Digital Communications
SSY125, Lectures 10 and 11

Convolutional Codes
(Chapter 9)

Alexandre Graell i Amat
alexandre.graell@chalmers.se
https://sites.google.com/site/agraellamat

November 22 and November 27, 2023

CHALMERS

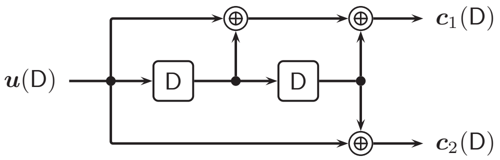
Convolutional codes

- Invented by Peter Elias in 1955.
- Widely used in wireless networks, satellite and spacecraft links, and terrestrial broadcast communications since the 1970s.
- Relatively low-complexity ML decoding algorithm (the Viterbi algorithm) and excellent performance when concatenated with block codes (e.g., Reed-Solomon codes).
- The main ingredients of turbo-like codes, one of the state-of-the-art coding schemes (included in most of the current communication standards).

Convolutional codes

- Introduce memory $\longrightarrow$ The n code bits that the encoder generates in correspondence to the k information bits at its input depend also on previous information bits.
- Stream-oriented in nature: the encoder encodes a potentially infinitely long sequence of information bits into an infinitely long sequence of code bits.

The convolutional code archetype

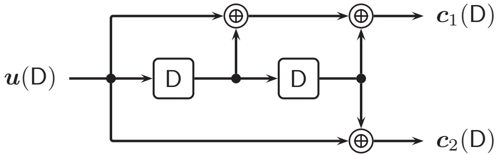


- A convolutional encoder can be regarded as a finite-state machine.
- The information sequence $\boldsymbol{u} = (u_1, u_2, \ldots)$ has potentially infinite length.
- The encoder generates two sequences,

$$\boldsymbol{c}_1 = (c_{1,1}, c_{1,2}, \ldots) \quad \text{and} \quad \boldsymbol{c}_2 = (c_{2,1}, c_{2,2}, \ldots),$$

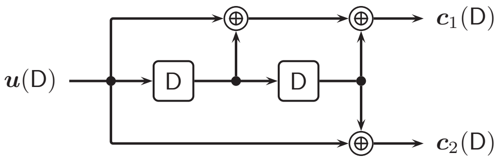
- The code is defined by parameters $(n, k) \longrightarrow$ for each k input bits the convolutional encoder generates n output bits. The code rate is $R_\mathsf{c} = k/n$.
- Convolutional code archetype: At each time instant i, two coded bits, $c_{1,i}$ and $c_{2,i}$, are produced for each information bit $u_i \longrightarrow (n, k) = (2, 1)$ and $R_\mathsf{c} = k/n = 1/2$.

The convolutional code archetype



- The encoder has two memory elements, which take values on $\{0,1\} \longrightarrow$ The encoder has 4 states $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$.
- Memory of the encoder, ν: The number of memory elements of the convolutional encoder.
- State of the encoder at time instant i: $\boldsymbol{s}_i \in \{0,1\}^\nu$.
- The state of the encoder at time instant $i+1$, $\boldsymbol{s}_{i+1}$, is a deterministic function of $\boldsymbol{s}_i$ and u_i.
- The code bits at time instant i, $c_{1,i}$ and $c_{2,i}$, are deterministic functions of $\boldsymbol{s}_i$ and u_i.

The convolutional code archetype



- The top and bottom parts acts as two discrete-time finite-impulse response filters with binary operations: The top filter has impulse response $\boldsymbol{g}_1 = (1, 1, 1)$, while the bottom filter has impulse response $\boldsymbol{g}_2 = (1, 0, 1)$.
- $\boldsymbol{c}_1 = (c_{1,1}, c_{1,2} \ldots)$ and $\boldsymbol{c}_2 = (c_{2,1}, c_{2,2} \ldots)$ can then be obtained as the convolution of $\boldsymbol{u} = (u_1, u_2, \ldots)$ and $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$,

$$\boldsymbol{c}_1 = \boldsymbol{u} * \boldsymbol{g}_1,$$
$$\boldsymbol{c}_2 = \boldsymbol{u} * \boldsymbol{g}_2.$$

- For an $(n, 1)$ convolutional code

$$\boldsymbol{c}_j = \boldsymbol{u} * \boldsymbol{g}_j, \quad j = 1, \ldots, n,$$

Encoding using the D-transform

- Define the D-transforms

$$\boldsymbol{u}(\mathsf{D}) = \sum_i u_i \mathsf{D}^i, \qquad \boldsymbol{c}(\mathsf{D}) = \sum_i c_i \mathsf{D}^i, \qquad \boldsymbol{g}(\mathsf{D}) = \sum_i g_i \mathsf{D}^i.$$

- $\boldsymbol{u}$, $\boldsymbol{c}$, and $\boldsymbol{g}$ can be obtained from the coefficients of $\boldsymbol{u}(\mathsf{D})$, $\boldsymbol{c}(\mathsf{D})$, and $\boldsymbol{g}(\mathsf{D})$.

- Given $\boldsymbol{u} = (1, 0, 0, 1, 1, \ldots)$ the corresponding D-transform is

$$\boldsymbol{u}(\mathsf{D}) = 1 \cdot \mathsf{D}^0 + 0 \cdot \mathsf{D}^1 + 0 \cdot \mathsf{D}^2 + 1 \cdot \mathsf{D}^3 + 1 \cdot \mathsf{D}^4 = 1 + \mathsf{D}^3 + \mathsf{D}^4.$$

- Given $\boldsymbol{u}(\mathsf{D}) = 1 + \mathsf{D}^2 + \mathsf{D}^4$, the information sequence $\boldsymbol{u}$ is obtained as

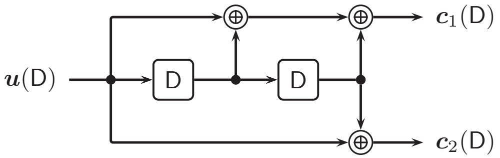
$$\boldsymbol{u}(\mathsf{D}) = \boldsymbol{u} = (1, 0, 1, 0, 1, \ldots).$$

- Convolution in time domain reverts to multiplication in the D-transform domain,

$$\boldsymbol{c}_j = \boldsymbol{u} * \boldsymbol{g}_j \qquad \longrightarrow \qquad \boldsymbol{c}_j(\mathsf{D}) = \boldsymbol{u}(\mathsf{D})\boldsymbol{g}_j(\mathsf{D})$$

  where the indeterminate D can be regarded as the delay operator.

Encoding using the D-transform



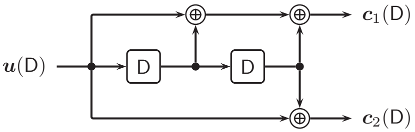
- Encoding operation in compact form as

$$\begin{aligned}\boldsymbol{c}(\mathsf{D}) &= (\boldsymbol{c}_1(\mathsf{D})\ \boldsymbol{c}_2(\mathsf{D})) = \boldsymbol{u}(\mathsf{D})(\boldsymbol{g}_1(\mathsf{D})\ \boldsymbol{g}_2(\mathsf{D})) \\ &= \boldsymbol{u}(\mathsf{D})\boldsymbol{G}(\mathsf{D}),\end{aligned}$$

  where $\boldsymbol{G}(\mathsf{D})$ is called the generator matrix of the code and the polynomials $\boldsymbol{g}_j(\mathsf{D})$ are called the generator polynomials.

- Typically, the codeword for an $R_\mathsf{c} = 1/2$ convolutional encoder is formed by multiplexing the bits corresponding to $\boldsymbol{c}_1(\mathsf{D})$ and $\boldsymbol{c}_2(\mathsf{D})$.

Encoding using the D-transform



Example: $R_c = 1/2$, $\nu = 2$ convolutional code

We obtain $\boldsymbol{G}(\mathsf{D}) = (\boldsymbol{g}_1(\mathsf{D})\ \boldsymbol{g}_2(\mathsf{D}))$, where

$$\boldsymbol{g}_1(\mathsf{D}) = 1 + \mathsf{D} + \mathsf{D}^2, \qquad \boldsymbol{g}_2(\mathsf{D}) = 1 + \mathsf{D}^2.$$

The code sequence $\boldsymbol{c}(\mathsf{D}) = (\boldsymbol{c}_1(\mathsf{D})\ \boldsymbol{c}_2(\mathsf{D}))$ is obtained by

$$\boldsymbol{c}_1(\mathsf{D}) = \boldsymbol{u}(\mathsf{D})\boldsymbol{g}_1(\mathsf{D}), \qquad \boldsymbol{c}_2(\mathsf{D}) = \boldsymbol{u}(\mathsf{D})\boldsymbol{g}_2(\mathsf{D}).$$

Encode $\boldsymbol{u} = (1, 0, 1, 0, 0, 0, \ldots)$. In the transform domain $\boldsymbol{u}(\mathsf{D}) = 1 + \mathsf{D}^2$, and

$$\begin{aligned}\boldsymbol{c}_1(\mathsf{D}) &= (1 + \mathsf{D}^2)(1 + \mathsf{D} + \mathsf{D}^2) = 1 + \mathsf{D} + \mathsf{D}^3 + \mathsf{D}^4\\ \boldsymbol{c}_2(\mathsf{D}) &= (1 + \mathsf{D}^2)(1 + \mathsf{D}^2) = 1 + \mathsf{D}^4.\end{aligned}$$

Multiplexing $\boldsymbol{c}_1(\mathsf{D})$ and $\boldsymbol{c}_2(\mathsf{D})$ we get $\boldsymbol{c} = (1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, \ldots)$.

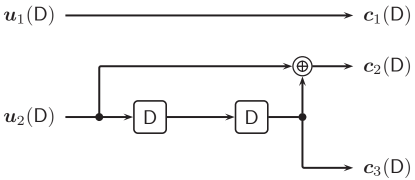
(n, k) convolutional code

- An (n, k) convolutional encoder is described by a $k \times n$ generator matrix $\boldsymbol{G}(\mathsf{D})$

$$\boldsymbol{G}(\mathsf{D}) = \begin{pmatrix} \boldsymbol{g}_{11}(\mathsf{D}) & \dots & \boldsymbol{g}_{1n}(\mathsf{D}) \\ \vdots & \ddots & \vdots \\ \boldsymbol{g}_{k1}(\mathsf{D}) & \dots & \boldsymbol{g}_{kn}(\mathsf{D}) \end{pmatrix}.$$

- $\boldsymbol{G}(\mathsf{D})$ completely defines the encoder, i.e., the mapping between information words and codewords.
- An encoder that has only polynomial entries in $\boldsymbol{G}(\mathsf{D})$ is said to be a feedforward encoder.
- An encoder that has rational functions in $\boldsymbol{G}(\mathsf{D})$ is said to be a recursive encoder.

(n, k) convolutional code



Example: $R_\mathsf{c} = 2/3$, $\nu = 2$ convolutional code

$$\boldsymbol{G}(\mathsf{D}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1+\mathsf{D}^2 & \mathsf{D}^2 \end{pmatrix}.$$

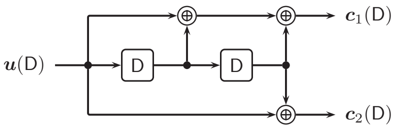
Systematic encoders

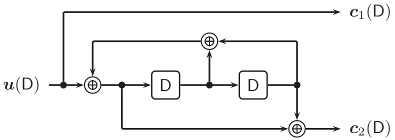
- For each generator matrix $\boldsymbol{G}(\mathsf{D})$ it is possible to obtain a systematic generator matrix $\boldsymbol{G}_{\mathrm{s}}(\mathsf{D})$ in the form

$$\boldsymbol{G}_{\mathrm{s}}(\mathsf{D}) = (\boldsymbol{I}_K \ \boldsymbol{P}(\mathsf{D}))\,,$$

  by linear combinations of the rows of $\boldsymbol{G}(\mathsf{D})$ combined with possible column permutations.
- Each feedforward convolutional encoder has an equivalent recursive, systematic encoder that generates the same code
- For $R_\mathsf{c} = 1/2$, an equivalent systematic encoder is obtained by dividing all the polynomials of $\boldsymbol{G}(\mathsf{D})$ by one of the polynomials.

Convolutional Codes





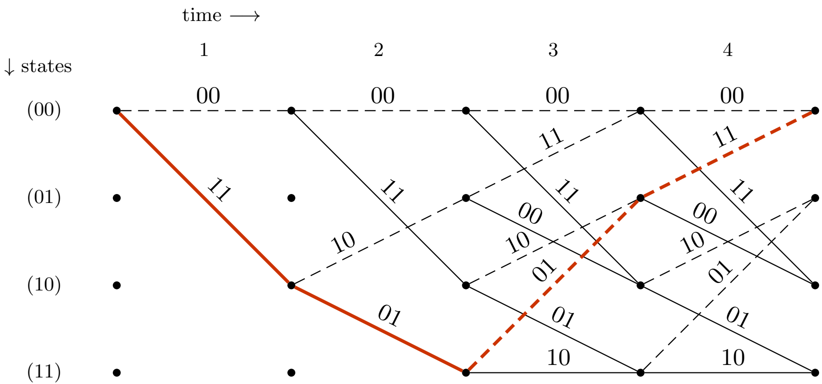
Example: $R_c = 1/2$ code with recursive, systematic encoder

Equivalent recursive systematic encoder of $\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2 \quad 1 + \mathsf{D}^2)$:

$$\boldsymbol{G}_s(\mathsf{D}) = \frac{\boldsymbol{G}(\mathsf{D})}{1 + \mathsf{D} + \mathsf{D}^2} = \begin{pmatrix} 1 & \dfrac{1 + \mathsf{D}^2}{1 + \mathsf{D} + \mathsf{D}^2} \end{pmatrix}.$$

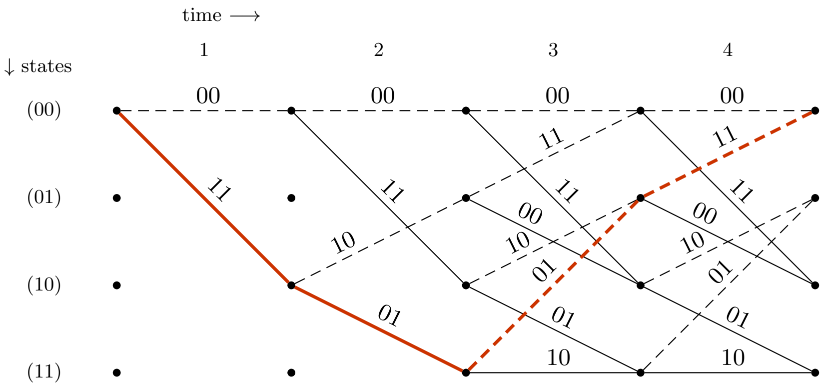
$\boldsymbol{G}(\mathsf{D})$ and $\boldsymbol{G}_s(\mathsf{D})$ generate the same code, i.e., the same list of codewords, but correspond to different encoders.

The trellis diagram



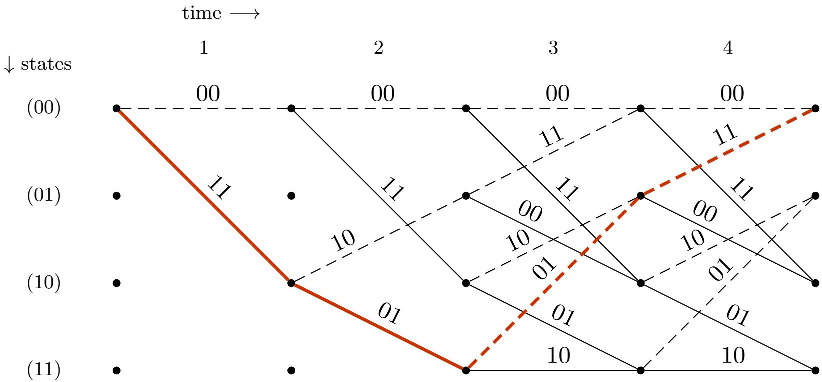
- Convolutional codes can be represented graphically by a trellis diagram.
- The nodes in the same vertical represent all encoder states.
- Horizontally, we represent time i, (trellis depth).
- It captures how the state of the encoder varies over time.

The trellis diagram



- Any code sequence corresponds to a path along the trellis.
- The edges are labeled with the code bits $c_{1,i}$ and $c_{2,i}$.
- Orange line: trellis path corresponding to $\boldsymbol{u} = (1, 1, 0, 0)$. The codeword is $\boldsymbol{c} = (1, 1, 0, 1, 0, 1, 1, 1)$.

The trellis diagram



- The trellis of a convolutional code is time-invariant, except at the beginning and at the end of the trellis.
- A convolutional encoder is completely described by a single section of the trellis.

Trellis termination

- Convolutional codes are stream oriented, but most practical applications require block-oriented transmission.
- Need to encode information words $\boldsymbol{u} = (u_1, \ldots, u_K)$ of length K bits into codewords of finite length $\boldsymbol{c} = (x_1, \ldots, x_N)$ $\longrightarrow$ Transform the (n, k) convolutional code into an (N, K) block code.
- Trivial trellis termination: Truncation
  - Run the encoder K/k steps and output the resulting codeword of length $N = \frac{K}{k} n$ bits.
  - The code rate is

$$R_{\mathsf{c}} = \frac{k}{n}.$$

  - Drawback: The last information bits are less reliable, since they are protected by fewer code bits.

Zero-termination

- Terminate the trellis to a known state (typically the all-zero state).
- It requires appending νk dummy bits to the information sequence to bring the state of the encoder back to the all-zero state.
- Results in a decrease of the code rate.
- Principle:
  - Run the encoder K/k steps, generating $N = \frac{K}{k} n$ coded bits.
  - Run the encoder ν additional steps by appending νk dummy bits to the information sequence, generating νn additional coded bits.

Zero-termination

We obtain:

- A block code of length $N = \frac{K}{k}n + \nu n$.
- The rate is $R_\mathsf{c} = \frac{K}{N} = \frac{K}{(K/k)n+\nu n} = \frac{k}{n}\frac{1}{1+k\nu/K} < \frac{k}{n}$.
- When the length K grows very large,

$$R_\mathsf{c} = \frac{K}{N} = \frac{k}{n}\frac{1}{1+k\nu/K} \stackrel{K\to\infty}{\longrightarrow} \frac{k}{n}.$$

- For $R_\mathsf{c} = 1/n$ feedforward encoders, termination is straightforward: We append ν zero bits to the information sequence.

Maximum-likelihood decoding of convolutional codes

- ML decoding rule:

$$\hat{\boldsymbol{c}} = \arg\max_{\boldsymbol{c}} p(\bar{\boldsymbol{y}}|\boldsymbol{c})$$

- We focus on the rate $R_{\mathsf{c}} = 1/2$ convolutional code with generator matrix $\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2 \quad 1 + \mathsf{D}^2)$.

Maximum-likelihood hard-decision decoding

The received sequence is

$$\bar{\boldsymbol{y}} = (\bar{\boldsymbol{y}}_1 \, \bar{\boldsymbol{y}}_2),$$

where

$$\begin{aligned} \bar{\boldsymbol{y}}_1 &= \boldsymbol{c}_1 + \boldsymbol{e}_1 \\ \bar{\boldsymbol{y}}_2 &= \boldsymbol{c}_2 + \boldsymbol{e}_2. \end{aligned}$$

with $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ being the error patterns introduced by the channel.

- ML decoding rule for hard-decision decoding:

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{H}}(\boldsymbol{c}, \bar{\boldsymbol{y}}).$$

Maximum-likelihood decoding of convolutional codes

- We assume block-oriented transmission, i.e., the convolutional code is terminated.
- If K is the information block length, we run the encoder $L = K/k + \nu$ times and

$$d_{\mathsf{H}}(\boldsymbol{c}, \bar{\boldsymbol{y}}) = \sum_{i=1}^{L} \left(d_{\mathsf{H}}\left(c_{1i}, \bar{y}_{1i}\right) + d_{\mathsf{H}}\left(c_{2i}, \bar{y}_{2i}\right)\right) = \sum_{i=1}^{L} d_{\mathsf{H}}\left(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i\right)$$

  where $\boldsymbol{c}^i = (c_{1i}, c_{2i})$, $\bar{\boldsymbol{y}}^i = (\bar{y}_{1i}, \bar{y}_{2i})$.
- Define

$$\lambda_i^{\mathsf{HARD}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i) \triangleq d_{\mathsf{H}}\left(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i\right).$$

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c} \in \mathcal{C}} \sum_{i=1}^{L} \lambda_i^{\mathsf{HARD}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i).$$

Maximum-likelihood soft-decision decoding (AWGN channel)

The received sequence is

$$\boldsymbol{y} = (\boldsymbol{y}_1\, \boldsymbol{y}_2),$$

where

$$\begin{aligned} \boldsymbol{y}_1 &= (-1)^{\boldsymbol{c}_1} + \mathbf{n}_1 = \boldsymbol{x}_1 + \boldsymbol{n}_1 \\ \boldsymbol{y}_2 &= (-1)^{\boldsymbol{c}_2} + \mathbf{n}_2 = \boldsymbol{x}_2 + \boldsymbol{n}_2, \end{aligned}$$

where $(-1)^{\boldsymbol{c}_1} = ((-1)^{c_{1,1}}, (-1)^{c_{1,2}}, \ldots)$ and $(-1)^{\boldsymbol{c}_2} = ((-1)^{c_{2,1}}, (-1)^{c_{2,2}}, \ldots)$ are the BPSK-modulated sequences, and the Gaussian noise is of zero mean and variance $\sigma^2 = \mathsf{N}_0/2$.

- ML decoding rule for soft-decision decoding:

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{E}}^2(\boldsymbol{x}, \boldsymbol{y}).$$

Maximum-likelihood soft-decision decoding (AWGN channel)

- Assume block-oriented transmission where we run the encoder L steps:

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d_{\mathsf{E}}^2(\boldsymbol{x},\boldsymbol{y}) = \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} ||\boldsymbol{y}^i - (-1)^{\boldsymbol{c}^i}||^2$$

$$= \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \left(|y_{1,i} - (-1)^{c_{1,i}}|^2 + |y_{2,i} - (-1)^{c_{2,i}}|^2\right)$$

- Define

$$\lambda_i^{\mathsf{SOFT}}(\boldsymbol{c}^i,\boldsymbol{y}^i) \triangleq \left(|y_{1,i} - (-1)^{c_{1,i}}|^2 + |y_{2,i} - (-1)^{c_{2,i}}|^2\right).$$

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \lambda_i^{\mathsf{SOFT}}(\boldsymbol{c}^i,\boldsymbol{y}^i).$$

Maximum-likelihood decoding of convolutional codes

- ML decoding consists of solving the problem

$$\hat{\boldsymbol{c}} = \arg\min_{\boldsymbol{c}\in\mathcal{C}} \sum_{i=1}^{L} \lambda_i. \tag{1}$$

  where $\lambda_i = \lambda_i^{\text{HARD}}(\boldsymbol{c}^i, \bar{\boldsymbol{y}}^i)$ for hard-decision decoding and $\lambda_i = \lambda_i^{\text{SOFT}}(\boldsymbol{c}^i, \boldsymbol{y}^i)$ for soft-decision decoding.
- The λ_i's are called the branch metrics for trellis section i.
- Every codeword $\boldsymbol{c}$ corresponds to a path in the trellis $\longrightarrow$ Solve (**??**) using the trellis diagram.
- The number of computations to solve (**??**) by brute force is enormous (2^K codewords), but...
- There exists an algorithm to solve (**??**) in linear time with K with no loss of optimality: The Viterbi algorithm.

The Viterbi Algorithm

- $\Gamma_\ell(\boldsymbol{c})$: The accumulated distance between a codeword $\boldsymbol{c}$ and the received vector $\boldsymbol{y}$ (or $\bar{\boldsymbol{y}}$) up to time ℓ, i.e.,

$$\Gamma_\ell(\boldsymbol{c}) = \sum_{i=1}^{\ell} \lambda_i.$$

- Consider two code sequences $\boldsymbol{c}$ and $\tilde{\boldsymbol{c}}$ that diverge from the all-zero state at time $i = 1$, remerge to the same state $\boldsymbol{s}_\ell$ at time $\ell > i$, and they are equal for all $t > \ell$:
  - If the path corresponding to $\boldsymbol{c}$ is such that $\Gamma_\ell(\boldsymbol{c}) > \Gamma_\ell(\tilde{\boldsymbol{c}})$ at the time they merge $\Longrightarrow \Gamma_L(\boldsymbol{c}) > \Gamma_L(\tilde{\boldsymbol{c}})$.
  - Therefore, we can safely discard $\boldsymbol{c}$. The path that is maintained is called the survivor.

The Viterbi Algorithm

Definitions

1. $\lambda_i(\boldsymbol{s}', \boldsymbol{s})$: The branch metric from state $\boldsymbol{s}'$ at time i to state $\boldsymbol{s}$ at time $i+1$, i.e., $\lambda_i(\boldsymbol{s}', \boldsymbol{s}) = \lambda_i(\boldsymbol{c}^i, \boldsymbol{y}^i)$, where $\boldsymbol{c}^i$ are the code bits of the branch $\boldsymbol{s}' \to \boldsymbol{s}$, and $\boldsymbol{y}^i$ (or $\bar{\boldsymbol{y}}^i$ for hard decoding) are the received symbols for trellis section i.
2. $\Gamma_i(\boldsymbol{s})$: The cumulative metric for the survivor state $\boldsymbol{s}$ at time i, i.e., it is the sum of the metrics for the surviving path.
3. $\Gamma_{i+1}(\boldsymbol{s}', \boldsymbol{s})$: The tentative cumulative metric for the path from $\boldsymbol{s}'$ at time i to $\boldsymbol{s}$ at time $i+1$, i.e., $\Gamma_{i+1}(\boldsymbol{s}', \boldsymbol{s}) = \Gamma_i(\boldsymbol{s}') + \lambda_i(\boldsymbol{s}', \boldsymbol{s})$.

The Viterbi Algorithm

1: Initialization. Set $\Gamma_1(00) = 0$ and $\Gamma_1(\boldsymbol{s}) = \infty$ for all $\boldsymbol{s} \in \{1, \ldots, 2^\nu - 1\}$. (The encoder, and hence the trellis, is initialized to the all-zero state).

2: **for** $i = 2$ to L **do**

3: Compute the possible branch metrics $\lambda_{i-1}(\boldsymbol{s}', \boldsymbol{s})$.

4: For each state $\boldsymbol{s}'$ at time $i-1$ and all possible states $\boldsymbol{s}$ at time i that can be reached from $\boldsymbol{s}'$, compute the metrics $\Gamma_i(\boldsymbol{s}', \boldsymbol{s}) = \Gamma_{i-1}(\boldsymbol{s}') + \lambda_{i-1}(\boldsymbol{s}', \boldsymbol{s})$ for the paths extending from $\boldsymbol{s}'$ to $\boldsymbol{s}$.

5: For each state $\boldsymbol{s}$ at time i, select and store the path possessing the minimum among the metrics $\Gamma_i(\boldsymbol{s}', \boldsymbol{s})$. The cumulative metric for state $\boldsymbol{s}$ will be $\Gamma_i(\boldsymbol{s}) = \min_{\boldsymbol{s}'} \Gamma_i(\boldsymbol{s}', \boldsymbol{s})$.

6: **end for**

7: Decision. Since we assume the termination of the trellis to the all-zero state, after the final ACS iteration, the ML trellis path (i.e., the ML codeword) will be the survivor at the all-zero state.

Viterbi decoding: Example

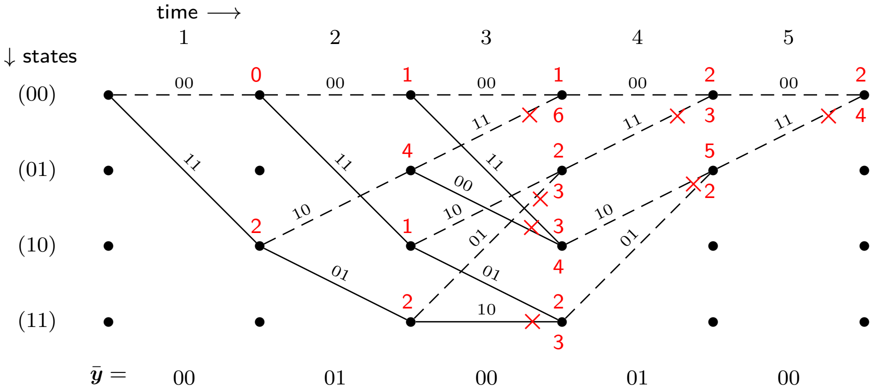
Viterbi decoding of the $R_\mathsf{c} = 1/2$ convolutional code

- Convolutional encoder with generator matrix

$$\boldsymbol{G}(\mathsf{D}) = (1 + \mathsf{D} + \mathsf{D}^2 \;\; 1 + \mathsf{D}^2)$$

  and hard-decision decoding.
- The information block length is $K = 3$, zero-termination $\longrightarrow$ We run the encoder $L = K/k + \nu = 3 + 2 = 5$ steps.
- The encoder is initialized to the all-zero state.
- Suppose we receive $\bar{\boldsymbol{y}} = (0, 0, 0, 1, 0, 0, 0, 1, 0, 0)$.
- Perform ML decoding using the Viterbi algorithm.

Viterbi algorithm: Example



Bounds on the probability of error

- Deriving the exact probability of error for long codes is unfeasible.
- We can derive bounds on the error probability using the union bound.
- Let $\boldsymbol{c}$ be the transmitted codeword and $\hat{\boldsymbol{c}}$ the decoded codeword.
- The word error probability is given by

$$P_{\mathsf{w}} = \sum_{\boldsymbol{c}\in\mathcal{C}} \mathsf{Pr}(\hat{\boldsymbol{c}} \neq \boldsymbol{c}|\boldsymbol{c})\mathsf{Pr}(\boldsymbol{c}) = \frac{1}{2^K}\sum_{\boldsymbol{c}\in\mathcal{C}} \mathsf{Pr}(\hat{\boldsymbol{c}} \neq \boldsymbol{c}|\boldsymbol{c}).$$

- If the code is linear and the channel symmetric, the probability of error does not depend on the transmitted codeword $\longrightarrow$We can assume that the all-zero codeword $\boldsymbol{c} = \mathbf{0}$ was transmitted and

$$P_{\mathsf{w}} = \mathsf{Pr}(\hat{\boldsymbol{c}} \neq \mathbf{0}|\mathbf{0}).$$

Bounds on the probability of error

Now,

$$P_{\mathsf{w}} = \Pr(\hat{\boldsymbol{c}} \neq \mathbf{0}|\mathbf{0}) = \Pr\left(\bigcup_{\boldsymbol{c}\neq\mathbf{0}} \hat{\boldsymbol{c}} = \boldsymbol{c}|\mathbf{0}\right) \leq \sum_{\boldsymbol{c}\neq\mathbf{0}} \Pr(\hat{\boldsymbol{c}} = \boldsymbol{c}|\mathbf{0}) = \sum_{\boldsymbol{c}\neq\mathbf{0}} \Pr\left(\mathbf{0} \to \boldsymbol{c}\right),$$

where $\Pr\left(\mathbf{0} \to \boldsymbol{c}\right) \triangleq \Pr(\hat{\boldsymbol{c}} = \boldsymbol{c}|\mathbf{0})$ is the pairwise error probability of decoding into codeword $\boldsymbol{c}$ if codeword $\mathbf{0}$ was transmitted.

$\Pr\left(\mathbf{0} \to \boldsymbol{c}\right)$ depends only on the Hamming distance between $\mathbf{0}$ and $\boldsymbol{c}$ (Euclidean distance between $\mathbf{0}$ and $\boldsymbol{x}$), i.e., on the weight of $\boldsymbol{c}$!

- Thus, denoting by $\boldsymbol{c}_d$ a codeword of Hamming weight d,

$$P_{\mathsf{w}} \leq \sum_{\boldsymbol{c}\neq\mathbf{0}} \Pr(\mathbf{0} \to \boldsymbol{c}) = \sum_{d=d_{\mathsf{min}}}^{N} A_d \Pr(\mathbf{0} \to \boldsymbol{c}_d),$$

  where A_d is the number of codewords of Hamming weight d.
- $\{A_d\}$, $d = 1, \ldots, N$, is referred to as the weight enumerator of the code.

Bounds on the probability of error (soft-decision decoding)

- Let $\boldsymbol{x}(\boldsymbol{c})$ is the BPSK-modulated sequence corresponding to codeword $\boldsymbol{c}$. Then,

$$\mathsf{Pr}\left(\mathbf{0} \rightarrow \boldsymbol{c}_d\right) = \mathsf{Pr}(d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{y}) < d_{\mathsf{E}}(\boldsymbol{x}(\mathbf{0}), \boldsymbol{y})).$$

- Let $\mathsf{E_s}$ the average energy per transmitted symbol and $x(c_i) = (-1)^{c_i}\sqrt{\mathsf{E_s}}$.
- Example: $\boldsymbol{x}(\mathbf{0}) = (+\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, \ldots)$.
- Example: $\boldsymbol{x}((1,1,1,0,0,\ldots)) = (-\sqrt{\mathsf{E_s}}, -\sqrt{\mathsf{E_s}}, -\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, +\sqrt{\mathsf{E_s}}, \ldots)$.

Bounds on the probability of error (soft-decision decoding)

- The Euclidean distance between $\boldsymbol{x}(\boldsymbol{c}_d)$ and $\boldsymbol{x}(\mathbf{0})$ is

$$d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0})) = 2\sqrt{d\mathsf{E}_{\mathsf{s}}}.$$

- Since the Gaussian noise has variance $\sigma^2 = \mathsf{N}_0/2$ in all directions, $\Pr(\mathbf{0} \to \boldsymbol{c}_d)$ is the probability that the noise in the direction of $\boldsymbol{c}_d$ has magnitude greater than

$$\frac{d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0}))}{2} = \sqrt{d\mathsf{E}_{\mathsf{s}}},$$

- Thus,

$$\begin{aligned}
\Pr(\mathbf{0} \to \boldsymbol{c}_d) &= \Pr(d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{y}) < d_{\mathsf{E}}(\boldsymbol{x}(\mathbf{0}), \boldsymbol{y})) \\
&= \Pr\left(\tilde{Y} > \frac{d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0}))}{2}\right)\Bigg|_{\tilde{Y}\sim\mathcal{N}(0,\sigma^2)} \\
&= \mathsf{Q}\left(\frac{d_{\mathsf{E}}(\boldsymbol{x}(\boldsymbol{c}_d), \boldsymbol{x}(\mathbf{0}))}{2\sigma}\right) = \mathsf{Q}\left(\sqrt{\frac{2d\mathsf{E}_{\mathsf{s}}}{\mathsf{N}_0}}\right) = \mathsf{Q}\left(\sqrt{\frac{2dR_{\mathsf{c}}\mathsf{E}_{\mathsf{b}}}{\mathsf{N}_0}}\right).
\end{aligned}$$

Bounds on the probability of error (soft-decision decoding)

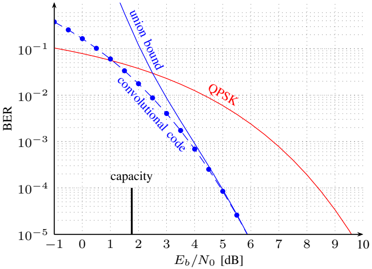
- Finally,

$$P_{\mathsf{w}}^{\mathsf{SOFT}} \leq \sum_{d=d_{\mathsf{min}}}^{N} A_d \mathsf{Pr}\left(\mathbf{0} \to \boldsymbol{c}_d\right)$$
$$= \sum_{d=d_{\mathsf{min}}}^{N} A_d \mathsf{Q}\left(\sqrt{\frac{2dR_{\mathsf{c}}\mathsf{E}_{\mathsf{b}}}{\mathsf{N}_0}}\right).$$

- The bound depends only on the weight enumerator of the code $\{A_d\}$.
- For high $\mathsf{E}_{\mathsf{b}}/\mathsf{N}_0$, the performance is dominated by the term with minimum Hamming distance and

$$P_{\mathsf{w}}^{\mathsf{SOFT}} \approx A_{d_{\mathsf{min}}} \mathsf{Q}\left(\sqrt{\frac{2d_{\mathsf{min}}R_{\mathsf{c}}\mathsf{E}_{\mathsf{b}}}{\mathsf{N}_0}}\right).$$

Bounds on the probability of error (soft-decision decoding)



- The bit error probability P_b can be computed in a similar way.
- Let $A_{w,d}$ the number of codewords of weight d produced by an information word of weight w.
- Then,

$$P_\mathsf{b}^{\mathsf{SOFT}} \leq \frac{1}{K} \sum_{d=d_{\mathsf{min}}}^{N} \sum_{w=1}^{K} w A_{w,d} Q\left(\sqrt{\frac{2dR_\mathsf{c}\mathsf{E_b}}{\mathsf{N_0}}}\right).$$