4F5: Advanced Wireless Communications

Handout 4: Trellis Decoders and Turbo Codes

Jossy Sayir

Signal Processing and Communications Lab
Department of Engineering
University of Cambridge
jossy.sayir@eng.cam.ac.uk

Lent 2012

(7,3) Hamming Code

Parity-check matrix, $N = 7$, $N - K = 4$:

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

Systematic Parity-check matrix (by Gaussian elimination):

$$H_{\text{sys}} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

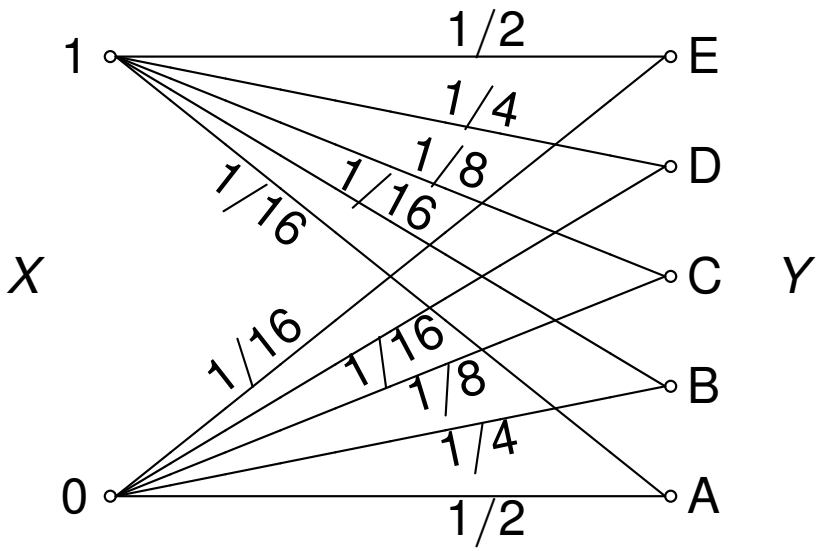
Systematic generator matrix:

$$G_{\text{sys}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Codewords
0000 000
0001 111
0010 110
0100 101
1000 011
0011 001
0101 010
1001 100
0110 011
1010 101
1100 110
0111 100
1011 010
1101 001
1110 000
1111 111

Decoding problem

Discrete Memoryless Channel, for example

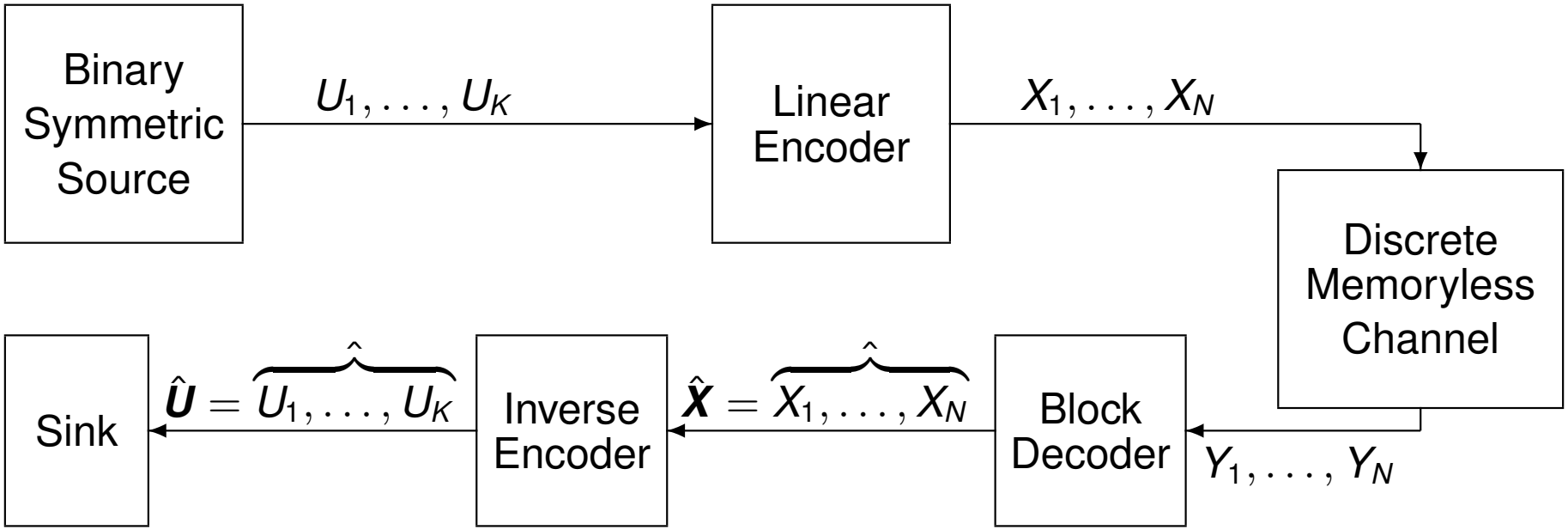


Hamming codeword transmitted, received word for ecample

$$\boldsymbol{y} = (y_1, \ldots, y_7) = (\text{A,D,C,B,E,D,E})$$

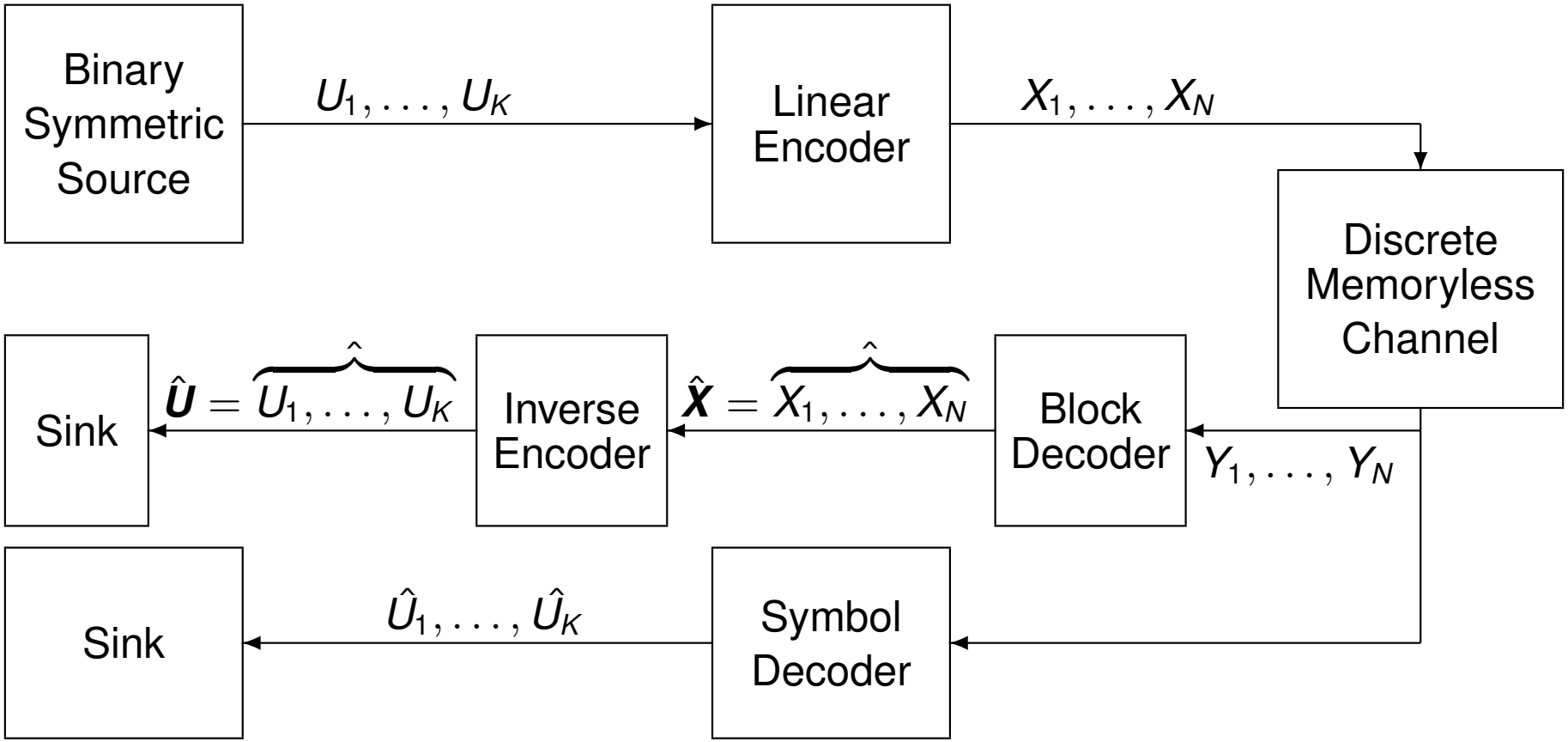
What was the encoded information sequence?

Symbol vs. Block Decoding



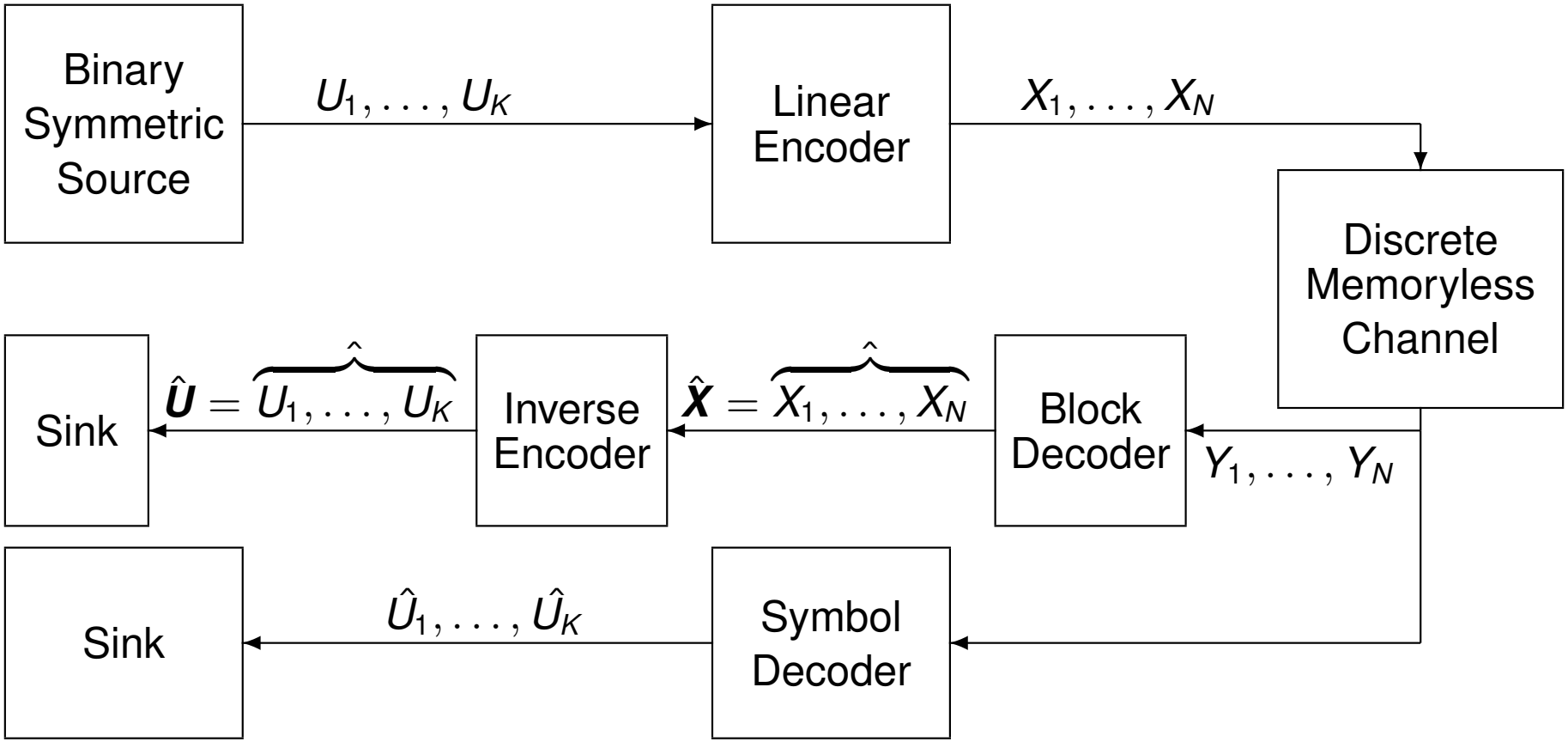
- For *systematic* encoders, the "inverse encoder" just picks the systematic part of the estimated codeword, and the symbol decoder can be viewed alternatively as estimating codeword symbols $(X_1, \ldots, X_K)$.
- Asymptotically, both approaches are equivalent and yield arbitrary reliability at rates below capacity for capacity-achieving families of codes.

Symbol vs. Block Decoding



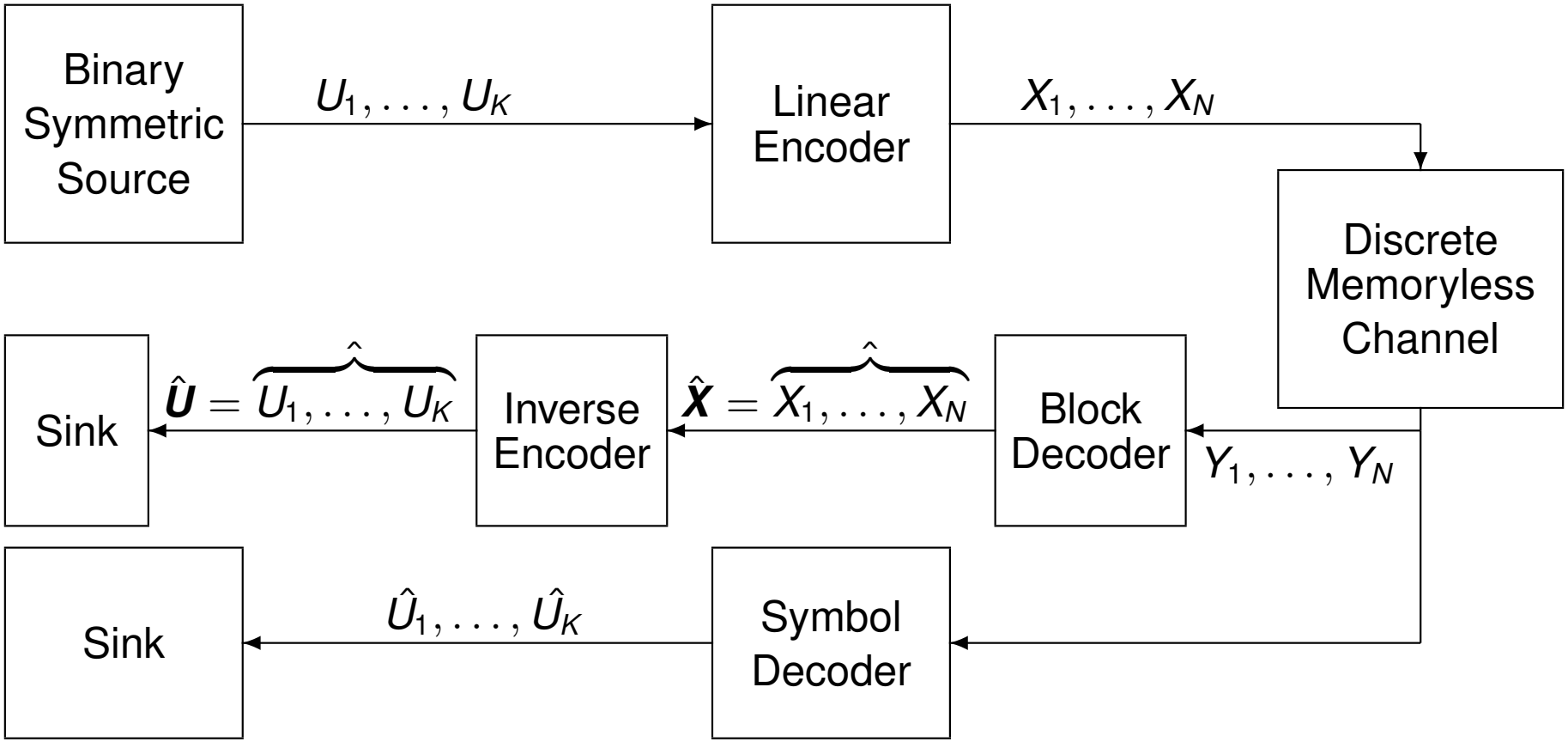
- For *systematic* encoders, the "inverse encoder" just picks the systematic part of the estimated codeword, and the symbol decoder can be viewed alternatively as estimating codeword symbols $(X_1, \ldots, X_K)$.
- Asymptotically, both approaches are equivalent and yield arbitrary reliability at rates below capacity for capacity-achieving families of codes.

Symbol vs. Block Decoding



- For *systematic* encoders, the "inverse encoder" just picks the systematic part of the estimated codeword, and the symbol decoder can be viewed alternatively as estimating codeword symbols $(X_1, \ldots, X_K)$.
- Asymptotically, both approaches are equivalent and yield arbitrary reliability at rates below capacity for capacity-achieving families of codes.

Symbol vs. Block Decoding



- For *systematic* encoders, the "inverse encoder" just picks the systematic part of the estimated codeword, and the symbol decoder can be viewed alternatively as estimating codeword symbols $(X_1, \ldots, X_K)$.
- Asymptotically, both approaches are equivalent and yield arbitrary reliability at rates below capacity for capacity-achieving families of codes.

Optimal ML and MAP block decoding

Maximum A-Posteriori (MAP) Decoding

Pick a codeword whose conditional probability given the channel output observation is maximal, i.e.,

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x}} P_{\boldsymbol{X}|\boldsymbol{Y}}(\boldsymbol{x}|\boldsymbol{y}).$$

where the notation $\arg\max_x f(x)$ is taken to mean here and hereafter "pick any one of the values achieving the maximum of $f(.)$".

Maximum Likelihood (ML) decoding

Pick a codeword that maximises the conditional probability of the channel output observation (likelihood), i.e.,

$$\hat{\boldsymbol{x}} = \arg\max_{\boldsymbol{x}} P_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}|\boldsymbol{x}).$$

For discrete memoryless channels, this reduces to

$$\hat{\boldsymbol{x}} = \arg\max_{(x_1,\ldots,x_N)\in\mathcal{C}} \prod_{i=1}^{N} P_{Y|X}(y_i|x_i).$$

It is easy to see that, for equally likely codewords, ML and MAP decoding are equivalent.

Optimal ML and MAP symbol decoding

MAP/ML decoding

MAP and ML are again equivalent when U_i is uniformly distributed, i.e., $P_{U_i}(0) = P_{U_i}(1) = 1/2$, which is true by definition. For discrete memoryless channels,

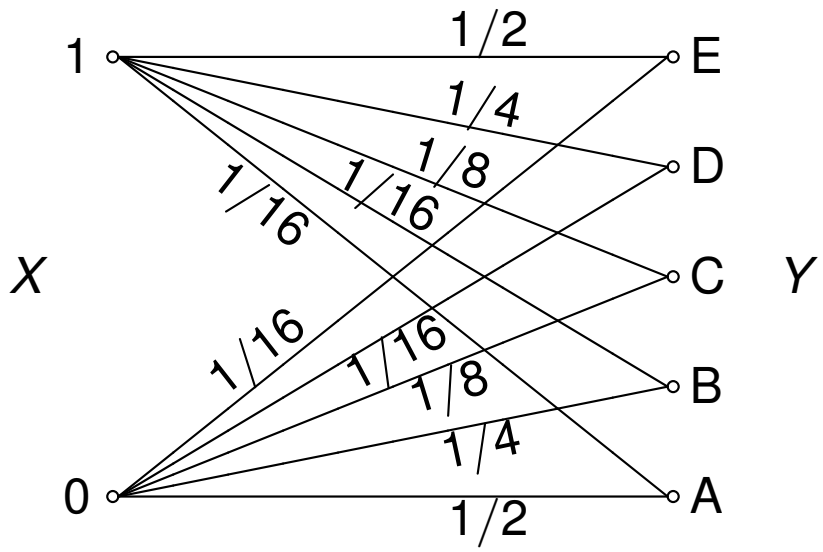
$$\begin{aligned}\hat{u}_i &= \arg\max_{u_i} P_{U_i|\boldsymbol{Y}}(u_i|\boldsymbol{y}) \\ &= \arg\max_{u_i} P_{\boldsymbol{Y}|U_i}(\boldsymbol{y}|u_i) \\ &= \arg\max_{u_i} \sum_{(x_1,\ldots,x_N)\in\mathcal{C}_{U_i=u_i}} \prod_{i=1}^{N} P_{Y|X}(y_i|x_i) \qquad (1)\end{aligned}$$

where the sum is over all the codewords corresponding to information sequences that have value u_i at their i-th position.

Since there are only 2 possibilities to chose from in the symbol decoding problem, the difficulty for this decoder is not in the decision, i.e., the arg max in (1), but in the evaluation of the a-posteriori probability, i.e., the sum and product in (1).

Brute Force Block Decoding

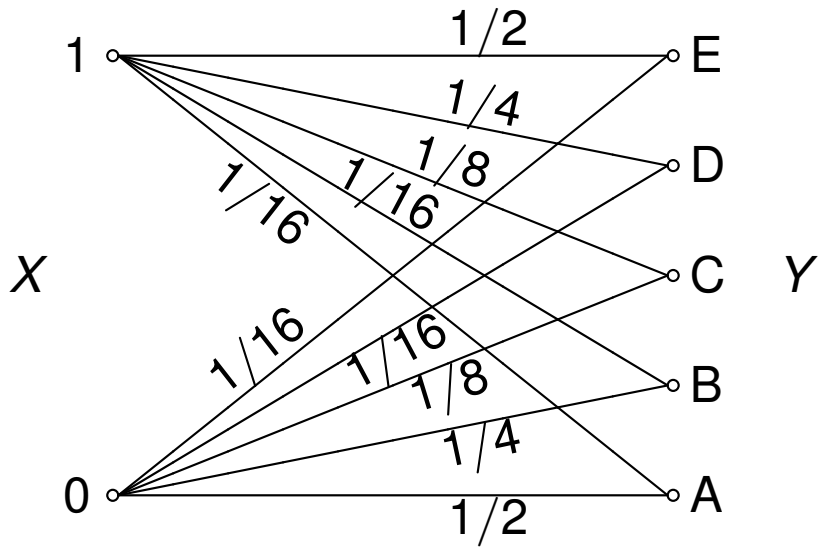
$\boldsymbol{x}$	$P_{\boldsymbol{Y}\|\boldsymbol{X}}(\boldsymbol{y}\|\boldsymbol{x})$
0000000	$\frac{1}{2}\times\frac{1}{16}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{16}$
0001111	$\frac{1}{2}\times\frac{1}{16}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{2}\times\frac{1}{4}\times\frac{1}{2}$
0010110	$\frac{1}{2}\times\frac{1}{16}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{2}\times\frac{1}{4}\times\frac{1}{16}$
0100101	$\frac{1}{2}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{2}\times\frac{1}{16}\times\frac{1}{2}$
1000011	$\frac{1}{16}\times\frac{1}{16}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{2}\times\frac{1}{4}\times\frac{1}{2}$
0011001	$\frac{1}{2}\times\frac{1}{16}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{2}$
0101010	$\frac{1}{2}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{4}\times\frac{1}{16}$
1001100	$\frac{1}{16}\times\frac{1}{16}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{2}\times\frac{1}{16}\times\frac{1}{16}$
0110011	$\frac{1}{2}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{16}\times\frac{1}{4}\times\frac{1}{2}$
1010101	$\frac{1}{16}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{2}\times\frac{1}{16}\times\frac{1}{2}$
1100110	$\frac{1}{16}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{2}\times\frac{1}{4}\times\frac{1}{16}$
0111100	$\frac{1}{2}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{2}\times\frac{1}{16}\times\frac{1}{16}$
1011010	$\frac{1}{16}\times\frac{1}{16}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{4}\times\frac{1}{16}$
1101001	$\frac{1}{16}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{2}$
1110000	$\frac{1}{16}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{4}\times\frac{1}{16}\times\frac{1}{16}\times\frac{1}{16}$
1111111	$\frac{1}{16}\times\frac{1}{4}\times\frac{1}{8}\times\frac{1}{16}\times\frac{1}{2}\times\frac{1}{4}\times\frac{1}{2}$



$\boldsymbol{y} = (y_1, \ldots, y_7) = (\text{A,D,C,B,E,D,E})$

Brute Force Block Decoding

$\boldsymbol{x}$	$P_{\boldsymbol{Y}\|\boldsymbol{X}}(\boldsymbol{y}\|\boldsymbol{x})$
0000000	2^{-22}
0001111	2^{-16}
0010110	2^{-17}
0100101	2^{-14}
1000011	2^{-20}
0011001	2^{-21}
0101010	2^{-20}
1001100	2^{-22}
0110011	2^{-15}
1010101	2^{-19}
1100110	2^{-15}
0111100	2^{-19}
1011010	2^{-25}
1101001	2^{-22}
1110000	2^{-23}
1111111	2^{-17}



$\boldsymbol{y} = (y_1, \ldots, y_7) = (\text{A,D,C,B,E,D,E})$

$\hat{\boldsymbol{x}} = (0, 1, 0, 0, 1, 0, 1)$

- Information bits: 0,1,0,0.
- $2^K(N-1)$ multiplications, $2^K - 1$ comparisons

Brute Force Symbol Decoding

$\boldsymbol{x}$	$P_{\boldsymbol{Y}\|\boldsymbol{X}}(\boldsymbol{y}\|\boldsymbol{x})$	Sum
0000000	2^{-22}	1.1802×10^{-4}
0001111	2^{-16}	
0010110	2^{-17}	
0100101	2^{-14}	
0011001	2^{-21}	
0101010	2^{-20}	
0110011	2^{-15}	
0111100	2^{-19}	
1000011	2^{-20}	4.1634×10^{-5}
1001100	2^{-22}	
1010101	2^{-19}	
1100110	2^{-15}	
1011010	2^{-25}	
1101001	2^{-22}	
1110000	2^{-23}	
1111111	2^{-17}	

- Decoding symbol at position 1, assuming systematic encoder

$$P_{X_1|\boldsymbol{Y}}(0|\boldsymbol{y}) = \frac{1.1802}{1.1802 + 0.441634}$$

- $P_{X_1|\boldsymbol{Y}}(0|\boldsymbol{y}) > P_{X_1|\boldsymbol{Y}}(1|\boldsymbol{y})$, thus $\hat{U}_1 = \hat{X}_1 = 0$.
- This operation must be repeated for U_2, U_3 and U_4. In general (though not in this case), the estimates of the symbol and block decoders do not necessarily coincide.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Block vs. Symbol Decoding in Practice

- Block decoding optimal with respect to Block Error Rate (BLER)
- Information to codeword assignment ("encoding") is irrelevant for block decoding: only the code (set of codewords) and its properties (distance spectrum etc.) matter.
- Symbol decoding optimal with respect to Bit Error Rate (BER)
- Performance of the symbol decoder depends on the code as well as on the encoder.
- The BER of a block decoder does depend on the encoder. Systematic is a good choice (although not necessarily optimal) because codewords at moderate distances from each other will result in few bit errors.
- BER $\to 0 \implies$ BLER $\to 0$ and vice versa, so asymptotically optimal systems in terms of BLER are also asymptotically optimal in terms of BER.
- Decoders presented are "decision algorithms", i.e., the output is a decision on the value of $\boldsymbol{X}$, $\boldsymbol{U}$, X_i or U_i, rather than the a-posteriori probability of those random variables.
- *Soft decoders* are "estimation algorithms" that provide a-posteriori probabilities rather than decisions. They are relevant, e.g., when the output of the decoder is used by another decoder rather than being the final verdict on the value of the information sequence.

Practical decoding metric for block ML decoding

- Multiplicative metrics are inconvenient for implementation
- Equivalent practical ML decoding rule, for any $\alpha > 0$:

$$\hat{\boldsymbol{x}} = \arg\max_{(x_1,\ldots,x_N)\in\mathcal{C}} \log_2 \left(\alpha \prod_{i=1}^{N} P_{Y|X}(y_i|x_i) \right)$$

$$= \arg\max_{(x_1,\ldots,x_N)\in\mathcal{C}} \sum_{i=1}^{N} \mu(x_i, y_i), \text{ where } \mu(x_i, y_i) \stackrel{\text{def}}{=} \log_2(P_{Y|X}(y_i|x_i)) + \beta$$

where the first step follows because multiplying by a positive constant or taking the logarithm leaves the arg max unchanged, and $\beta = \frac{1}{N} \log_2 \alpha$. α and β are picked so that all metrics are non-negative and can be implemented in fixed-point arithmetic.

- Additive decoding metric for our channel:

Practical decoding metric for block ML decoding

- Multiplicative metrics are inconvenient for implementation
- Equivalent practical ML decoding rule, for any $\alpha > 0$:

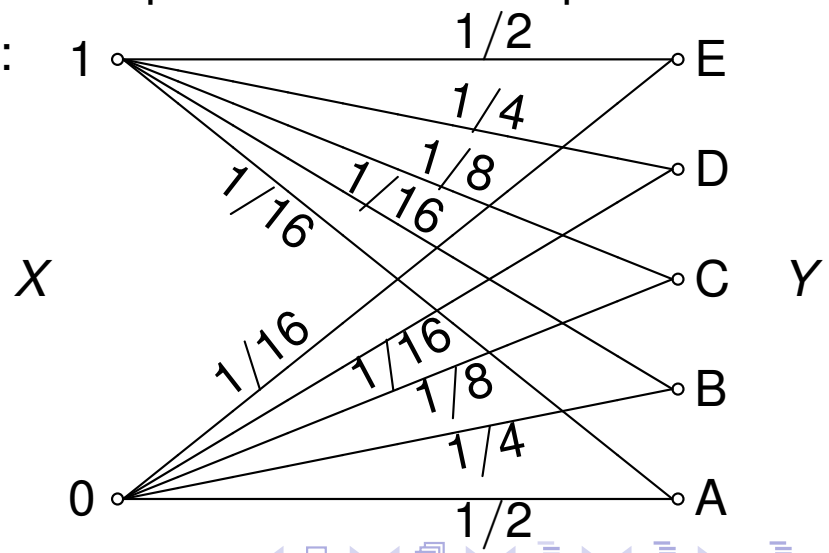
$$\hat{\boldsymbol{x}} = \arg\max_{(x_1,\ldots,x_N)\in\mathcal{C}} \log_2 \left(\alpha \prod_{i=1}^{N} P_{Y|X}(y_i|x_i) \right)$$

$$= \arg\max_{(x_1,\ldots,x_N)\in\mathcal{C}} \sum_{i=1}^{N} \mu(x_i, y_i), \text{ where } \mu(x_i, y_i) \stackrel{\text{def}}{=} \log_2(P_{Y|X}(y_i|x_i)) + \beta$$

where the first step follows because multiplying by a positive constant or taking the logarithm leaves the arg max unchanged, and $\beta = \frac{1}{N} \log_2 \alpha$. α and β are picked so that all metrics are non-negative and can be implemented in fixed-point arithmetic.

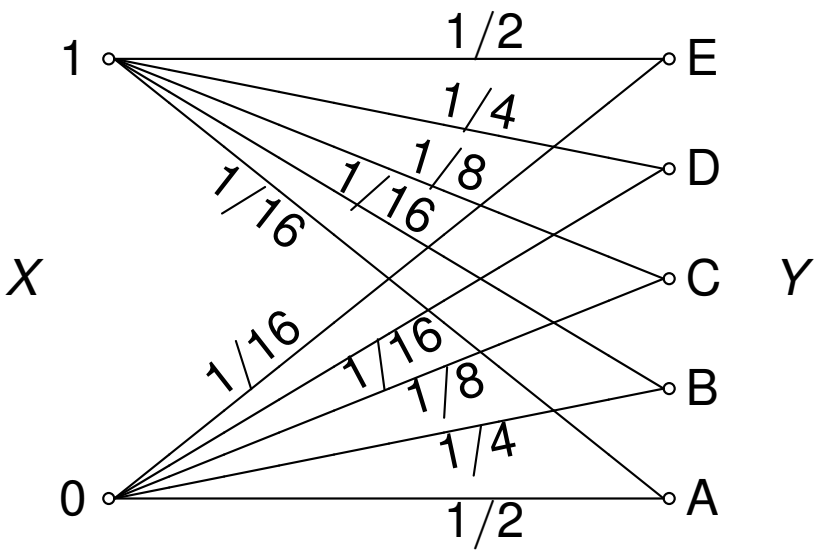
- Additive decoding metric for our channel:

x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



Brute Force Block Decoding with Additive Metric

$\boldsymbol{x}$	$\mu(\boldsymbol{X},\boldsymbol{Y})$
0000000	$3+0+1+2+0+0+0=6$
0001111	$3+0+1+0+3+2+3=12$
0010110	$3+0+1+2+3+2+0=11$
0100101	$3+2+1+2+3+0+3=14$
1000011	$0+0+1+2+0+2+3=8$
0011001	$3+0+1+0+0+0+3=7$
0101010	$3+2+1+0+0+2+0=8$
1001100	$0+2+1+0+3+0+0=6$
0110011	$3+2+1+2+0+2+3=13$
1010101	$0+0+1+2+3+0+3=9$
1100110	$0+2+1+2+3+2+0=10$
0111100	$3+2+1+0+3+0+0=9$
1011010	$0+0+1+0+0+2+0=3$
1101001	$0+2+1+0+0+0+3=6$
1110000	$0+2+1+2+0+0+0=5$
1111111	$0+2+1+0+3+2+3=11$



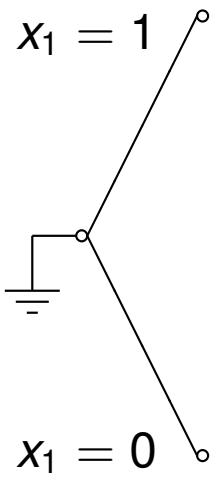
$\boldsymbol{y} = (y_1, \ldots, y_7) = (\text{A,D,C,B,E,D,E})$

$\hat{\boldsymbol{x}} = (0,1,0,0,1,0,1)$

- Information bits: 0,1,0,0.
- $2^K(N-1)$ additions, $2^K - 1$ comparisons

Tree and trellis of a Linear Block Code

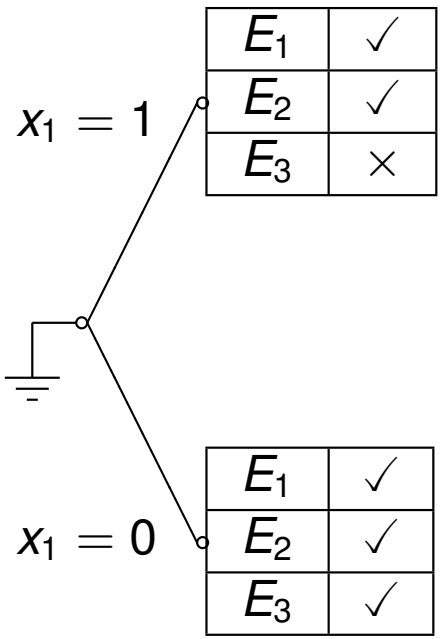
$$0 = \boldsymbol{x}\boldsymbol{H}^T = \boldsymbol{x} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \qquad \begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$



Tree and trellis of a Linear Block Code

$$0 = \boldsymbol{x}\boldsymbol{H}^T = \boldsymbol{x} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

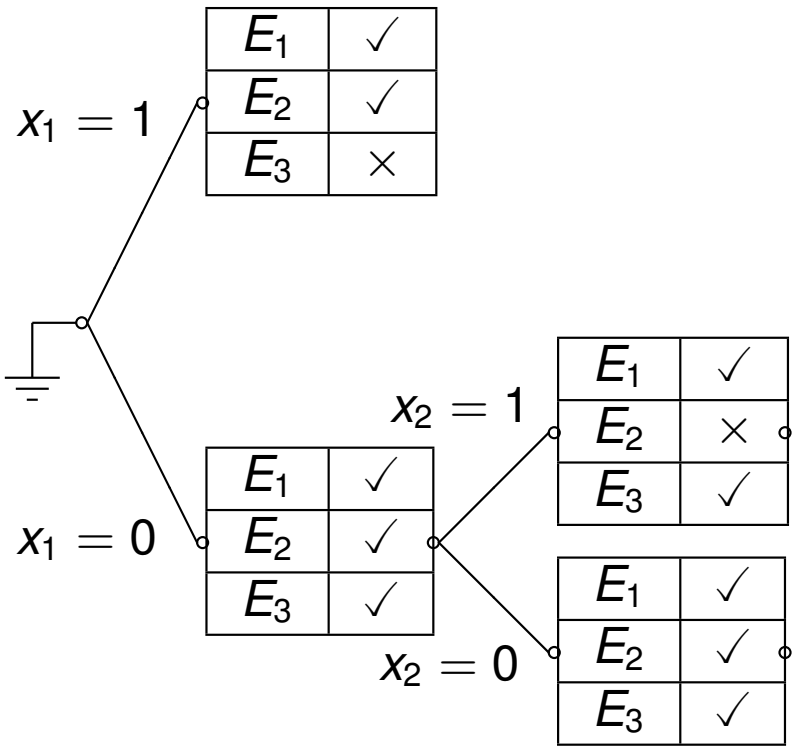
$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$



Tree and trellis of a Linear Block Code

$$0 = \boldsymbol{x}\boldsymbol{H}^T = \boldsymbol{x} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

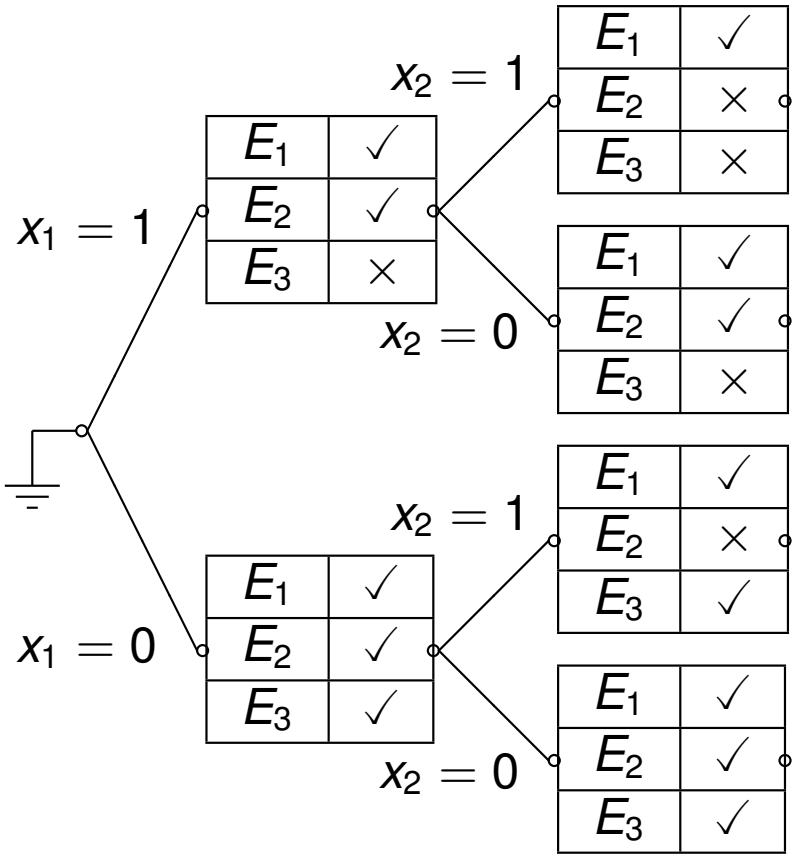
$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$



Tree and trellis of a Linear Block Code

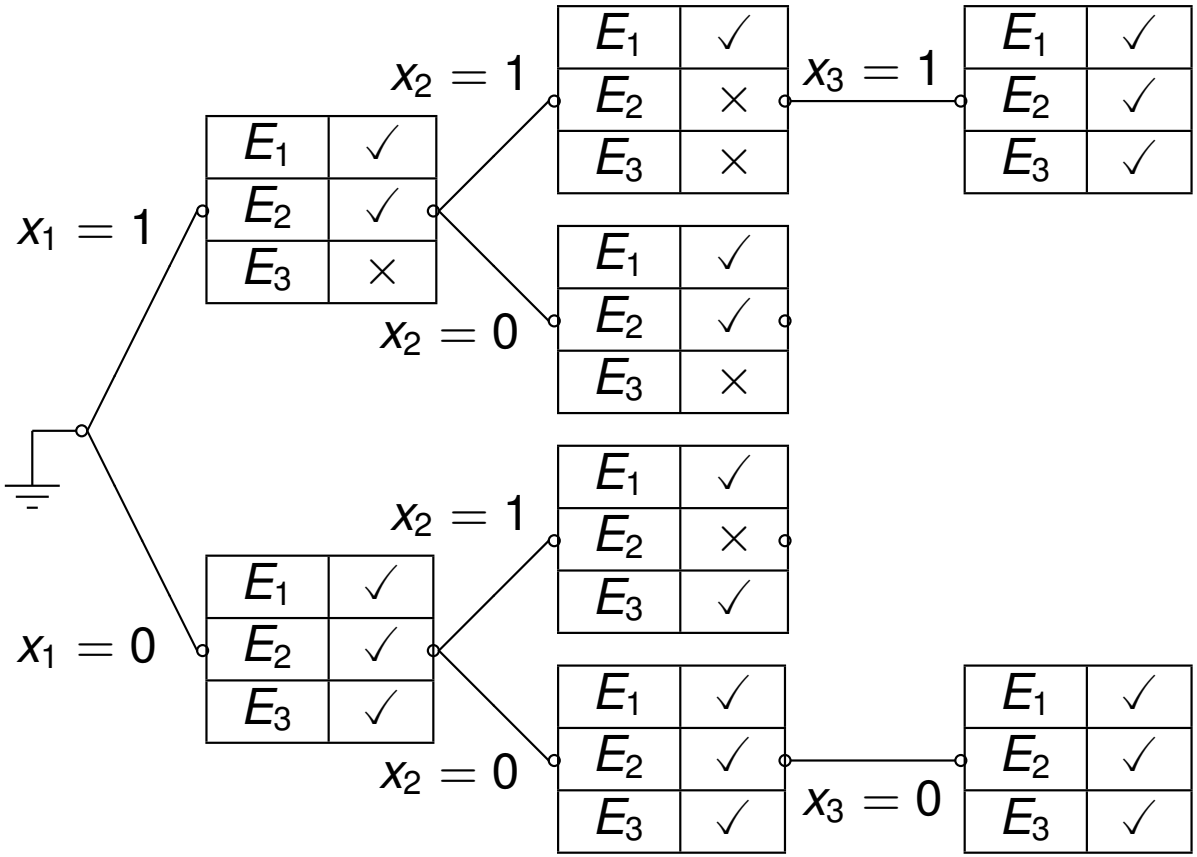
$$0 = \boldsymbol{x}\boldsymbol{H}^T = \boldsymbol{x} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$



Tree and trellis of a Linear Block Code

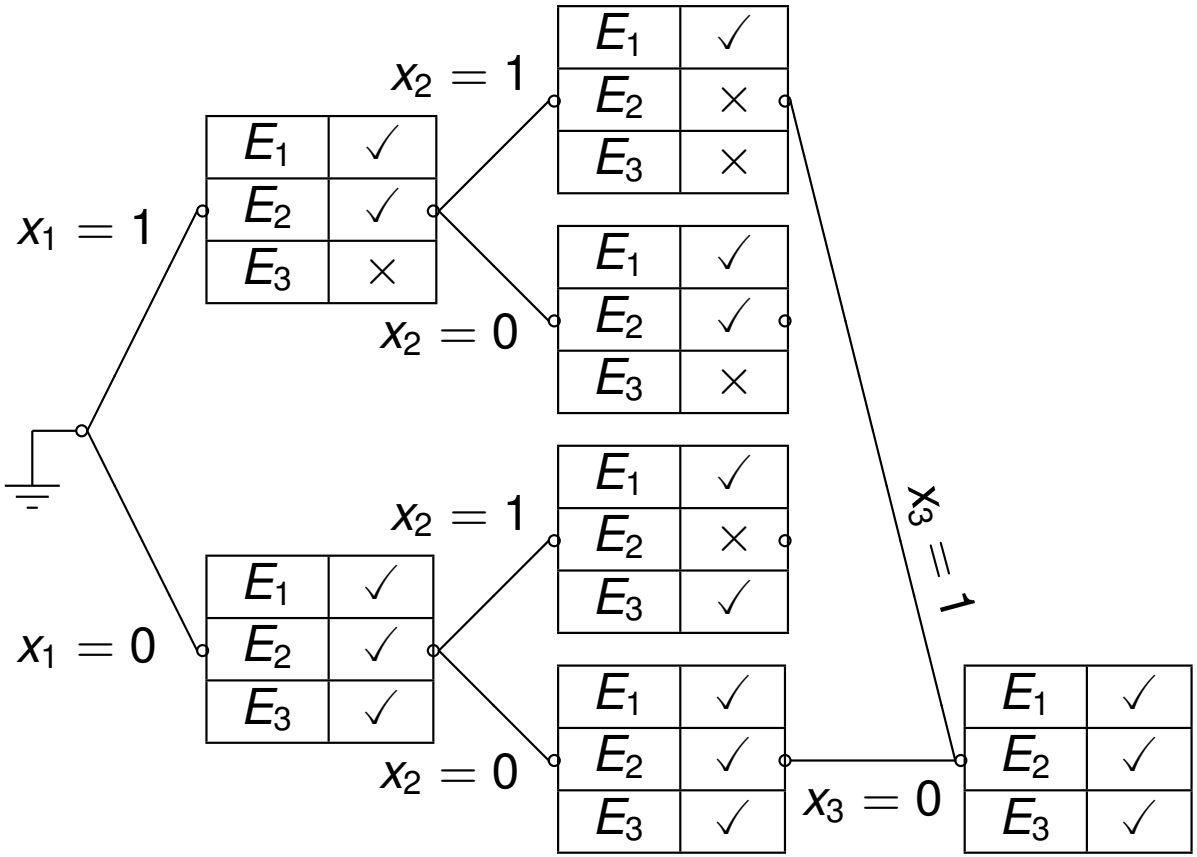
$$0 = \boldsymbol{x}\boldsymbol{H}^T = \boldsymbol{x} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \qquad \begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$



Tree and trellis of a Linear Block Code

$$0 = \boldsymbol{x}\boldsymbol{H}^T = \boldsymbol{x} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

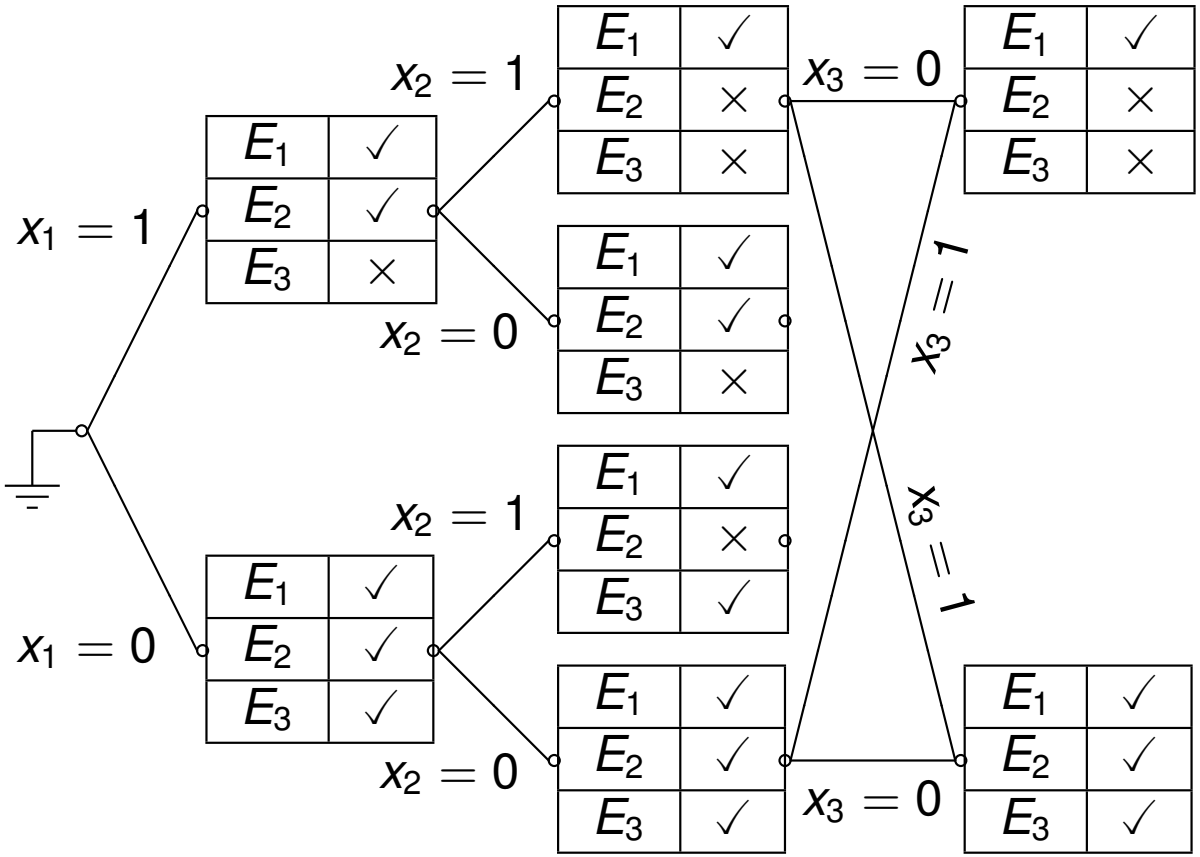
$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$



For every codeword $\boldsymbol{x}$ with prefix $(x_1, x_2, x_3) = (0, 0, 0)$, there is another codeword $\boldsymbol{x}'$ with prefix $(x_1', x_2', x_3') = (1, 1, 1)$ for which $(x_4', x_5', x_6', x_7') = (x_4, x_5, x_6, x_7)$

Tree and trellis of a Linear Block Code

$$0 = \boldsymbol{x}\boldsymbol{H}^T = \boldsymbol{x} \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T \qquad \begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$



For every codeword $\boldsymbol{x}$ with prefix $(x_1, x_2, x_3) = (1, 1, 0)$, there is another codeword $\boldsymbol{x}'$ with prefix $(x_1', x_2', x_3') = (0, 0, 1)$ for which $(x_4', x_5', x_6', x_7') = (x_4, x_5, x_6, x_7)$

etc.

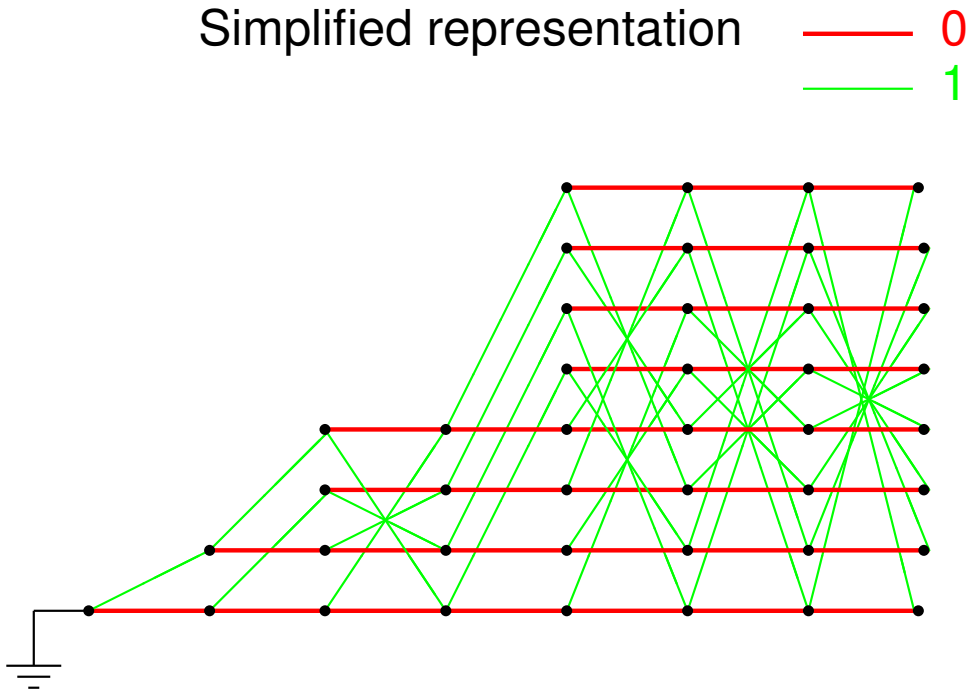
For every codeword $\boldsymbol{x}$ with prefix $(x_1, x_2, x_3) = (0, 0, 0)$, there is another codeword $\boldsymbol{x}'$ with prefix $(x_1', x_2', x_3') = (1, 1, 1)$ for which $(x_4', x_5', x_6', x_7') = (x_4, x_5, x_6, x_7)$

Tree and trellis of a Linear Block Code

$$0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$

E_1	E_2	E_3
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

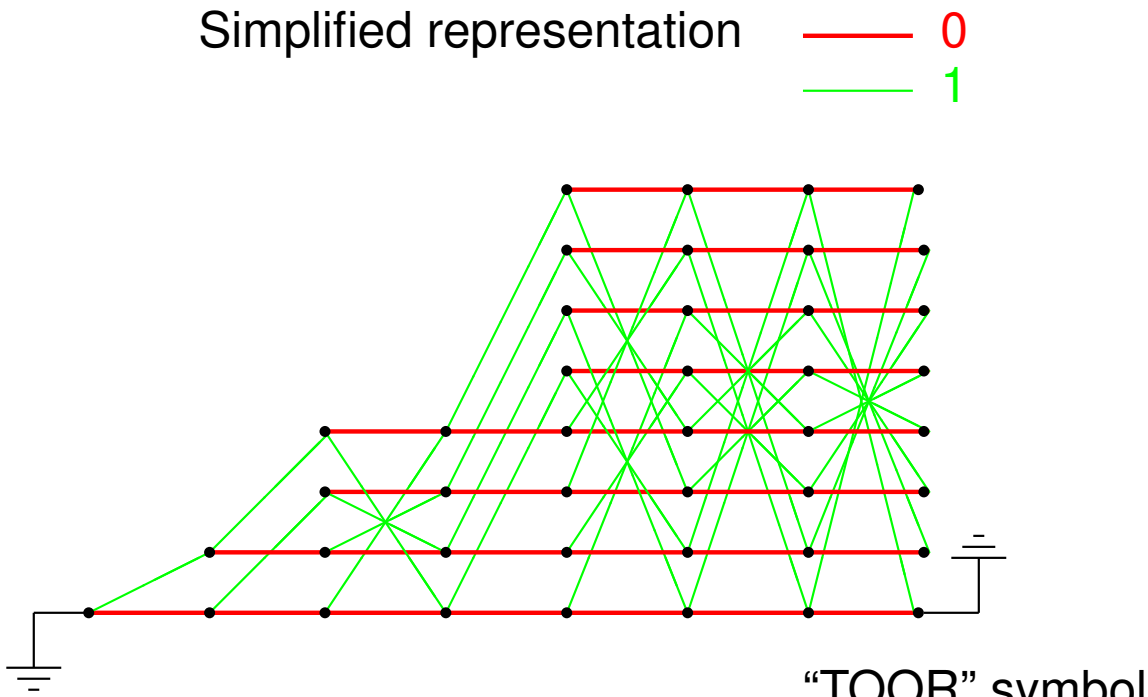


Tree and trellis of a Linear Block Code

$$0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$

E_1	E_2	E_3
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0

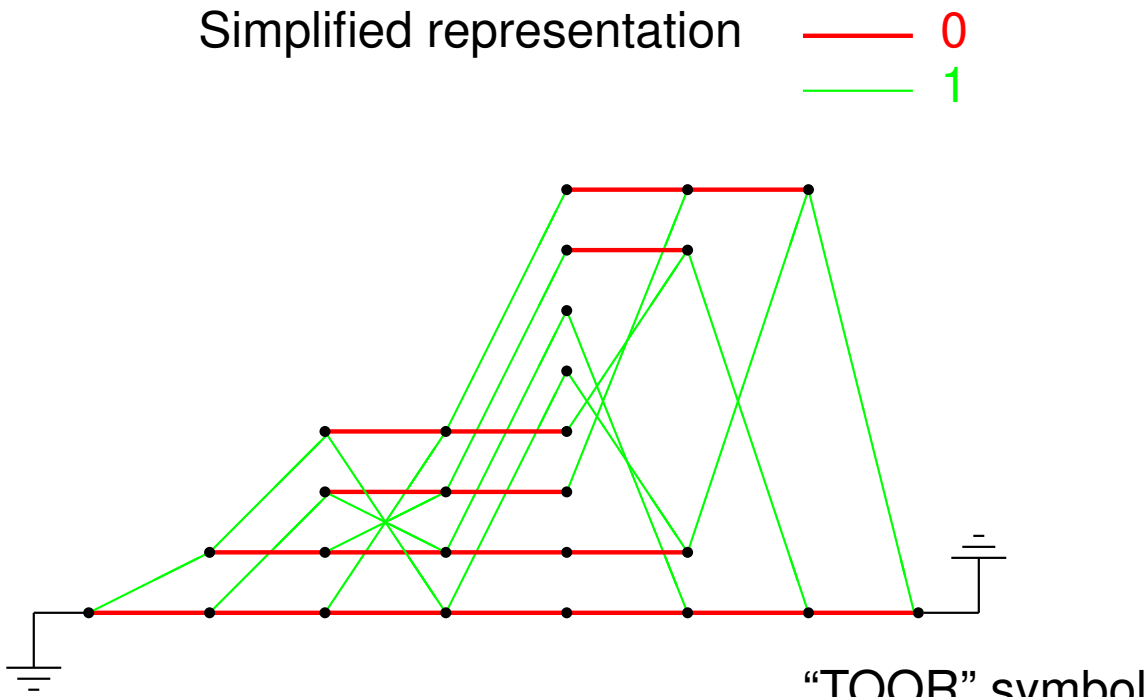


Tree and trellis of a Linear Block Code

$$0 = xH^T = x \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}^T$$

$$\begin{cases} x_4 + x_5 + x_6 + x_7 = 0 & (E_1) \\ x_2 + x_3 + x_6 + x_7 = 0 & (E_2) \\ x_1 + x_3 + x_5 + x_7 = 0 & (E_3) \end{cases}$$

E_1	E_2	E_3
1	1	1
1	1	0
1	0	1
1	0	0
0	1	1
0	1	0
0	0	1
0	0	0



Viterbi's principle

For every codeword $\boldsymbol{x}$ with prefix $(x_1, x_2, x_3) = (0, 0, 0)$, there is another codeword $\boldsymbol{x}'$ with prefix $(x'_1, x'_2, x'_3) = (1, 1, 1)$ for which $(x'_4, x'_5, x'_6, x'_7) = (x_4, x_5, x_6, x_7)$

If prefix $(1, 1, 1)$ beats prefix $(0, 0, 0)$ in terms of additive metric, then the metrics of all codewords starting with $(1, 1, 1)$ will be larger than the metrics of all codewords starting with $(0, 0, 0)$

When paths merge in a trellis, eliminate the prefix with the lower metric!

The Viterbi Algorithm

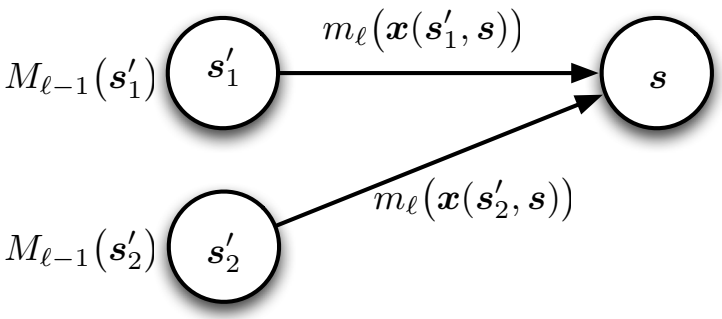
Let $M_\ell(\boldsymbol{s})$ be the metric corresponding to state $\boldsymbol{s}$ after stage ℓ in the trellis, and $m_\ell(\boldsymbol{x}(\boldsymbol{s}', \boldsymbol{s}))$ be the additive metric term added between stage $\ell - 1$ and ℓ in the trellis when transitioning between state $\boldsymbol{s}'$ and $\boldsymbol{s}$.

Viterbi Decoding

1. Initialisation $M_0(\mathbf{0}) = 0$
2. Add Compare Select (ACS) recursion: for $\ell = 1, \ldots, L$ and for all states $\boldsymbol{s} = (s_1, \ldots, s_\nu)$, calculate

$$M_\ell(\boldsymbol{s}) = \max_{\boldsymbol{s}' \in \Pi(\boldsymbol{s})} \{M_{\ell-1}(\boldsymbol{s}') + m_\ell(\boldsymbol{x}(\boldsymbol{s}', \boldsymbol{s}))\}$$

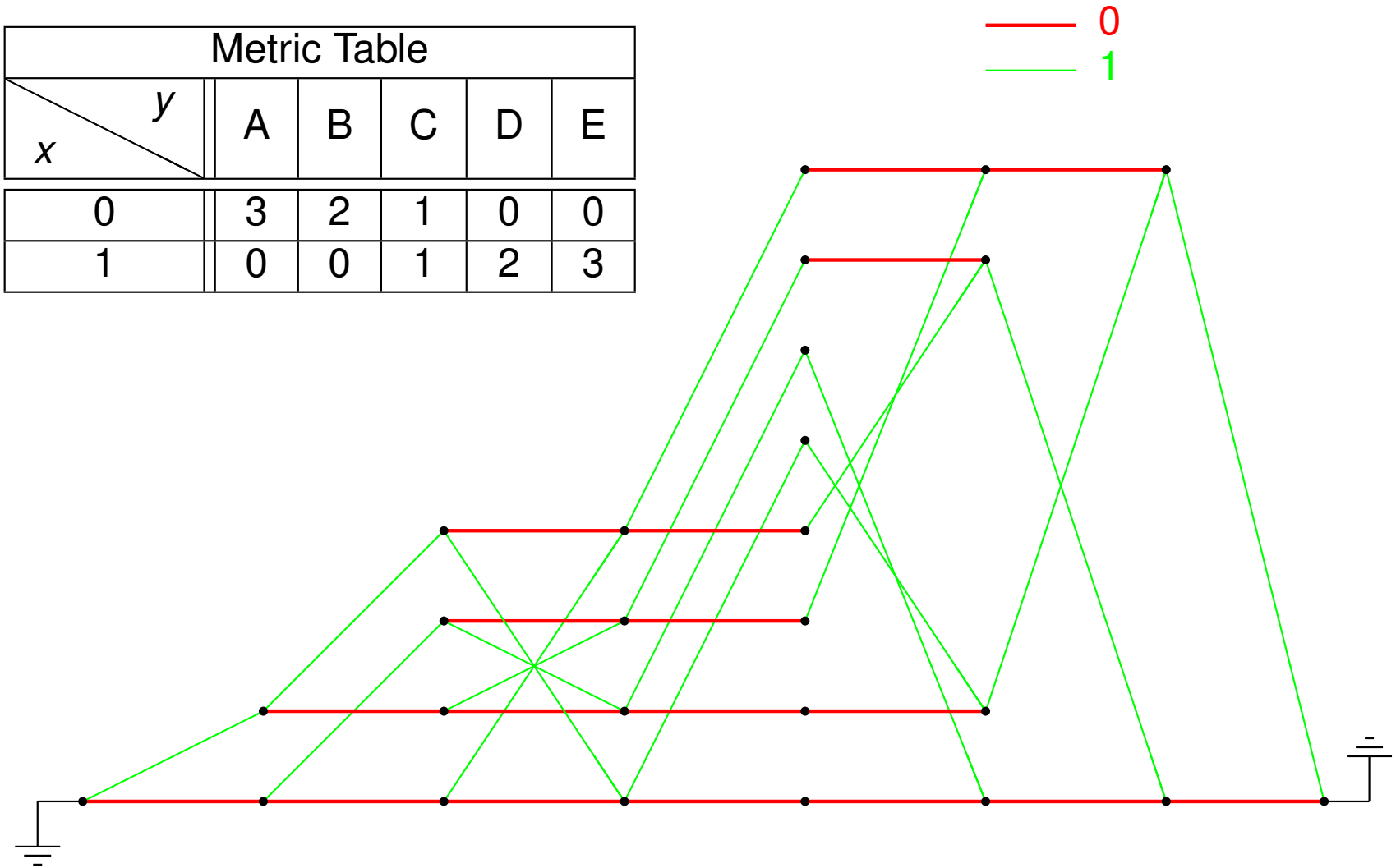
   where $\Pi(\boldsymbol{s})$ is the set of parent states to state $\boldsymbol{s}$ (states $\boldsymbol{s}'$ that have a connection with $\boldsymbol{s}$). Ties in the maximum are resolved by picking a winner at random.
3. Trace back. The maximum metric is $M_L(\mathbf{0})$, output the input sequence corresponding to $M_L(\mathbf{0})$.



$$M_\ell(\boldsymbol{s}) = \max_{\boldsymbol{s}'_1, \boldsymbol{s}'_2} \left\{ M_{\ell-1}(\boldsymbol{s}'_1) + m_\ell\big(\boldsymbol{x}(\boldsymbol{s}'_1, \boldsymbol{s})\big) \quad , \quad M_{\ell-1}(\boldsymbol{s}'_2) + m_\ell\big(\boldsymbol{x}(\boldsymbol{s}'_2, \boldsymbol{s})\big) \right\}$$

The Viterbi Decoder in Action

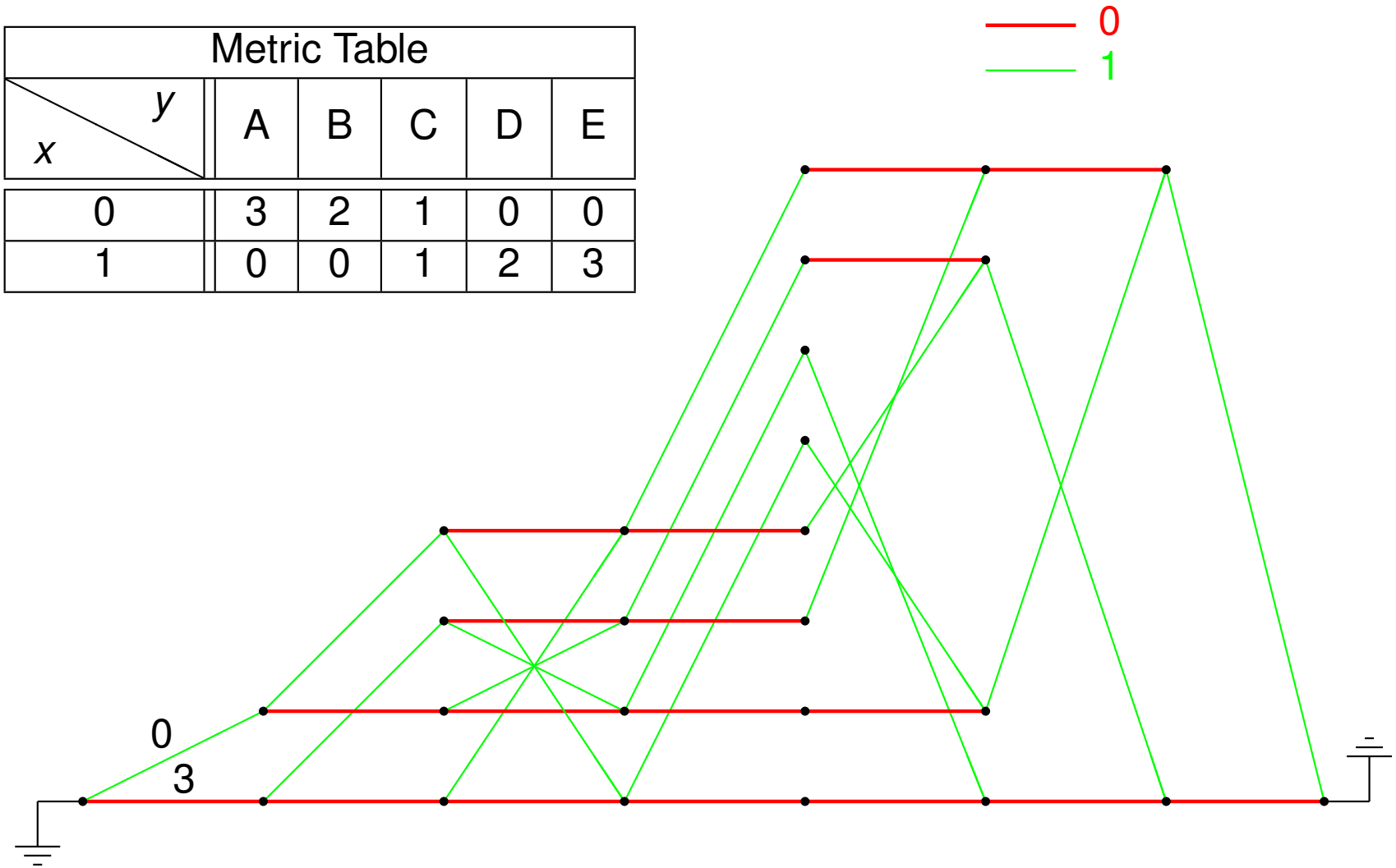
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

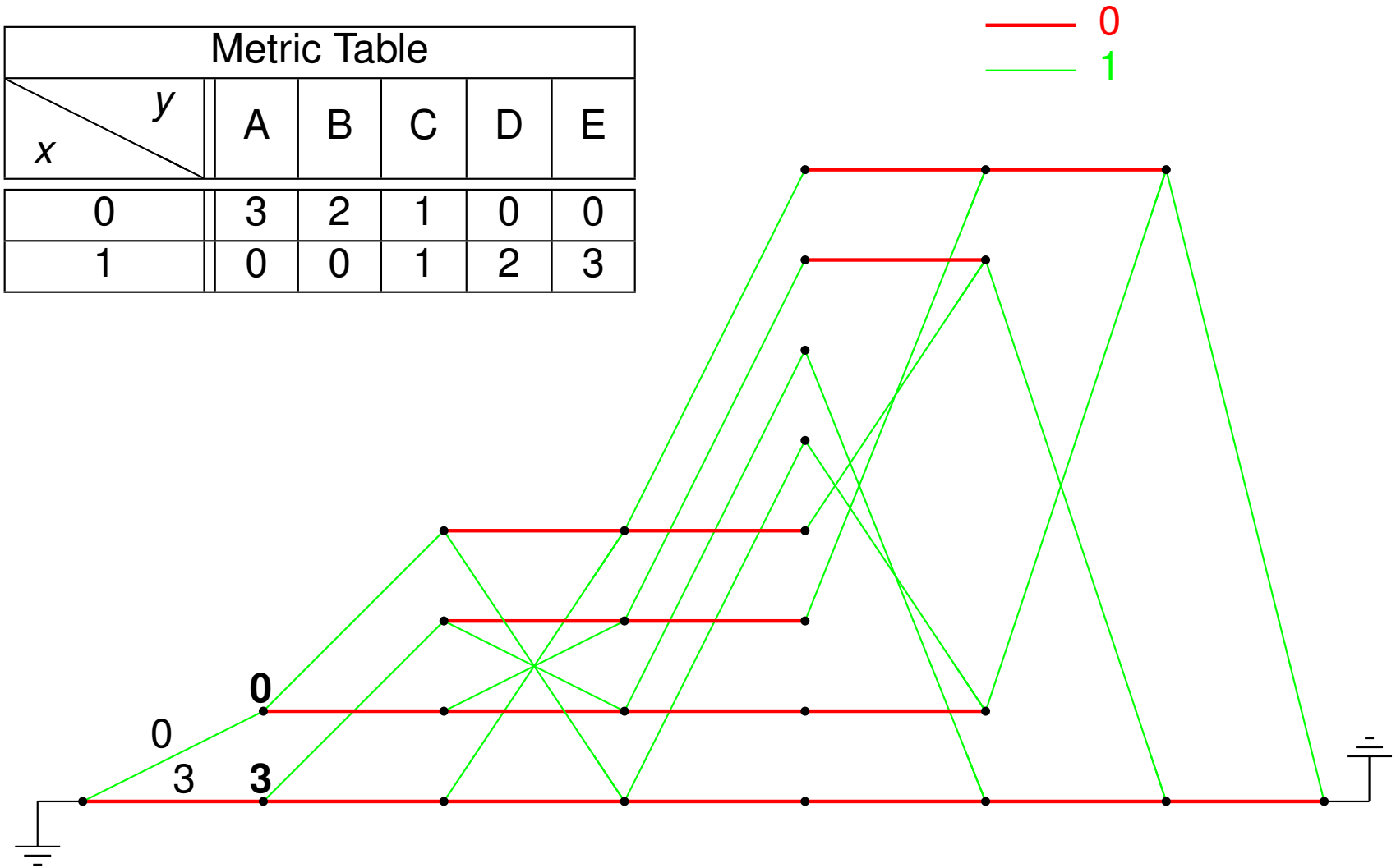
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

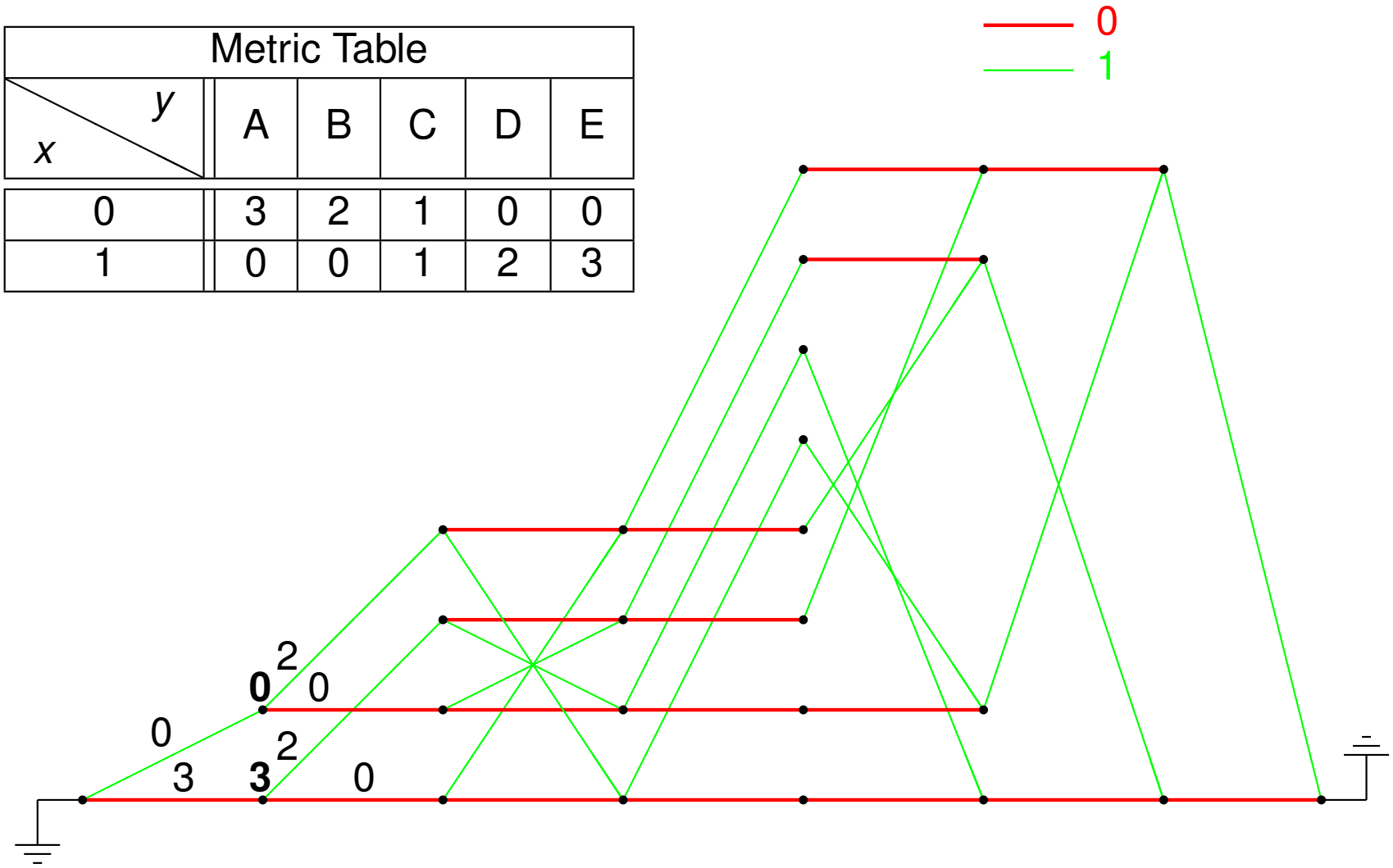
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

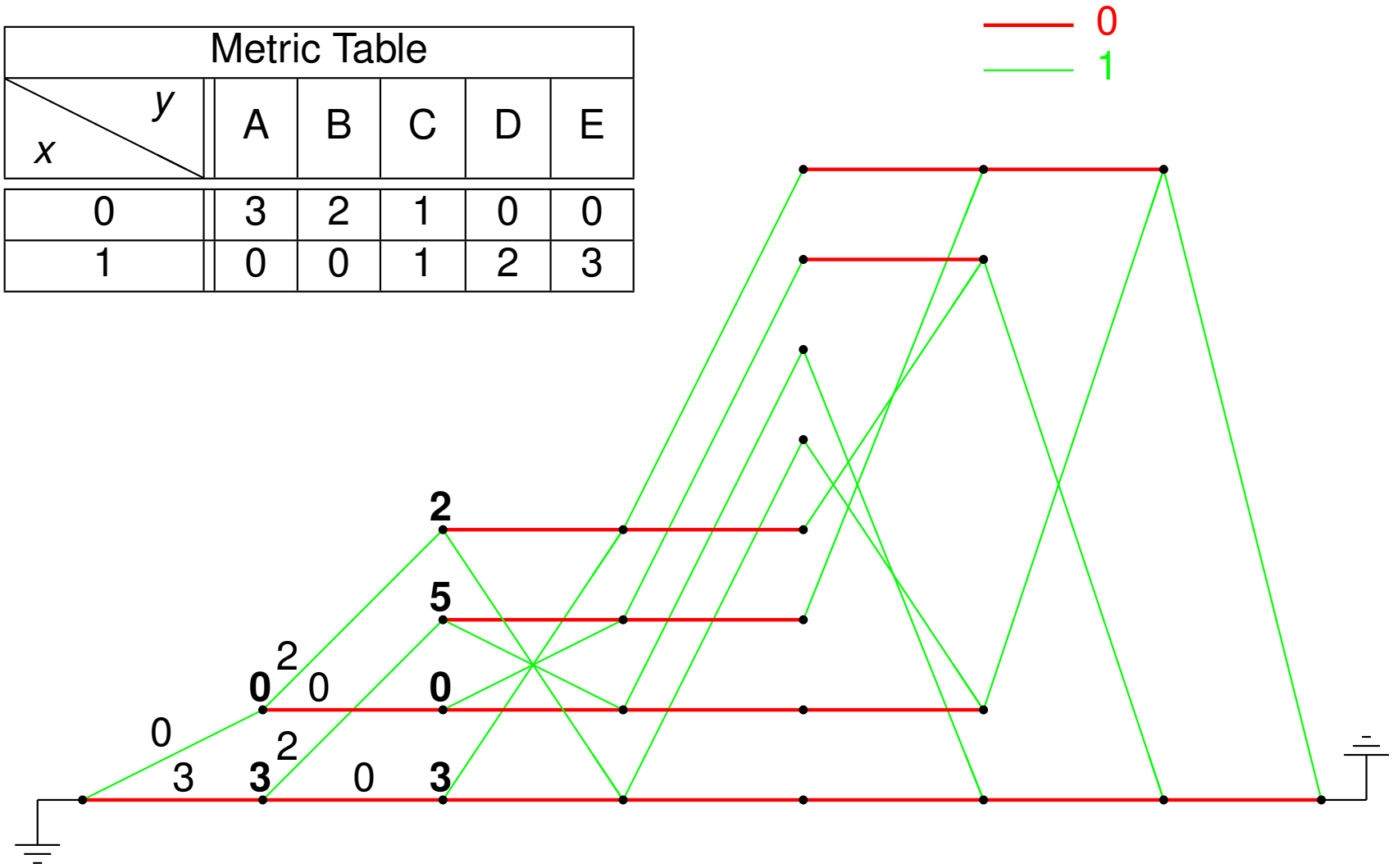
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

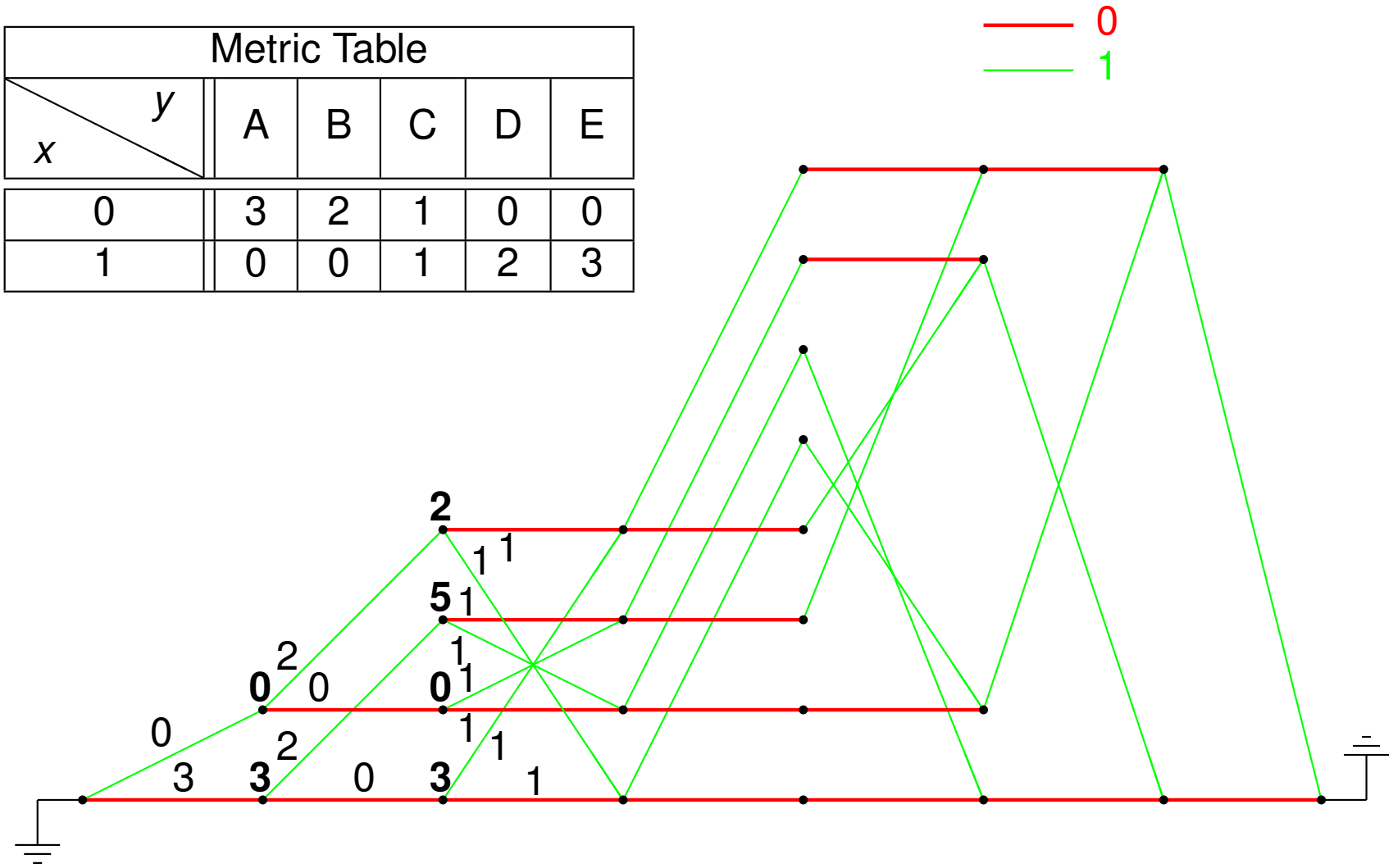
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

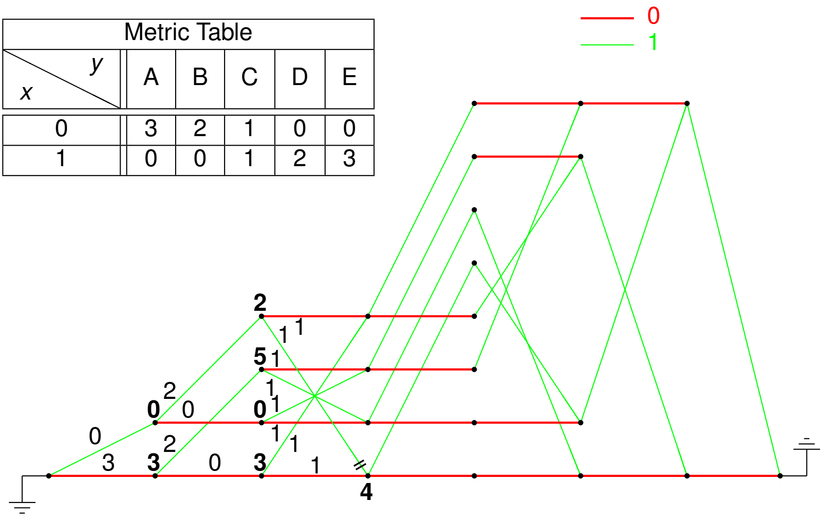
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

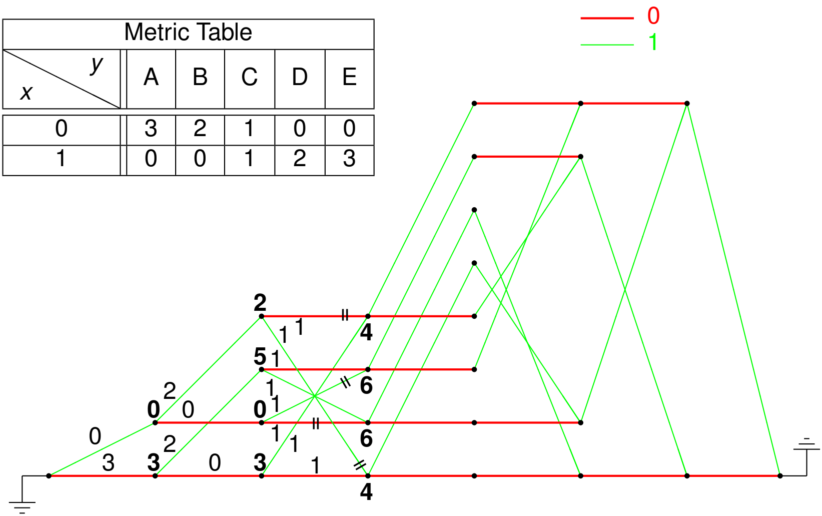
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E$)$

The Viterbi Decoder in Action

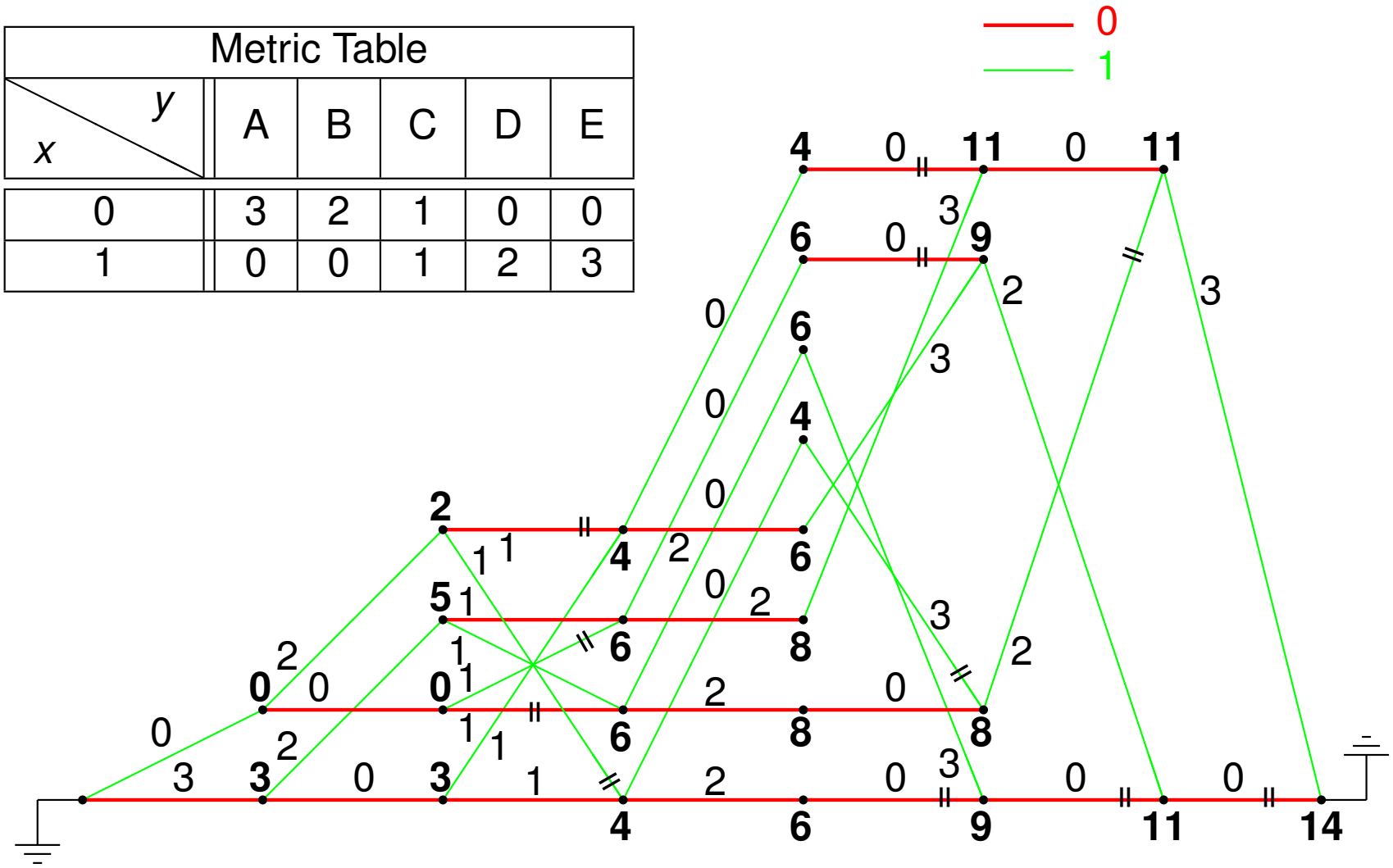
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E$)$

The Viterbi Decoder in Action

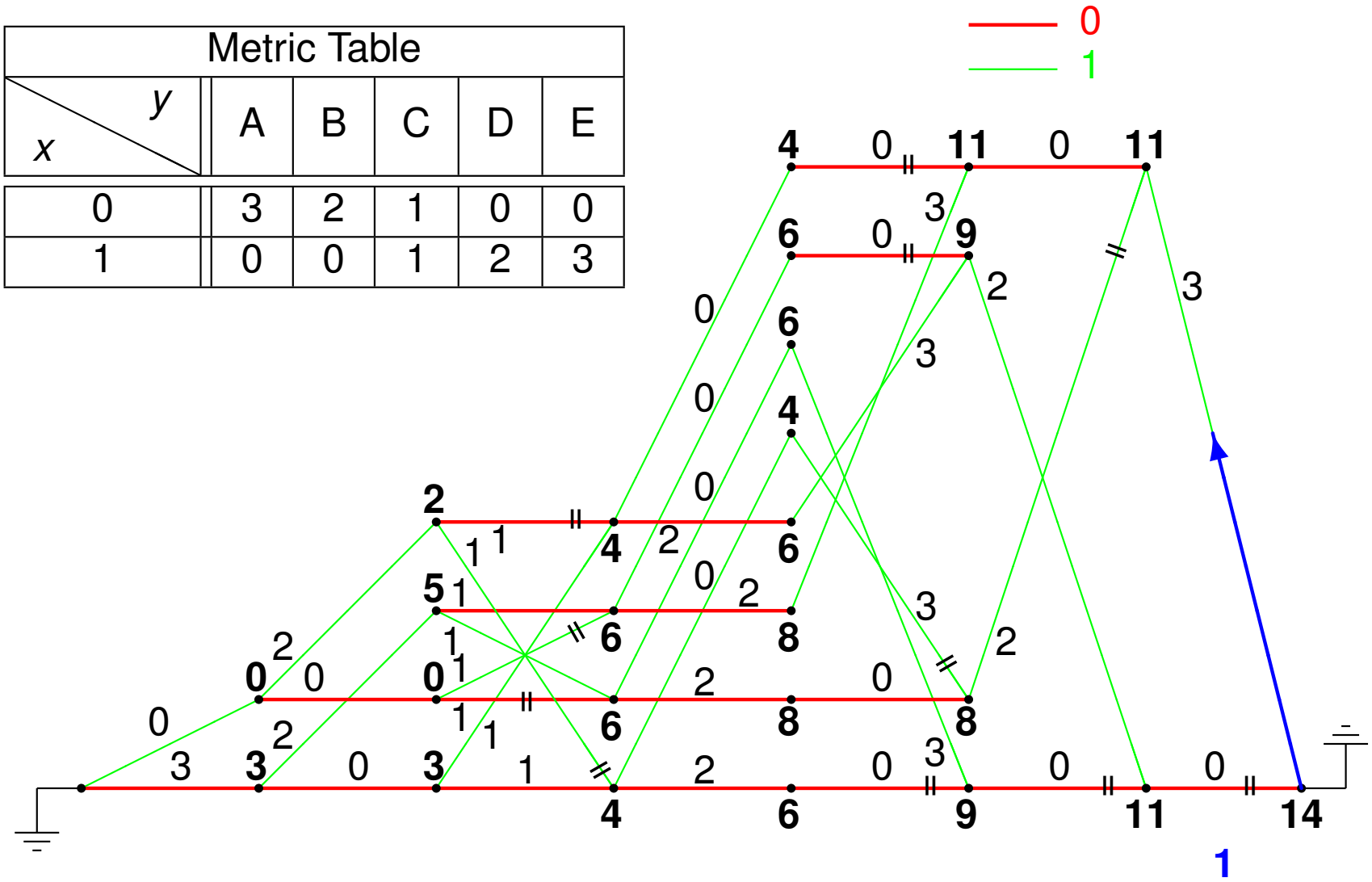
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

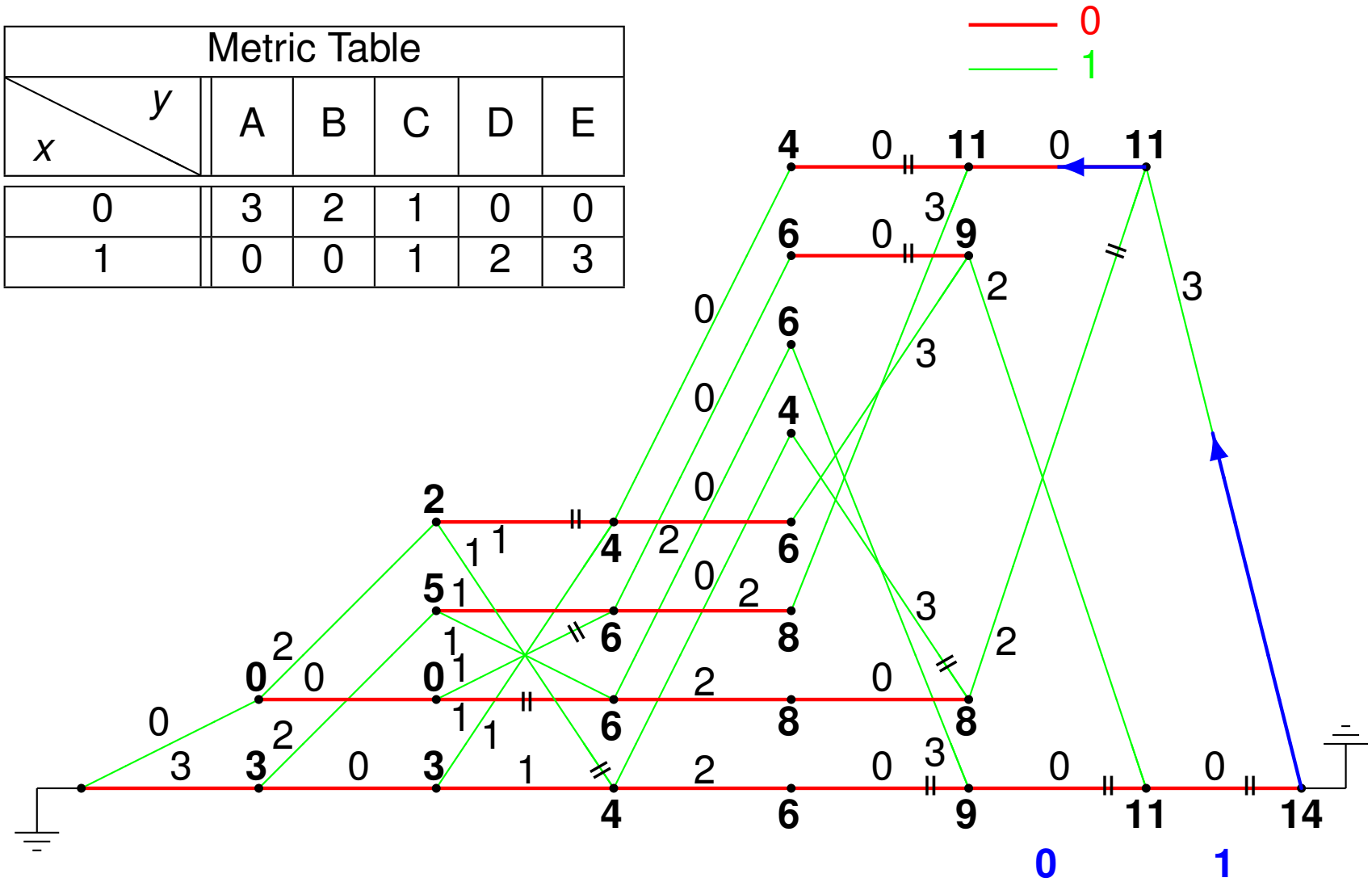
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

The Viterbi Decoder in Action

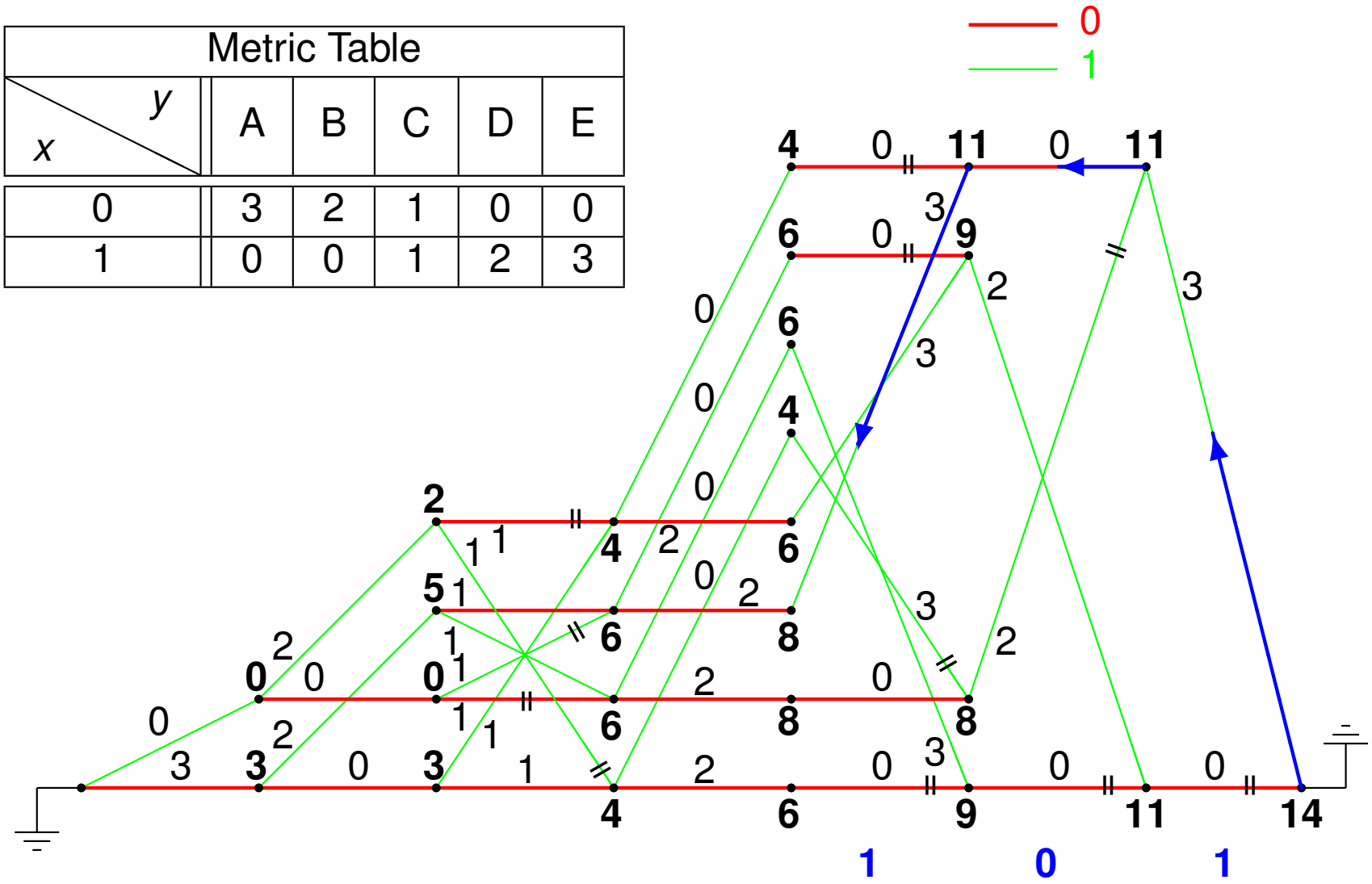
Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\boldsymbol{y} = ($ A, D, C, B, E, D, E)

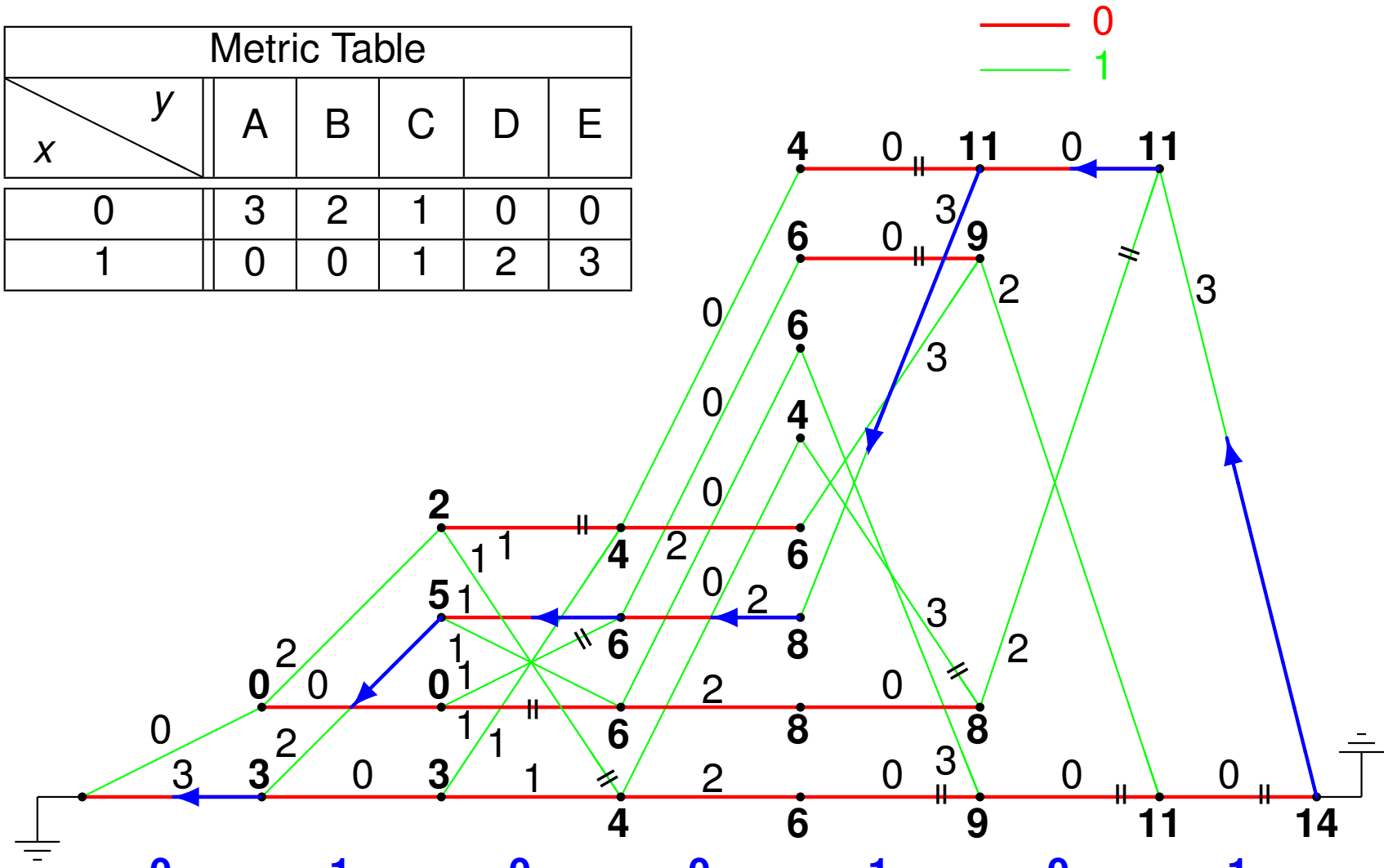
The Viterbi Decoder in Action

Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



The Viterbi Decoder in Action

Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



Complexity Analysis

- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all 2^{N-K} states in use at each trellis stage, i.e., $(N-1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length N who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N-1)$ additions, one comparison
- Best case: single parity-check code of length N, i.e., $\boldsymbol{H} = [11 \ldots 1]$, has 2^{N-1} codewords and hence the brute force decoder requires $(N-1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N-1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the bruten force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performancexs
- Can we design a code with a predictable, e.g., regular, trellis structure?

Complexity Analysis

- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all 2^{N-K} states in use at each trellis stage, i.e., $(N-1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length N who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N-1)$ additions, one comparison
- Best case: single parity-check code of length N, i.e., $\boldsymbol{H} = [11 \ldots 1]$, has 2^{N-1} codewords and hence the brute force decoder requires $(N-1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N-1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the bruten force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performancexs
- Can we design a code with a predictable, e.g., regular, trellis structure?

Complexity Analysis

- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all 2^{N-K} states in use at each trellis stage, i.e., $(N-1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length N who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N-1)$ additions, one comparison
- Best case: single parity-check code of length N, i.e., $\boldsymbol{H} = [11 \ldots 1]$, has 2^{N-1} codewords and hence the brute force decoder requires $(N-1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N-1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the bruten force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performancexs
- Can we design a code with a predictable, e.g., regular, trellis structure?

Complexity Analysis

- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all 2^{N-K} states in use at each trellis stage, i.e., $(N-1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length N who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N-1)$ additions, one comparison
- Best case: single parity-check code of length N, i.e., $\boldsymbol{H} = [11\ldots1]$, has 2^{N-1} codewords and hence the brute force decoder requires $(N-1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N-1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the bruten force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performancexs
- Can we design a code with a predictable, e.g., regular, trellis structure?

Complexity Analysis

- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all 2^{N-K} states in use at each trellis stage, i.e., $(N-1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length N who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N-1)$ additions, one comparison
- Best case: single parity-check code of length N, i.e., $\boldsymbol{H} = [11 \ldots 1]$, has 2^{N-1} codewords and hence the brute force decoder requires $(N-1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N-1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the bruten force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performancexs
- Can we design a code with a predictable, e.g., regular, trellis structure?

Complexity Analysis

- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all 2^{N-K} states in use at each trellis stage, i.e., $(N-1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length N who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N-1)$ additions, one comparison
- Best case: single parity-check code of length N, i.e., $\boldsymbol{H} = [11 \ldots 1]$, has 2^{N-1} codewords and hence the brute force decoder requires $(N-1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N-1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the bruten force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performancexs
- Can we design a code with a predictable, e.g., regular, trellis structure?

Complexity Analysis

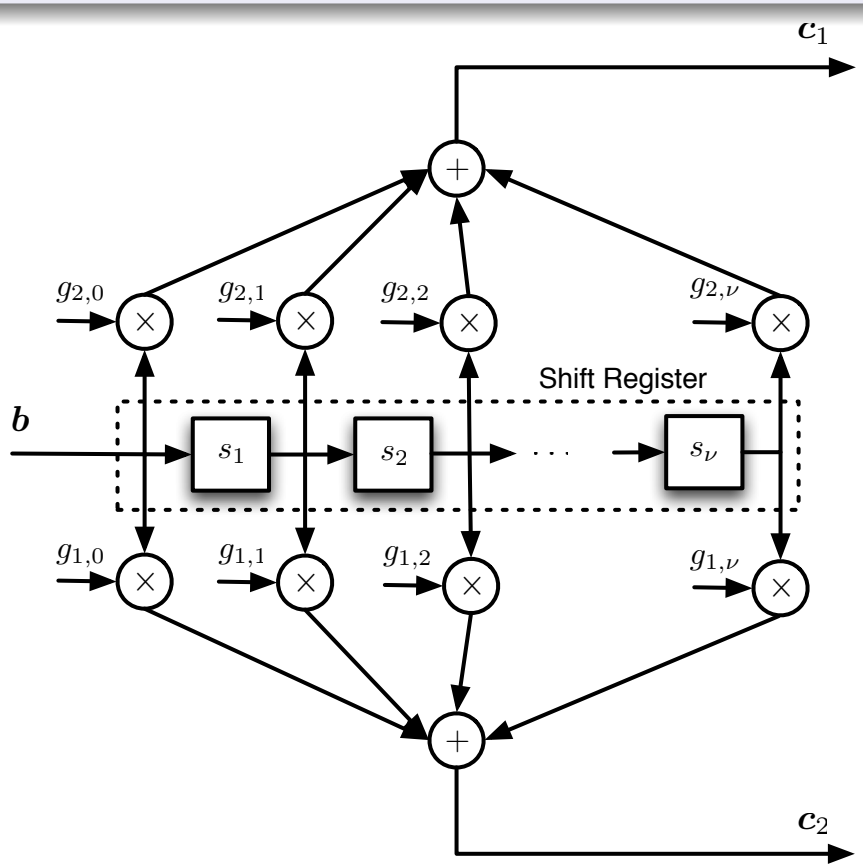
- In our example: 32 additions and 11 comparisons instead of 96 additions and 15 comparisons for the brute force decoder
- Trick question: is the performance of the Viterbi algorithm better or worse than that of the brute force decoder?
- In general, the complexity of the Viterbi decoder will depend on the structure of the code (how many codewords have common prefixes?)
- Worst case: no common prefixes, or all 2^{N-K} states in use at each trellis stage, i.e., $(N-1)2^{N-K}$ additions and $2^K - 1$ comparisons, complexity comparable to brute force. For example, $(N, 1)$ repetition code has a trellis with two parallel paths of length N who never merge, and therefore the same complexity for the brute force and Viterbi decoders, $2(N-1)$ additions, one comparison
- Best case: single parity-check code of length N, i.e., $\boldsymbol{H} = [11\ldots1]$, has 2^{N-1} codewords and hence the brute force decoder requires $(N-1)2^{N-1}$ additions, whereas the Viterbi decoder needs only $2(N-1)$ additions.
- In general, the complexity of the Viterbi algorithm will be much better than that of the bruten force decoder, but a random block code has variable, unpredictable number of states at each trellis stage which makes it difficult to predict performancexs
- Can we design a code with a predictable, e.g., regular, trellis structure?

Linear Convolutional Codes

Shift-Register Representation

Definitions

- Linear convolutional codes are defined as a finite state machine (FSM)
- Output $c_{i,j} = \bigoplus_{\ell=0}^{\nu} g_{i,\ell} b_{i-\ell}$ $i = 1, \ldots, N$ is the output generator index and j denotes the time index, where the symbol $\bigoplus$ denotes sum is in the binary field
- Common representations are: state diagrams and trellis

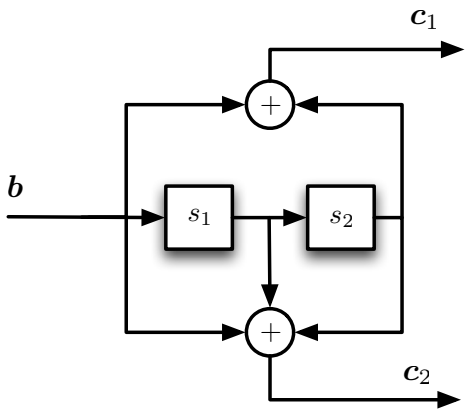


Linear Convolutional Codes

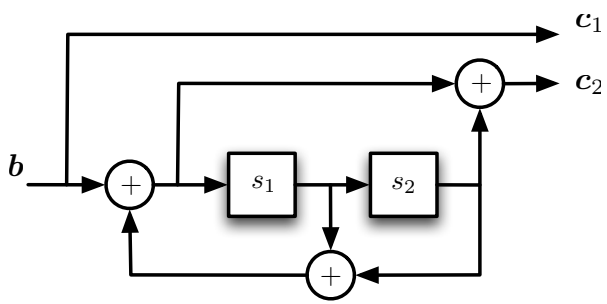
Shift-Register Representation

Example (4 states rate $R = \frac{1}{2}$)

- Rate $R = \frac{K}{N} = \frac{1}{2}$, number of states $2^{\nu} = 4$
- State: content of the shift register $\boldsymbol{s} = (s_1, s_2)$
- Generators (in octal form) $(5, 7)_8 = (101, 111)$



Non-recursive non-systematic

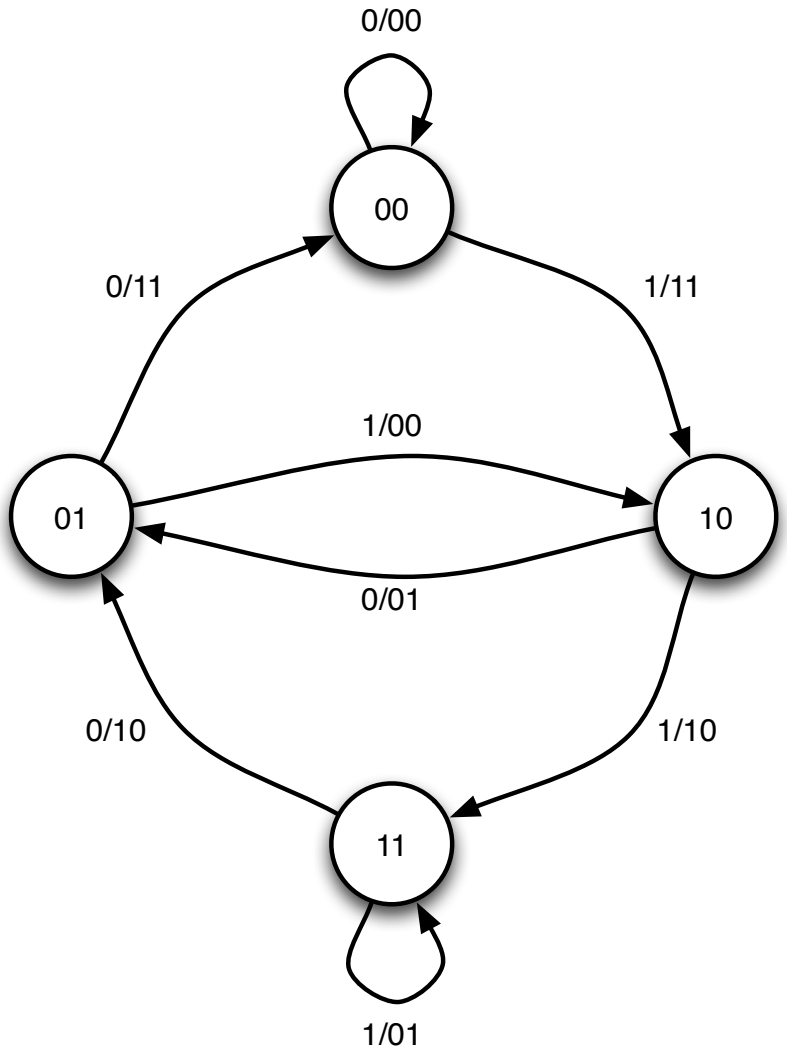


Recursive systematic

- Note that the state is not linked to the parity-check matrix as it was for linear block codes(there are many ways to define the state of a code)
- Note also that this encoder generates two code digits per state transition, as opposed to our previous trellis diagram for general linear block codes that generated only one code digit per state transition

Linear Convolutional Codes

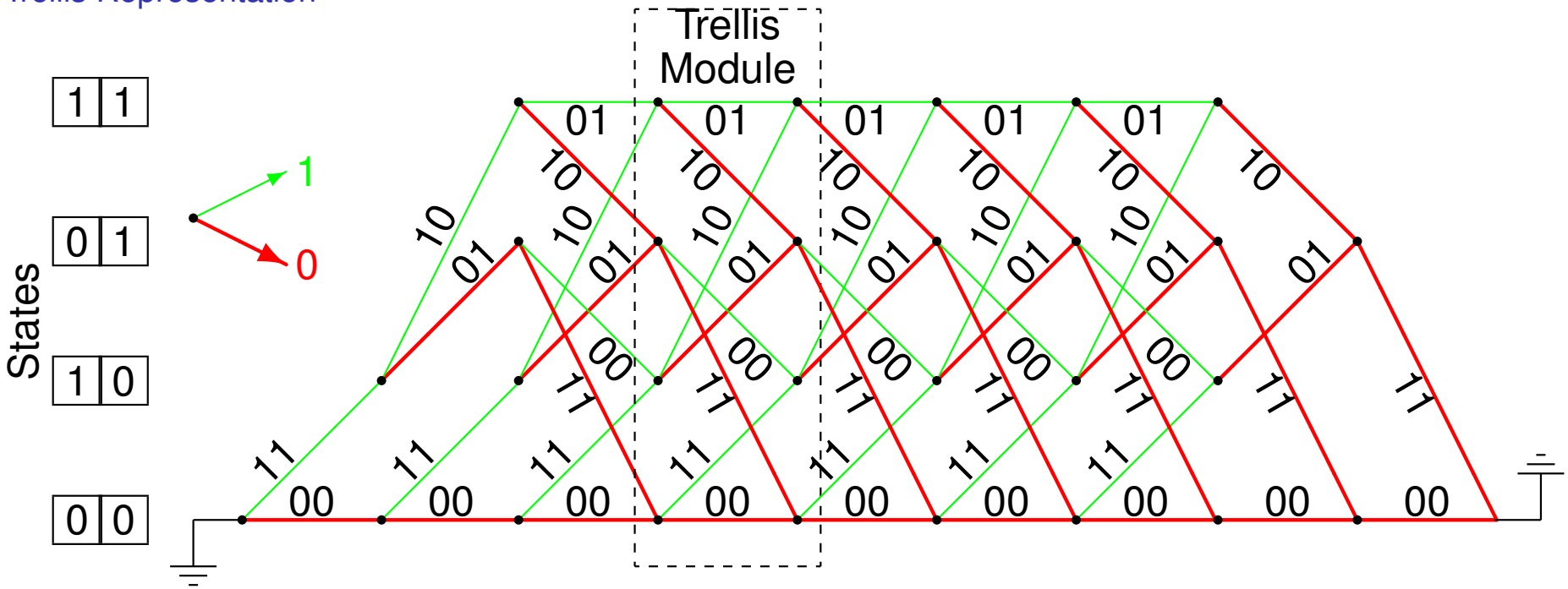
State Diagram



Notation: $b_i / c_{2i} c_{2i+1}$ where b_i is the encoded input bit corresponding to a state transition, and c_{2i}, c_{2i+1} are the resulting code digits for the state transition

Linear Convolutional Codes

Trellis Representation



Trellis of a Convolutional Code

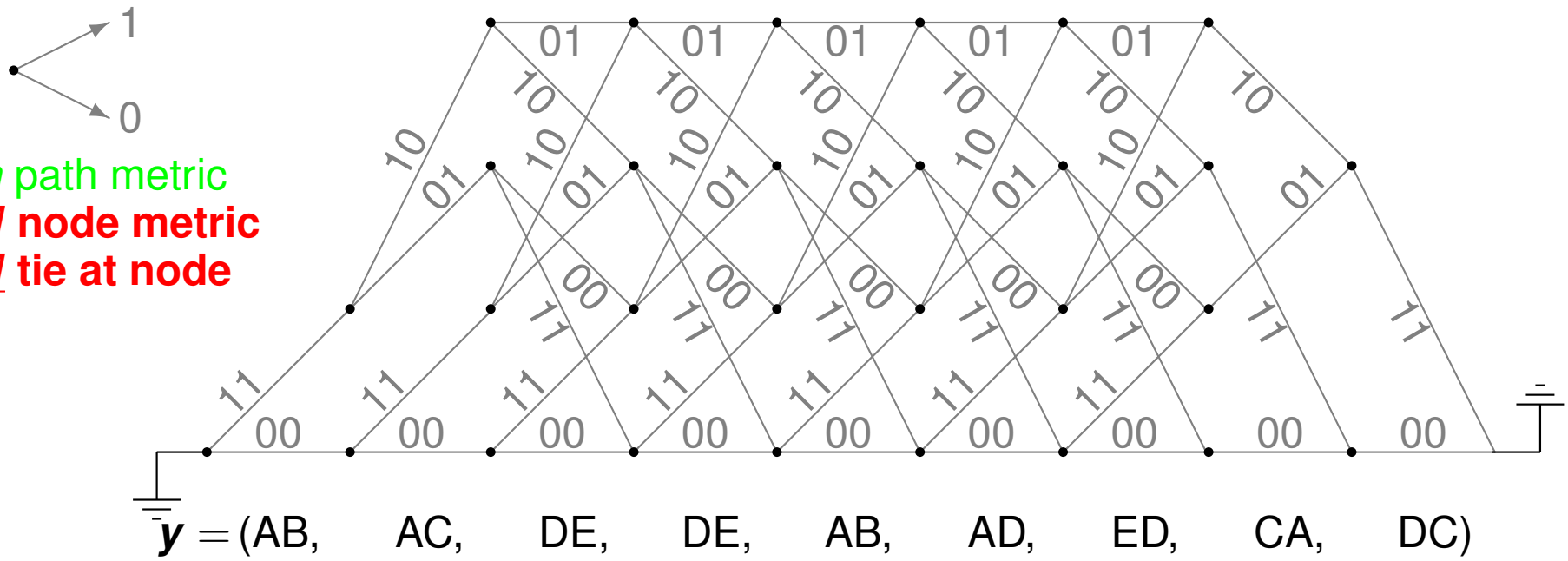
- Trellis consists of identical copies of the trellis module (except near root and toor)
- Trellis termination drives the FSM back to its all-zero state and thus encodes no information. For feedforward (non-recursive) convolutional encoders, this is achieved simply by padding zeros at the end of the information sequence.
- Note that states can always be ordered so the outgoing edge corresponding to a “1” is above the edge corresponding to a “0”, so there is no need for colour coding.

Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3

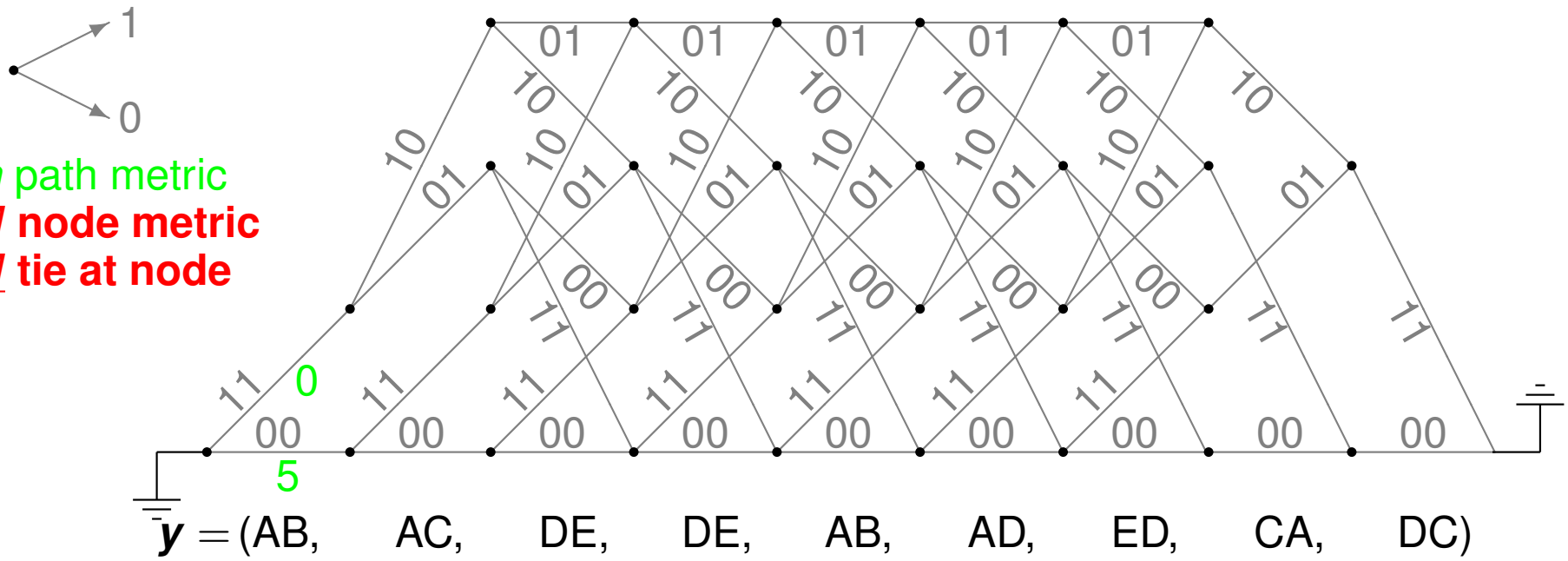


Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3

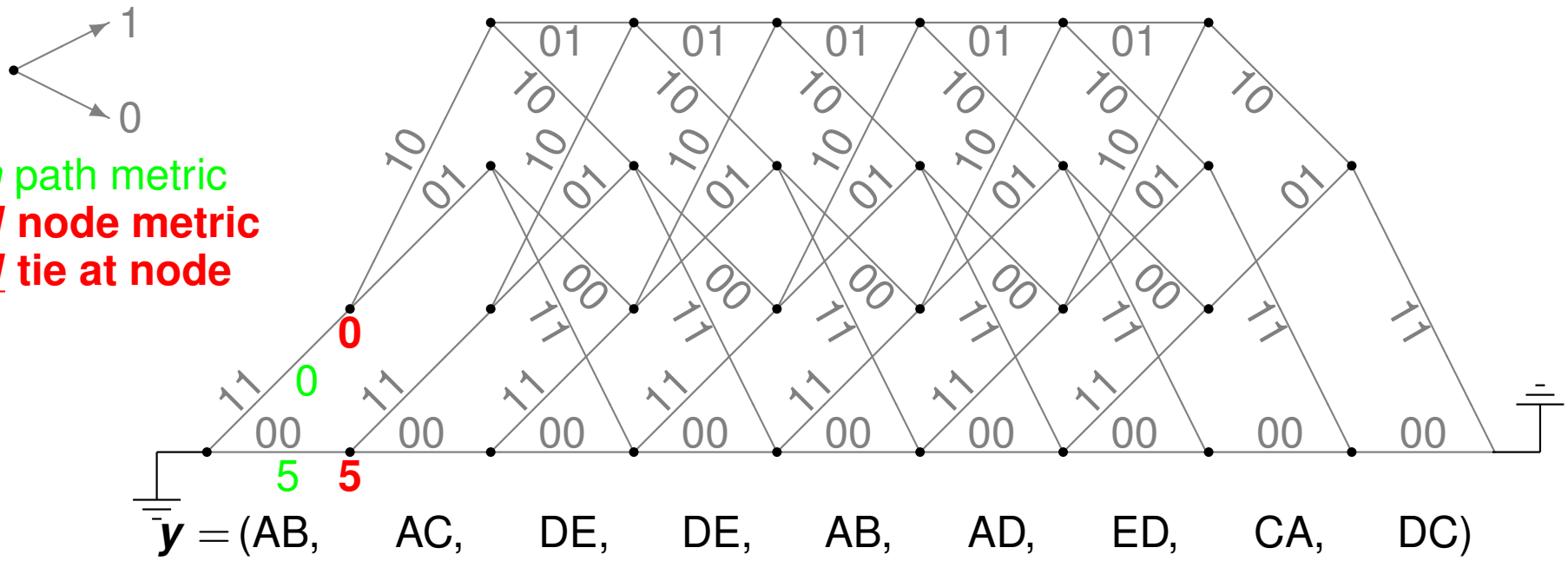


Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3

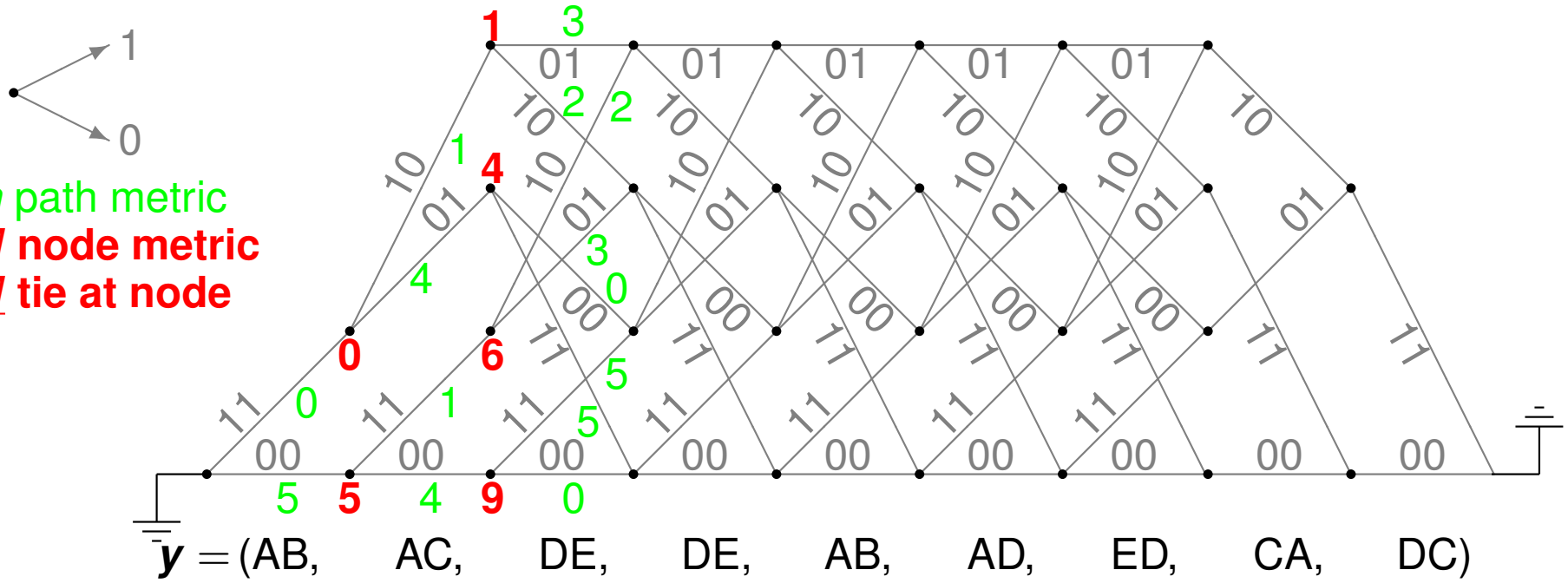


Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3

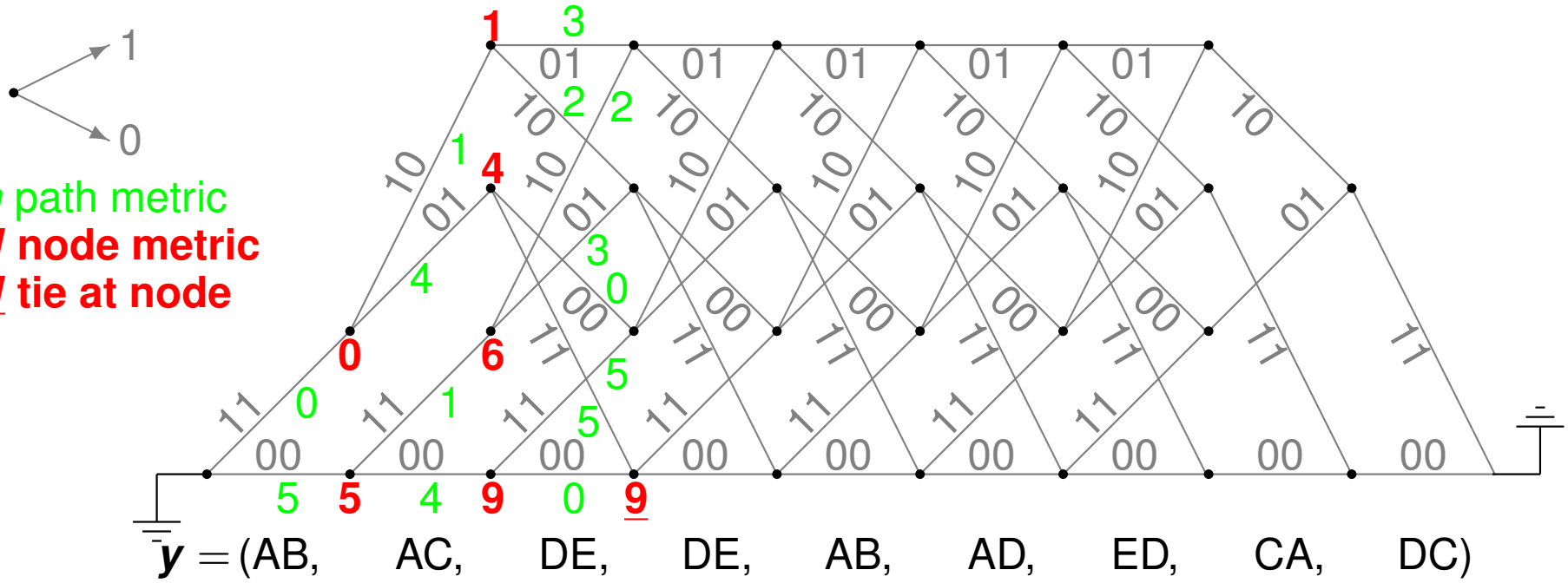


Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3

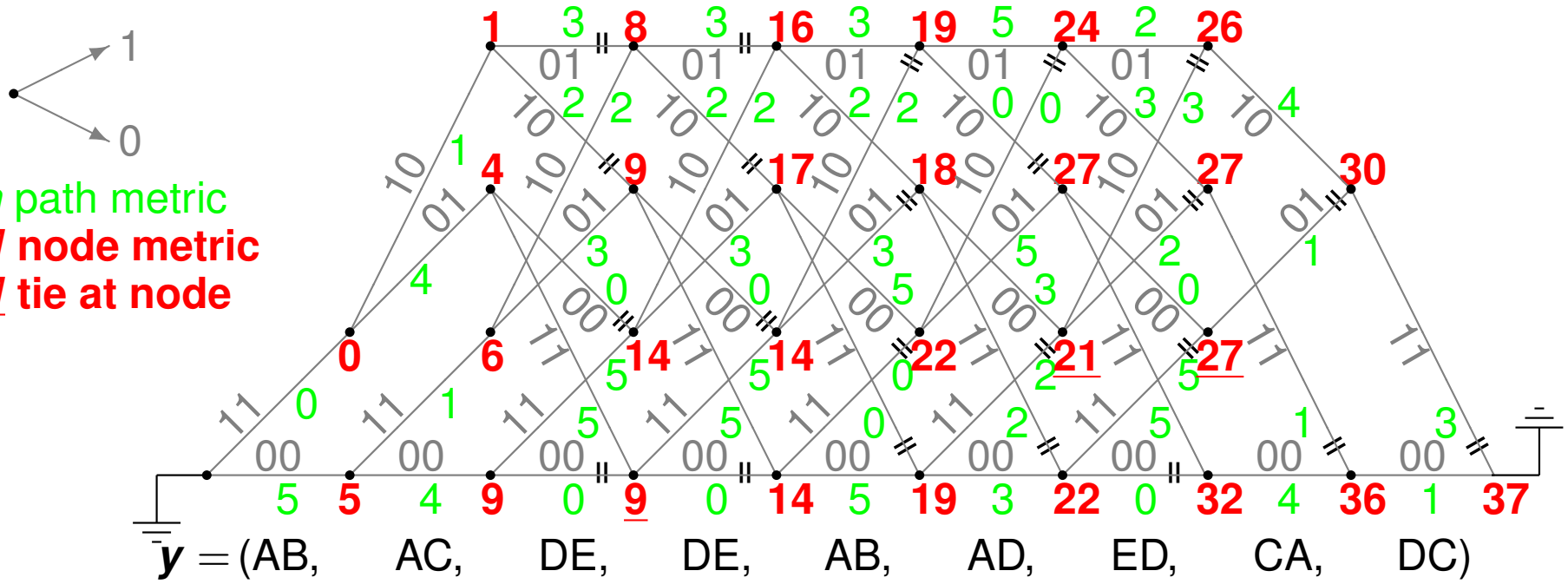


Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3

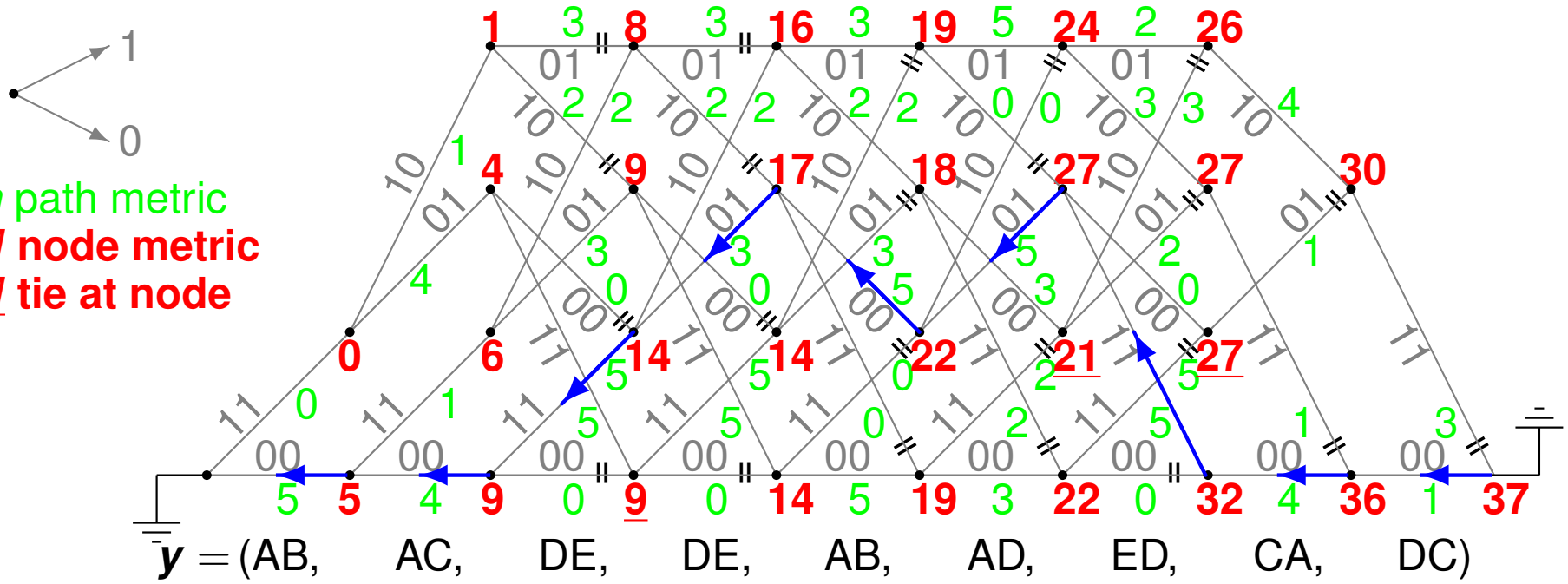


Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3

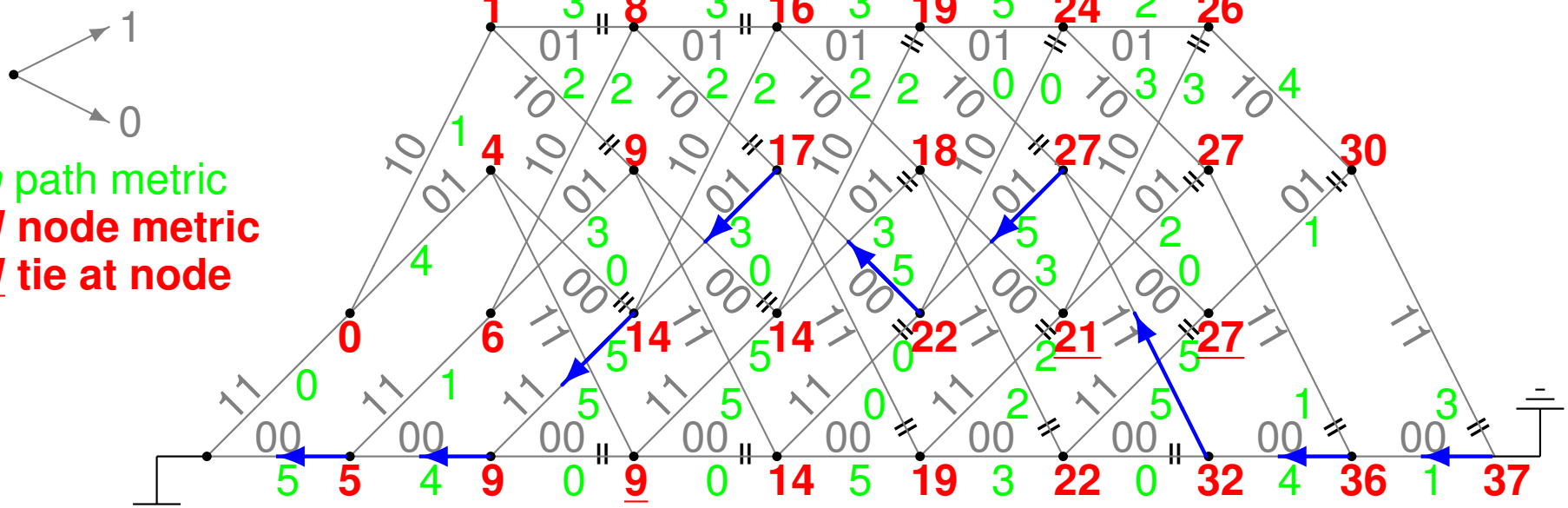


Viterbi Algorithm for Convolutional Codes

Example

- Transmission over the channel defined previously
- Received sequence ABACDEDEABADEDCADC

Additive Metric Table					
x \ y	A	B	C	D	E
0	3	2	1	0	0
1	0	0	1	2	3



$\bar{\boldsymbol{y}} = (\text{AB}, \text{AC}, \text{DE}, \text{DE}, \text{AB}, \text{AD}, \text{ED}, \text{CA}, \text{DC})$

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

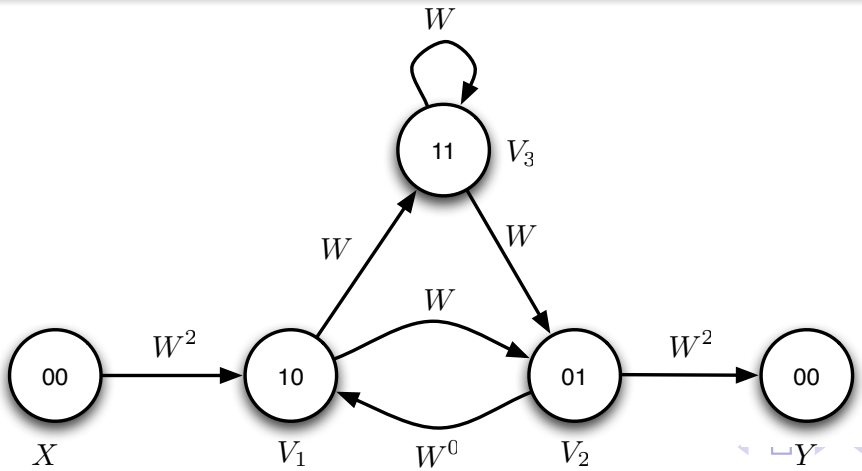
No ties on winning path: ML solution is unique

Transfer Function and Distance Spectrum

- The distance spectrum determines the error probability performance
- The distance spectrum can be determined from the transfer function
- To derive the transfer function, we split the state diagram from and to the zero state, labeling the Hamming weight of each transition with a dummy polynomial, i.e., a transition with weight 2 is labelled W^2
- This will count (simple) paths leaving and merging back into the zero state
- Label each state with a variable and solve $T(W) = \frac{Y}{X}$

$$V_1 = W^2 X, \quad V_2 = WV_1 + WV_3, \quad Y = W^2 V_2, \quad V_3 = WV_1 + WV_3$$

$$T(W) = \frac{Y}{X} = \sum_d T_d W^d = \frac{W^5}{1-2W} = W^5 + 2W^6 + 4W^7 + 8W^8 + 16W^9 + \ldots$$

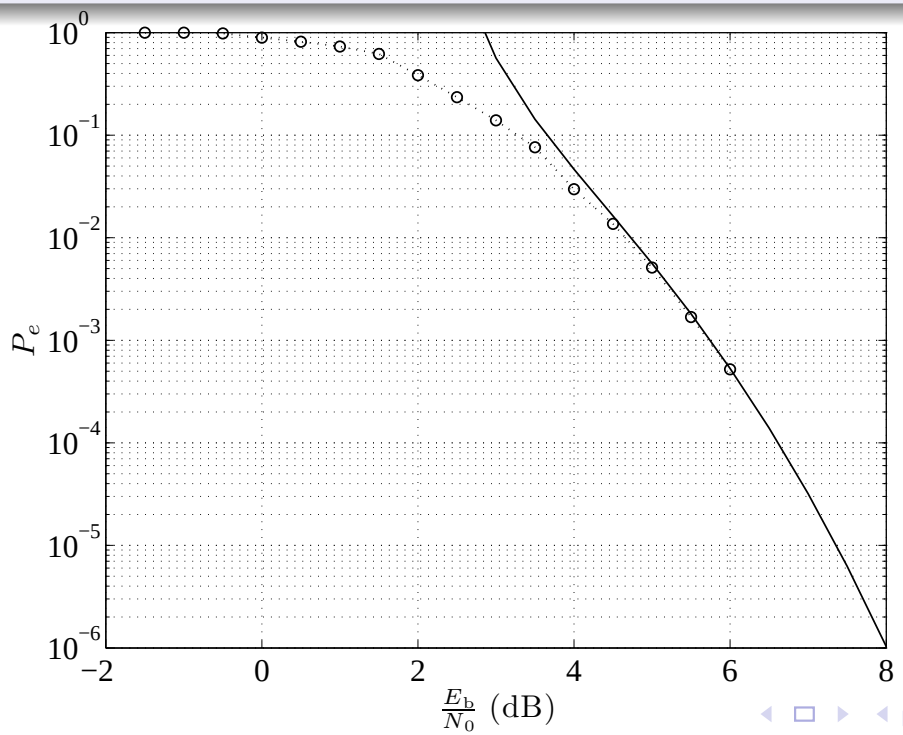


Error Probability and Union Bound

- The minimum power is the *free* distance $d_{\text{free}} = 5$
- It is not difficult to show that $A_d \leq LT_d$
- Hence, we can bound P_e as

$$P_e \leq \sum_d LT_d Q\left(\sqrt{2d\,\text{SNR}}\right)$$

  where T_d are obtained from the transfer function
- The high-SNR performance is dominated by d_{free}



Convolutional Codes

What we have learned. . .

- Unlike block codes, convolutional codes have predictable complexity for Viterbi decoding irrespective of code length
- However, performance does not improve with code length to the point of achieving arbitrary reliability at rates approaching capacity. Performance is limited instead by the shortest distance paths in the trellis (the free distance).
- Can we combine the benefits of block codes in terms of performance with the advantages of convolutional codes in terms of decoding complexity?

Convolutional Codes

What we have learned. . .

- Unlike block codes, convolutional codes have predictable complexity for Viterbi decoding irrespective of code length
- However, performance does not improve with code length to the point of achieving arbitrary reliability at rates approaching capacity. Performance is limited instead by the shortest distance paths in the trellis (the free distance).
- Can we combine the benefits of block codes in terms of performance with the advantages of convolutional codes in terms of decoding complexity?

Convolutional Codes

What we have learned. . .

- Unlike block codes, convolutional codes have predictable complexity for Viterbi decoding irrespective of code length
- However, performance does not improve with code length to the point of achieving arbitrary reliability at rates approaching capacity. Performance is limited instead by the shortest distance paths in the trellis (the free distance).
- Can we combine the benefits of block codes in terms of performance with the advantages of convolutional codes in terms of decoding complexity?

Back to basics: symbol decoding on a trellis

Forward-Backward Decoding

- Bitwise maximum a posteriori (MAP)

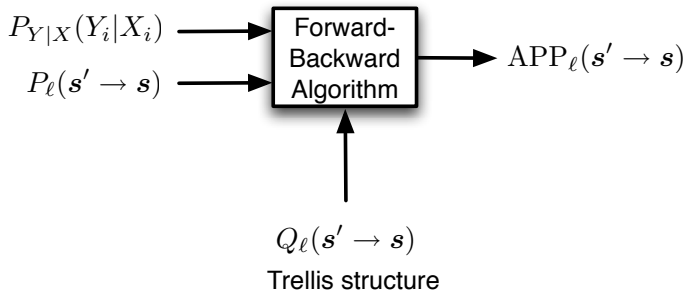
$$\hat{b}_i = \arg\max_{b_i=0,1} P_{B|\boldsymbol{Y}}(b_j|\boldsymbol{y}) = \arg\max_{b_i=0,1} P_{B,\boldsymbol{Y}}(b_j,\boldsymbol{y}) = \arg\max_{b_i=0,1} \sum_{\boldsymbol{s},\boldsymbol{s}'|\boldsymbol{x}_\ell(\boldsymbol{s}',\boldsymbol{s})} \mathrm{APP}_\ell(\boldsymbol{s}' \to \boldsymbol{s})$$

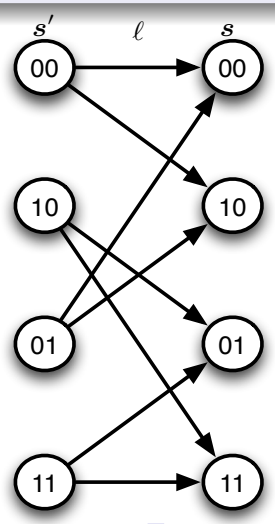
- $\mathrm{APP}_\ell(\boldsymbol{s}' \to \boldsymbol{s})$ is the a posteriori probability of the transition $\boldsymbol{s}' \to \boldsymbol{s}$ at trellis step ℓ
- Forward-backward algorithm calculates $\mathrm{APP}_\ell(\boldsymbol{s}' \to \boldsymbol{s})$
- A posteriori probabilities for input/output bits of the transition are easily obtained from $\mathrm{APP}_\ell(\boldsymbol{s}' \to \boldsymbol{s})$

Forward-Backward Decoding

Forward-Backward Decoding Algorithm

- Input
  - $P_{Y|X}(y_i|x_i)$: channel transition probability
  - $P_\ell(\boldsymbol{s}' \to \boldsymbol{s})$: a priori probability of the transition $\boldsymbol{s}' \to \boldsymbol{s}$ at trellis step $\ell = 1, \ldots, L$
  - $Q_\ell(\boldsymbol{s}', \boldsymbol{s}) = 1$ if the transition $\boldsymbol{s}' \to \boldsymbol{s}$ exists, 0 otherwise
- Output
  - $\mathrm{APP}_\ell(\boldsymbol{s}' \to \boldsymbol{s})$: a posteriori probability of the transition $\boldsymbol{s}' \to \boldsymbol{s}$ at trellis step ℓ
- Variables
  - $\gamma_\ell(\boldsymbol{s}', \boldsymbol{s}) = \Pr\{\boldsymbol{S}_\ell = \boldsymbol{s}, y_\ell | \boldsymbol{S}_{\ell-1} = \boldsymbol{s}'\}$: the metric used by the algorithm
  - $\alpha_\ell(\boldsymbol{s}') = \Pr\{\boldsymbol{S}_\ell = \boldsymbol{s}', \boldsymbol{y}_1^\ell\}$: joint probability of state $\boldsymbol{s}$ and the observation from 1 to ℓ
  - $\beta_\ell(\boldsymbol{s}') = \Pr\{\boldsymbol{y}_{\ell+1}^L | \boldsymbol{S}_\ell = \boldsymbol{s}'\}$: probability of the observation from $\ell + 1$ to L given state $\boldsymbol{s}'$





Forward-Backward Decoding

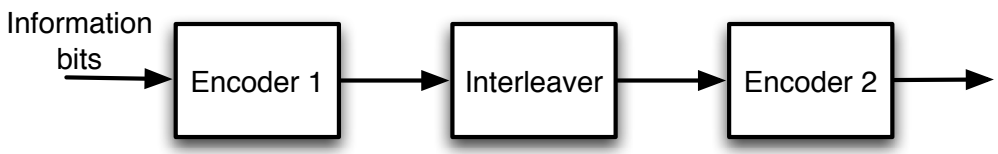
Forward-Backward Decoding Algorithm

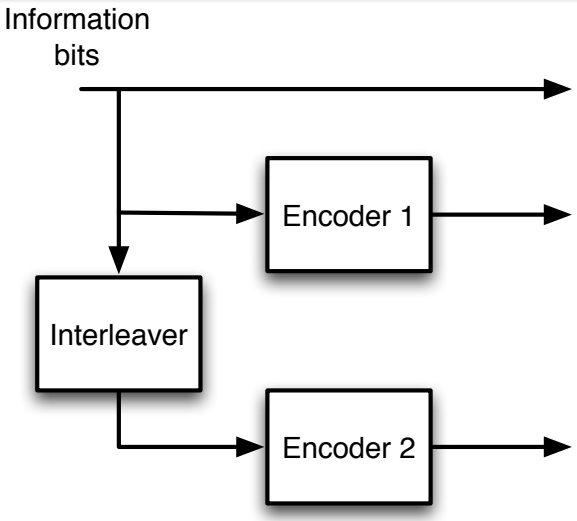
1. Initialisation
   - $\alpha_0(\mathbf{0}) = 1$ and $\alpha_0(\boldsymbol{s}) = 0$ for $\boldsymbol{s} \neq \mathbf{0}$
   - $\beta_L(\mathbf{0}) = 1$ and $\beta_L(\boldsymbol{s}) = 0$ for $\boldsymbol{s} \neq \mathbf{0}$
   - $\gamma_\ell(\boldsymbol{s}', \boldsymbol{s}) = Q_\ell(\boldsymbol{s}', \boldsymbol{s}) \times P_\ell(\boldsymbol{s}' \to \boldsymbol{s}) \times P_{\boldsymbol{Y}|\boldsymbol{X}}(\boldsymbol{y}_\ell|\boldsymbol{x}_\ell)$
2. Forward step
   - $\alpha_\ell(\boldsymbol{s}) = \sum_{\boldsymbol{s}'} \gamma_\ell(\boldsymbol{s}', \boldsymbol{s}) \times \alpha_{\ell-1}(\boldsymbol{s}')$ for $\ell = 1, \ldots, L$
3. Backward step
   - $\beta_\ell(\boldsymbol{s}') = \sum_{\boldsymbol{s}} \gamma_{\ell+1}(\boldsymbol{s}', \boldsymbol{s}) \times \beta_{\ell+1}(\boldsymbol{s})$ for $\ell = L-1, \ldots, 0$
4. APP computation
   - $\text{APP}_\ell(\boldsymbol{s}' \to \boldsymbol{s}) = \alpha_{\ell-1}(\boldsymbol{s}') \times \gamma_\ell(\boldsymbol{s}', \boldsymbol{s}) \times \beta_\ell(\boldsymbol{s})$

Turbo-Codes

Basic Structure

- Turbo-codes (Berrou, Glavieux & Thitimajshima, 1993) consist of the concatenation of 2 component codes through an interleaver permutation
- The concatenation can be serial or parallel (original turbo-code, UMTS turbo-code)
- Component encoders typically recursive convolutional codes
- Bad minimum distance (improves with length), very low number of codewords at minimum distance
- Performance improves with length (longer interleaver, more randomness)
- It can be proved that for $n \to \infty$, turbo-codes have a threshold SNR_{th}, such that for $\text{SNR} > \text{SNR}_{\text{th}}$ then $P_b \to 0$ (asymptotically good codes)

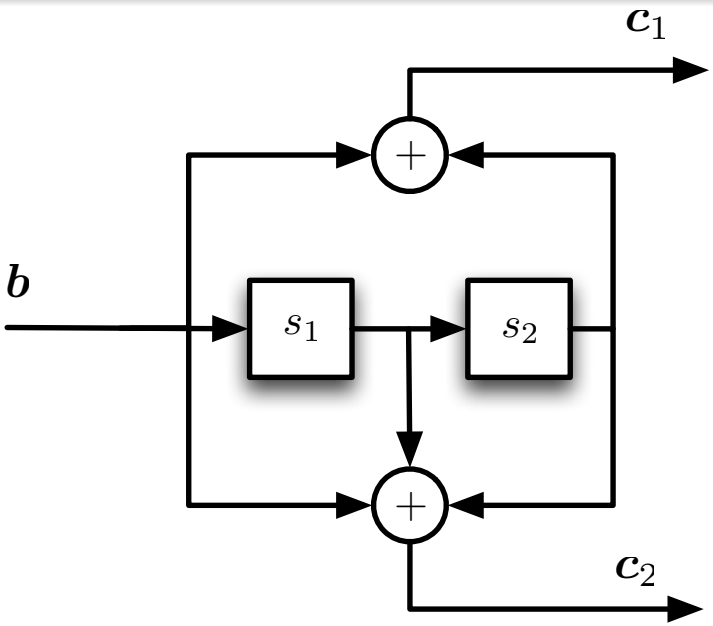


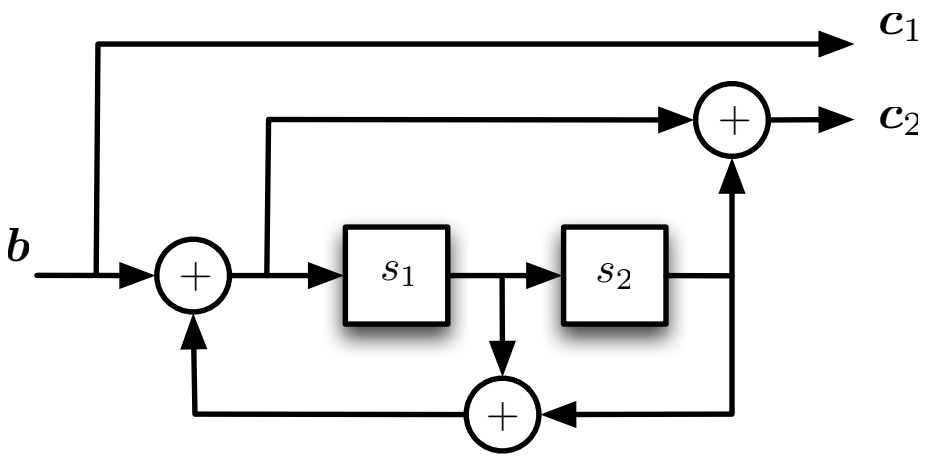


Turbo-Codes

Back to convolutional codes

- The component encoders of parallel turbo-codes, or the inner encoder of serial turbo-codes are recursive.
- Note that $(5,7)_8 = 7 \times \left(\frac{5}{7},1\right)_8$ (see figure)
- The impulse response of a non-recursive convolutional encoder vanishes after ν trellis steps (same as trellis termination)
- The impulse response of a recursive convolutional encoder settles into a periodic state (infinite impulse response)
- Interestingly, the two encoders represent the same code (same set of codewords)

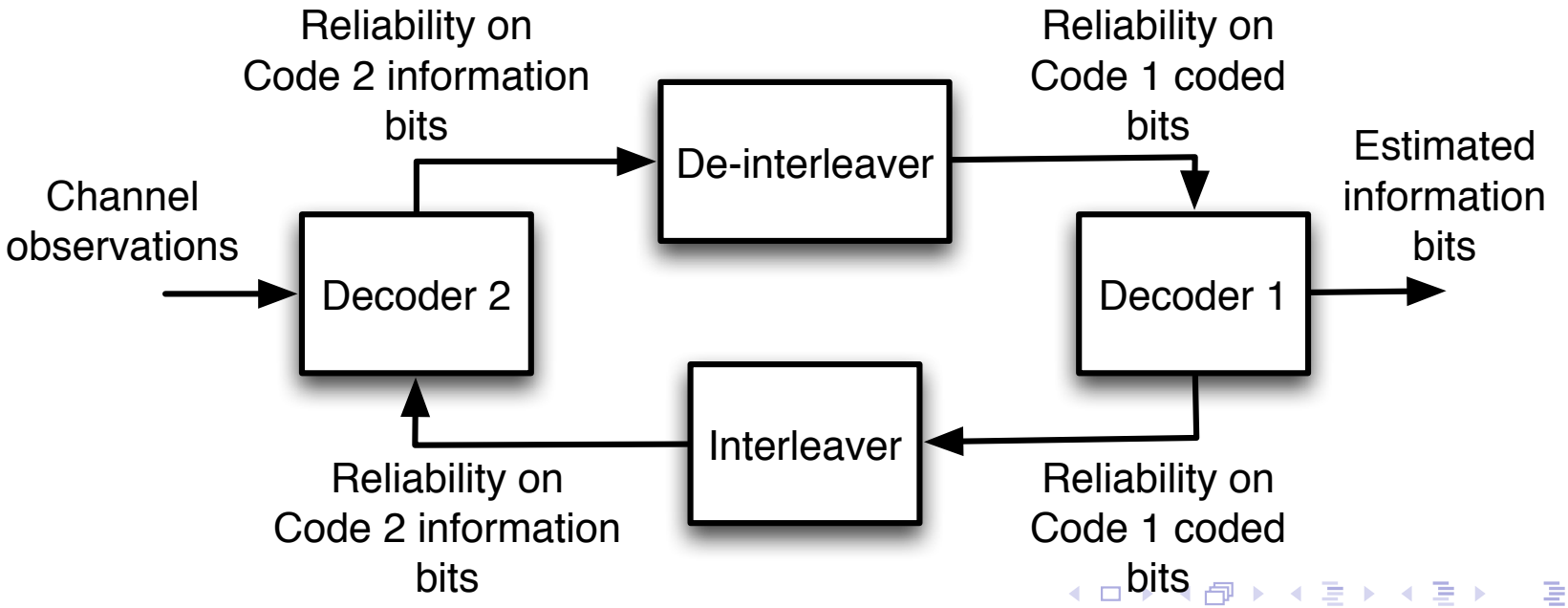




Turbo-Codes

Decoding

- Due to the presence of interleaver, maximum likelihood is only possible through exhaustive search (too complex)
- Viterbi decoding of individual codes is not appropriate: hard decisions on word
- Forward-backward (or BCJR) used instead: soft decisions on individual bits
- We use an iterative (turbo) decoder that exchanges reliability messages (soft values) about the bits over the iterations
- For serial turbo-codes, the decoder is given below



Setting the historical record straight

- Convolutional codes were proposed by Peter Elias in the 1950s
- The Viterbi algorithm was proposed by Andrew J. Viterbi in 1966 and shown by Dave Forney to provide the ML solution in 1973. Forney also came up with the graphical representation of a code trellis. The Viterbi algorithm is an instance of dynamic programming for which algorithms existed in the mathematical optimisation literature.
- The forward-backward algorithm was known in the hidden Markov models literature and introduced to the communications community by Bahl, Cocke, Jelinek and Raviv (BCJR) in a 1974 paper.
- The method for constructing the trellis of a block codes presented in these notes was introduced by BCJR in the less well-known but more original second part of their famous 1974 paper.
- Turbo codes were presented by Berrou, Glavieux and Thitimajshima in 1993 at the International Communications Conference and constitute arguably the most important contribution to the information theory literature since Shannon's 1948 paper. It was later discovered by MacKay that Low-Density Parity-Check codes, invented along with their iterative decoding technique by Bob Gallager in 1962, achieve similar performance to that of Turbo Codes.

Setting the historical record straight

- Convolutional codes were proposed by Peter Elias in the 1950s
- The Viterbi algorithm was proposed by Andrew J. Viterbi in 1966 and shown by Dave Forney to provide the ML solution in 1973. Forney also came up with the graphical representation of a code trellis. The Viterbi algorithm is an instance of dynamic programming for which algorithms existed in the mathematical optimisation literature.
- The forward-backward algorithm was known in the hidden Markov models literature and introduced to the communications community by Bahl, Cocke, Jelinek and Raviv (BCJR) in a 1974 paper.
- The method for constructing the trellis of a block codes presented in these notes was introduced by BCJR in the less well-known but more original second part of their famous 1974 paper.
- Turbo codes were presented by Berrou, Glavieux and Thitimajshima in 1993 at the International Communications Conference and constitute arguably the most important contribution to the information theory literature since Shannon's 1948 paper. It was later discovered by MacKay that Low-Density Parity-Check codes, invented along with their iterative decoding technique by Bob Gallager in 1962, achieve similar performance to that of Turbo Codes.

Setting the historical record straight

- Convolutional codes were proposed by Peter Elias in the 1950s
- The Viterbi algorithm was proposed by Andrew J. Viterbi in 1966 and shown by Dave Forney to provide the ML solution in 1973. Forney also came up with the graphical representation of a code trellis. The Viterbi algorithm is an instance of dynamic programming for which algorithms existed in the mathematical optimisation literature.
- The forward-backward algorithm was known in the hidden Markov models literature and introduced to the communications community by Bahl, Cocke, Jelinek and Raviv (BCJR) in a 1974 paper.
- The method for constructing the trellis of a block codes presented in these notes was introduced by BCJR in the less well-known but more original second part of their famous 1974 paper.
- Turbo codes were presented by Berrou, Glavieux and Thitimajshima in 1993 at the International Communications Conference and constitute arguably the most important contribution to the information theory literature since Shannon's 1948 paper. It was later discovered by MacKay that Low-Density Parity-Check codes, invented along with their iterative decoding technique by Bob Gallager in 1962, achieve similar performance to that of Turbo Codes.

Setting the historical record straight

- Convolutional codes were proposed by Peter Elias in the 1950s
- The Viterbi algorithm was proposed by Andrew J. Viterbi in 1966 and shown by Dave Forney to provide the ML solution in 1973. Forney also came up with the graphical representation of a code trellis. The Viterbi algorithm is an instance of dynamic programming for which algorithms existed in the mathematical optimisation literature.
- The forward-backward algorithm was known in the hidden Markov models literature and introduced to the communications community by Bahl, Cocke, Jelinek and Raviv (BCJR) in a 1974 paper.
- The method for constructing the trellis of a block codes presented in these notes was introduced by BCJR in the less well-known but more original second part of their famous 1974 paper.
- Turbo codes were presented by Berrou, Glavieux and Thitimajshima in 1993 at the International Communications Conference and constitute arguably the most important contribution to the information theory literature since Shannon's 1948 paper. It was later discovered by MacKay that Low-Density Parity-Check codes, invented along with their iterative decoding technique by Bob Gallager in 1962, achieve similar performance to that of Turbo Codes.

Setting the historical record straight

- Convolutional codes were proposed by Peter Elias in the 1950s
- The Viterbi algorithm was proposed by Andrew J. Viterbi in 1966 and shown by Dave Forney to provide the ML solution in 1973. Forney also came up with the graphical representation of a code trellis. The Viterbi algorithm is an instance of dynamic programming for which algorithms existed in the mathematical optimisation literature.
- The forward-backward algorithm was known in the hidden Markov models literature and introduced to the communications community by Bahl, Cocke, Jelinek and Raviv (BCJR) in a 1974 paper.
- The method for constructing the trellis of a block codes presented in these notes was introduced by BCJR in the less well-known but more original second part of their famous 1974 paper.
- Turbo codes were presented by Berrou, Glavieux and Thitimajshima in 1993 at the International Communications Conference and constitute arguably the most important contribution to the information theory literature since Shannon's 1948 paper. It was later discovered by MacKay that Low-Density Parity-Check codes, invented along with their iterative decoding technique by Bob Gallager in 1962, achieve similar performance to that of Turbo Codes.