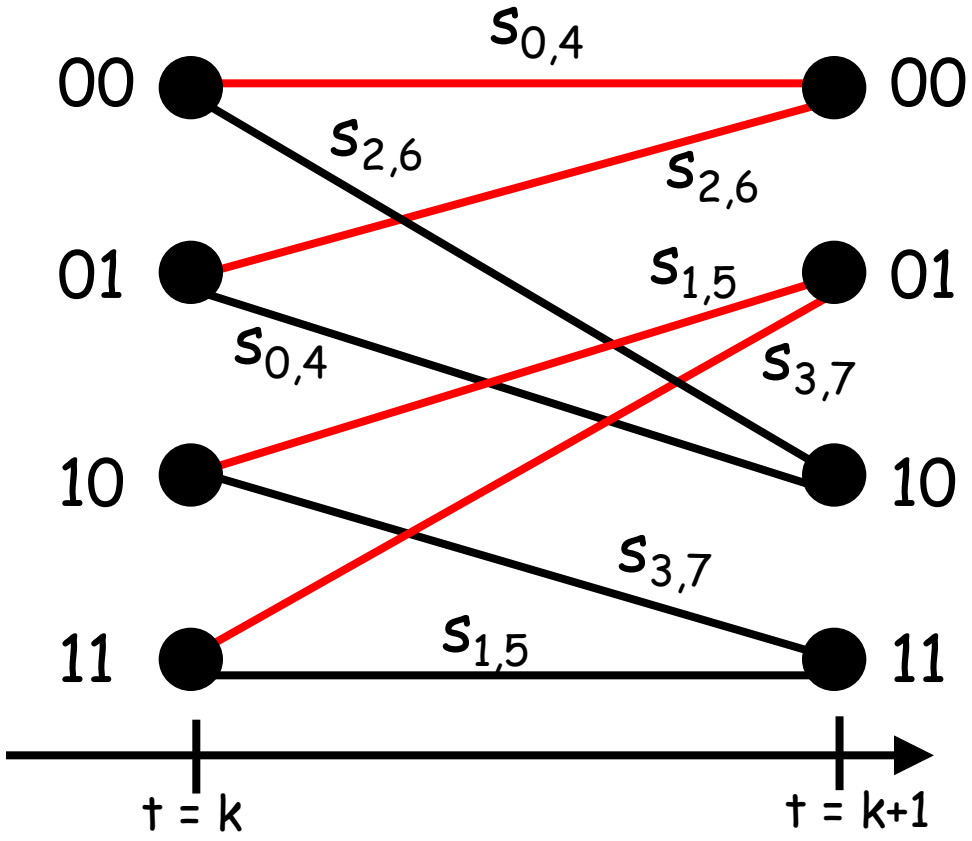


Not a Gray mapping !

Trellis-coded modulation – An example

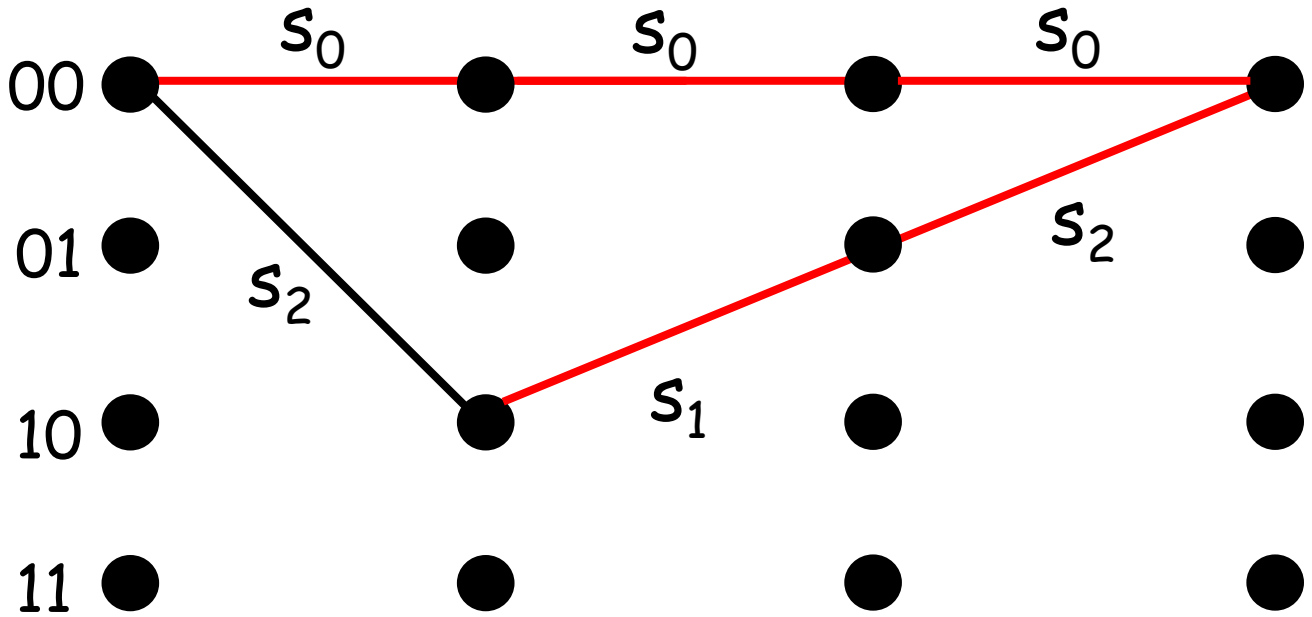
The resulting trellis of the TCM encoder is shown below.





Trellis-coded modulation – An example

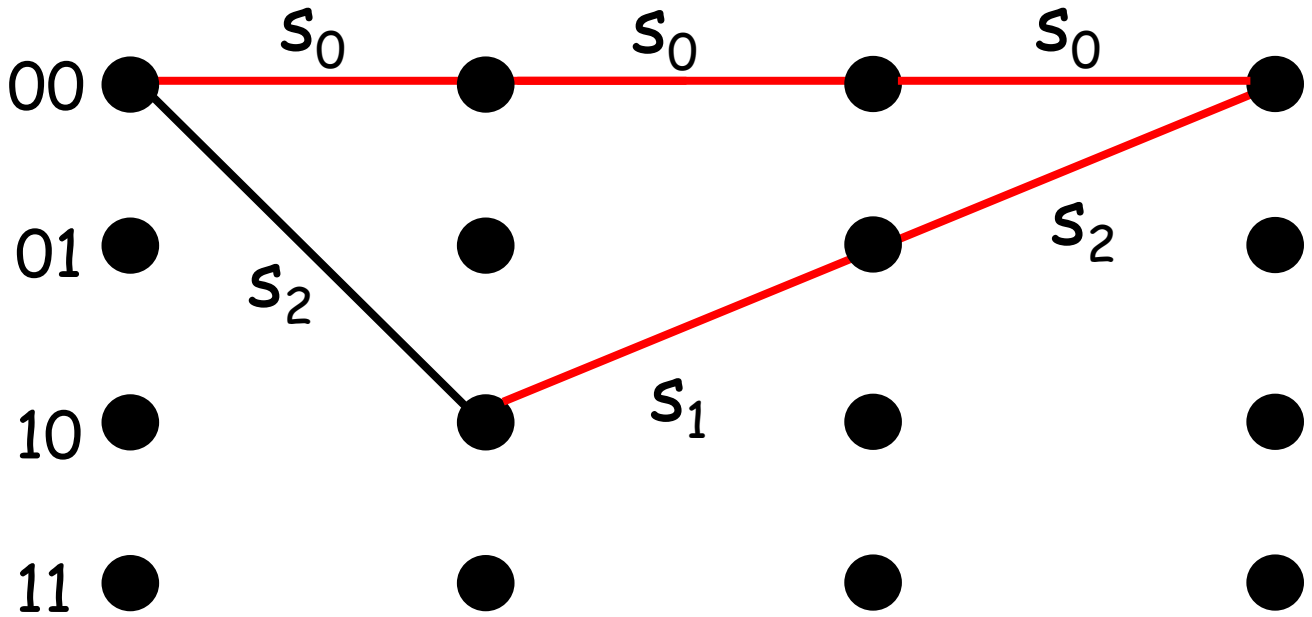
The free distance of the TCM encoder corresponds to the path shown below.



The free distance is computed in terms of squared Euclidean distance between transmitted sequences (not in terms of Hamming distance between codewords)

Trellis-coded modulation – An example

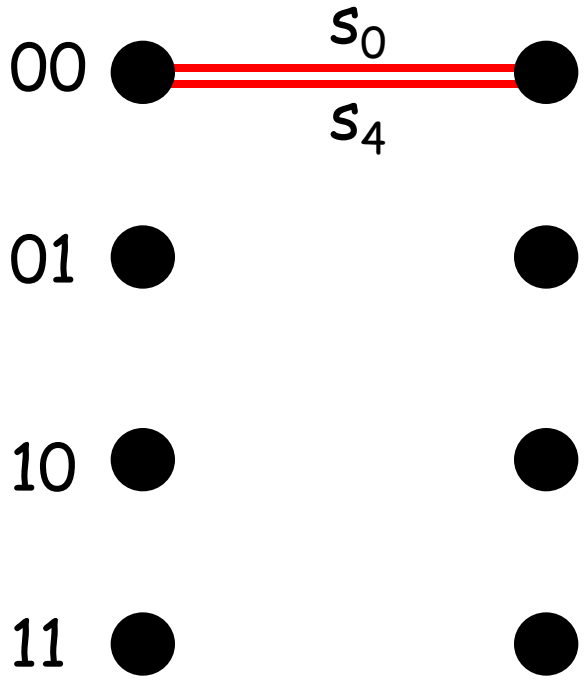
Free distance of the TCM encoder



$$\left(d_{free,1}\right)^2 = d_1^2 + d_0^2 + d_1^2 = 4 \cdot \left[1 + \sin^2\left(\frac{\pi}{8}\right)\right]$$

Trellis-coded modulation – An example

Free distance of the TCM encoder



Do not forget the parallel branches!

$$\left(d_{free,2}\right)^2 = d_3^2 = 4$$

$$\left(d_{free}\right)^2 = Min\left\{\left(d_{free,1}\right)^2, \left(d_{free,2}\right)^2\right\} = 4$$

Trellis-coded modulation – An example

Symbol error probability of a TCM modulation scheme over AWGN channel, at high SNR:

$$P_{es} \approx \frac{N_{free}}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{d_{free}^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

where N_{free} is the average number of nearest neighbour signal sequences with distance d_{free} that diverge at any state from a transmitted signal sequence and remerge with it after one or more transitions (N_{free} = 1 in the previous example).

Trellis-coded modulation – An example

For uncoded QPSK, we have

$$P_{es} \approx \frac{N}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{d_0^2}{4} \cdot \frac{E_s}{N_0}} \right)$$

with N = 2 and $d_0 = \sqrt{2}$.

Asymptotic coding gain of the 4-state, 8PSK TCM over uncoded QPSK (in dB):

$$G \approx 10 \cdot \log_{10}\left[\frac{d_{free}^2}{d_0^2} \right] = 10 \cdot \log_{10}\left[\frac{4}{2} \right] = 3.01$$

Trellis-coded modulation
An example

This 3-dB asymptotic gain is obtained at no cost in terms of spectral efficiency!

This is more than with any other coding approach (e.g., BICM).

But, since the discovery of near-capacity codes, TCM has lost its appeal.