Channel coding

Convolutional codes

Manuel A. Vázquez
Jose Miguel Leiva
Joaquín Míguez

March 4, 2024

Index

Index

Linear block codes v. convolutional codes

A few (related) differences...

Convolutional codes	Linear block codes

Linear block codes v. convolutional codes

A few (related) differences...

Convolutional codes	Linear block codes
• Encoding is *continuous*	• Encoding is *blockwise*

Linear block codes v. convolutional codes

A few (related) differences...

Convolutional codes	Linear block codes
• Encoding is *continuous* • System has **memory**	• Encoding is *blockwise* • System has **no memory**

Linear block codes v. convolutional codes

A few (related) differences...

Convolutional codes	Linear block codes
• Encoding is *continuous*	• Encoding is *blockwise*
• System has **memory**	• System has **no memory**
• Sequence-to-sequence mapping	• Message-to-codeword mapping

Linear block codes v. convolutional codes

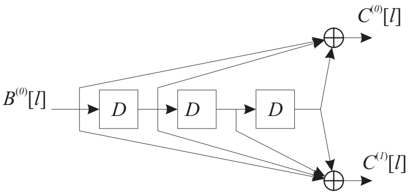
A few (related) differences...

Convolutional codes	Linear block codes
• Encoding is *continuous*	• Encoding is *blockwise*
• System has **memory**	• System has **no memory**
• Sequence-to-sequence mapping	• Message-to-codeword mapping

In both schemes, every operation (e.g., convolution) is in $GF(2)$.

Specification

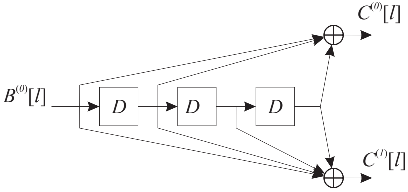
Codes are often specified through a block diagram that illustrates how the input is transformed into the output, e.g.



- $B^{(j)}[l]$ is the l-th bit of the j-th input
- $C^{(j)}[l]$ is the l-th bit of the j-th output
- D is a delay element

Specification

Codes are often specified through a block diagram that illustrates how the input is transformed into the output, e.g.

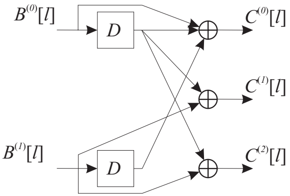


- $B^{(j)}[l]$ is the l-th bit of the j-th input
- $C^{(j)}[l]$ is the l-th bit of the j-th output
- D is a delay element

The system has memory

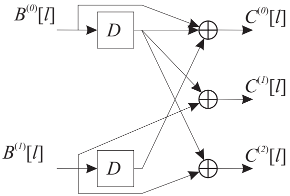
The output bits depend on previous (and current) inputs: this is a **state machine**

Specification: equations



- Connection between the inputs and the outputs

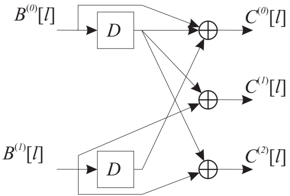
Specification: equations



- Connection between the inputs and the outputs

$$C^{(0)}[l] \quad = B^{(0)}[l] + B^{(0)}[l-1] + B^{(1)}[l-1]$$

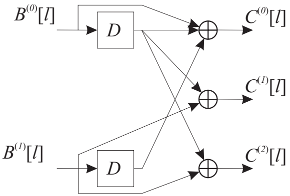
Specification: equations



- Connection between the inputs and the outputs

$$
\begin{aligned}
C^{(0)}[l] &= B^{(0)}[l] + B^{(0)}[l-1] + B^{(1)}[l-1] \\
C^{(1)}[l] &= B^{(0)}[l-1] + B^{(1)}[l]
\end{aligned}
$$

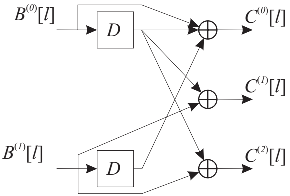
Specification: equations



- Connection between the inputs and the outputs

$$
\begin{aligned}
C^{(0)}[l] &= B^{(0)}[l] + B^{(0)}[l-1] + B^{(1)}[l-1] \\
C^{(1)}[l] &= B^{(0)}[l-1] + B^{(1)}[l] \\
C^{(2)}[l] &= B^{(0)}[l-1] + B^{(1)}[l]
\end{aligned}
$$

Specification: equations



- Connection between the inputs and the outputs

$$
\begin{aligned}
C^{(0)}[l] &= B^{(0)}[l] + B^{(0)}[l-1] + B^{(1)}[l-1] \\
C^{(1)}[l] &= B^{(0)}[l-1] + B^{(1)}[l] \\
C^{(2)}[l] &= B^{(0)}[l-1] + B^{(1)}[l]
\end{aligned}
$$

- we express this relations in the D domain...

Convolution

...but, where is the convolution in a *convolutional* code?

Convolution

...but, where is the convolution in a *convolutional* code?

Convolution of discrete-time signals

$$x[n] * h[n] = h[n] * x[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

and assuming the impulse response, $h[k]$, is non-zero between time instants, e.g., 0 and 1

$$= h[0]x[n] + h[1]x[n-1]$$

Convolution

...but, where is the convolution in a *convolutional* code?

Convolution of discrete-time signals

$$x[n] * h[n] = h[n] * x[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

and assuming the impulse response, $h[k]$, is non-zero between time instants, e.g., 0 and 1

$$= h[0]x[n] + h[1]x[n-1]$$

For the first output, e.g., we have

$$C^{(0)}[l] = \underbrace{B^{(0)}[l] + B^{(0)}[l-1]}_{B^{(0)}[l]*\begin{bmatrix}1 & 1\end{bmatrix}} + \underbrace{B^{(1)}[l-1]}_{B^{(1)}[l]*\begin{bmatrix}0 & 1\end{bmatrix}}$$

Convolution

...but, where is the convolution in a *convolutional* code?

Convolution of discrete-time signals

$$x[n] * h[n] = h[n] * x[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

and assuming the impulse response, $h[k]$, is non-zero between time instants, e.g., 0 and 1

$$= h[0]x[n] + h[1]x[n-1]$$

For the first output, e.g., we have

$$C^{(0)}[l] = \underbrace{B^{(0)}[l] + B^{(0)}[l-1]}_{B^{(0)}[l] * \begin{bmatrix}1 & 1\end{bmatrix}} + \underbrace{B^{(1)}[l-1]}_{B^{(1)}[l] * \begin{bmatrix}0 & 1\end{bmatrix}}$$

Every output is a sum of convolutions!!

D transform

Definition: The D transform...

...of a binary *sequence* $B^{(i)}[l]$ is

$$B^{(i)}(D) = \sum_u B^{(i)}[u] \cdot D^u$$
$$= \cdots B^{(i)}[-1] \cdot D^{-1} + B^{(i)}[0] + B^{(i)}[1] \cdot D^{1} + \cdots$$

...and we write $B^{(i)}[l] \stackrel{D}{\longleftrightarrow} B^{(i)}(D)$

D transform

Definition: The D transform...

...of a binary *sequence* $B^{(i)}[l]$ is

$$B^{(i)}(D) = \sum_u B^{(i)}[u] \cdot D^u$$
$$= \cdots B^{(i)}[-1] \cdot D^{-1} + B^{(i)}[0] + B^{(i)}[1] \cdot D^1 + \cdots$$

...and we write $B^{(i)}[l] \xleftrightarrow{D} B^{(i)}(D)$

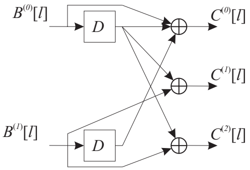
with the property

$$\boxed{B^{(i)}[l-d] \leftrightarrow B^{(i)}(D) \cdot D^d}$$

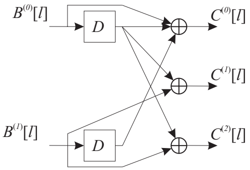
so that, e.g.,

$$B^{(i)}[l] + B^{(i)}[l-1] + B^{(i)}[l-2] + B^{(i)}[l-3] \xleftrightarrow{D} B^{(i)}(D)(1 + D + D^2 + D^3).$$

Generator matrix



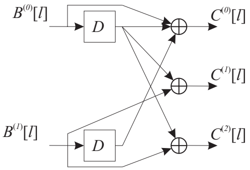
Generator matrix



- In polynomial form

$$C^{(0)}(D) = (1 + D)B^{(0)}(D) + DB^{(1)}(D)$$
$$C^{(1)}(D) = DB^{(0)}(D) + B^{(1)}(D)$$
$$C^{(2)}(D) = DB^{(0)}(D) + B^{(1)}(D)$$

Generator matrix



- In polynomial form

$$C^{(0)}(D) = (1 + D)B^{(0)}(D) + DB^{(1)}(D)$$
$$C^{(1)}(D) = DB^{(0)}(D) + B^{(1)}(D)$$
$$C^{(2)}(D) = DB^{(0)}(D) + B^{(1)}(D)$$

- Using matrices: $\mathbf{C}(D) = \mathbf{B}(D)\mathbf{G}(D)$, with

$$\mathbf{C}(D) = \begin{bmatrix} C^{(0)}(D) & C^{(1)}(D) & C^{(2)}(D) \end{bmatrix} \quad \mathbf{B}(D) = \begin{bmatrix} B^{(0)}(D) & B^{(1)}(D) \end{bmatrix}$$

$$\mathbf{G}(D) = \begin{bmatrix} 1 + D & D & D \\ D & 1 & 1 \end{bmatrix}_{k \times n}$$

$\mathbf{G} \equiv$ generator matrix
$g_{ij} \equiv$ contribution of the i-th input to the j-th output

Definitions

Definition: Overall memory...

...of the code, M_t, is the numer of delay units in the coding scheme.

$$M_t = \sum_{i=0}^{k-1} M^{(i)}$$

with

$$M^{(i)} = \max_j \text{degree}\left(g_{ij}(D)\right) \equiv \text{memory } i\text{-th input}$$

Definitions

Definition: Overall memory...

...of the code, M_t, is the numer of delay units in the coding scheme.

$$M_t = \sum_{i=0}^{k-1} M^{(i)}$$

with

$$M^{(i)} = \max_j \text{degree}\left(g_{ij}(D)\right) \equiv \text{memory } i\text{-th input}$$

Definition: Constraint length

...of the code, K, is the maximum length of the impulse response,

$$K = 1 + \max_{i,j} \text{degree}\ \left(g_{ij}(D)\right)$$

Definitions

Definition: Overall memory...

...of the code, M_t, is the numer of delay units in the coding scheme.

$$M_t = \sum_{i=0}^{k-1} M^{(i)}$$

with

$$M^{(i)} = \max_j \text{degree}\left(g_{ij}(D)\right) \equiv \text{memory } i\text{-th input}$$

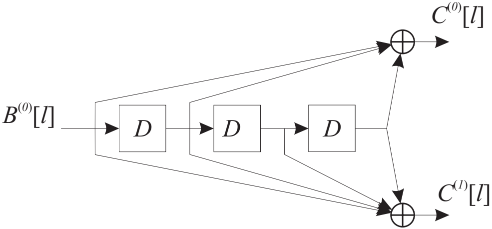
Definition: Constraint length

...of the code, K, is the maximum length of the impulse response,

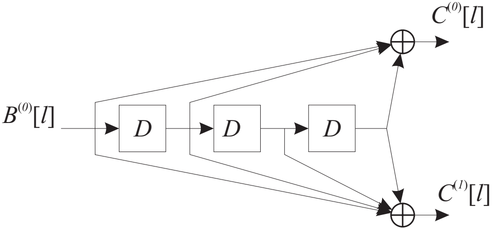
$$K = 1 + \max_{i,j} \text{degree } \left(g_{ij}(D)\right)$$

A convolutional code can also be **systematic** (same definition).

The encoder as a finite-state machine



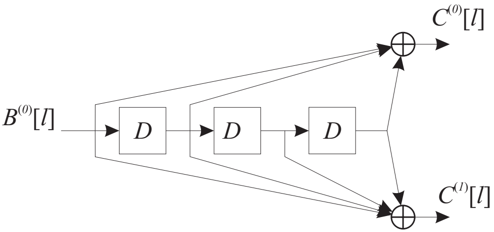
The encoder as a finite-state machine



- The *state* of the encoder is given by the bits stored (yielded as output) in the delay elements, here

$$\Psi \equiv (B^{(0)}[l-1], B^{(0)}[l-2], B^{(0)}[l-3]).$$

The encoder as a finite-state machine



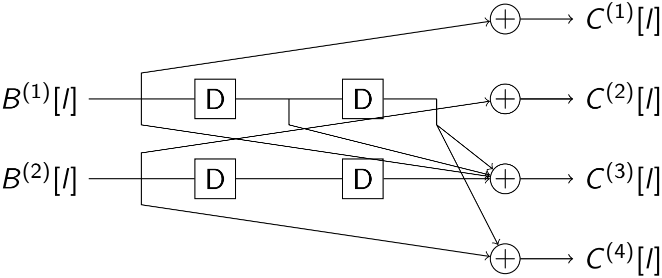
- The *state* of the encoder is given by the bits stored (yielded as output) in the delay elements, here

$$\Psi \equiv (B^{(0)}[l-1], B^{(0)}[l-2], B^{(0)}[l-3]).$$

- **In general there are 2^{M_t} possible states** (bits from delay elements across all the inputs are stacked together). A possible mapping here:

$$\Psi_0 \to (0,0,0) \quad \Psi_1 \to (1,0,0) \quad \Psi_2 \to (0,1,0) \quad \Psi_3 \to (1,1,0)$$
$$\Psi_4 \to (0,0,1) \quad \Psi_5 \to (1,0,1) \quad \Psi_6 \to (0,1,1) \quad \Psi_7 \to (1,1,1)$$

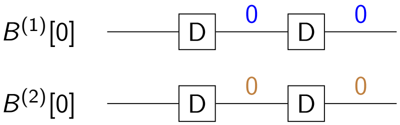
State transition I



Let us take a look at the evolution of the state of the system for a simple example...

l	−2	−1	0	+1
$B^{(1)}[l]$	0	0	1	1
$B^{(2)}[l]$	0	0	0	1
	previous bits		**bits to be encoded**	

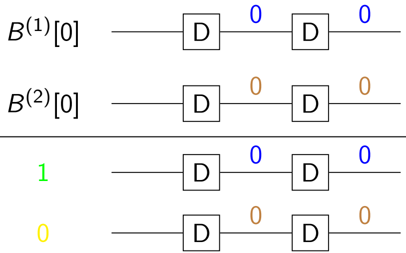
State transition II



Initial state:

$\Psi = [0, 0, 0, 0,]$

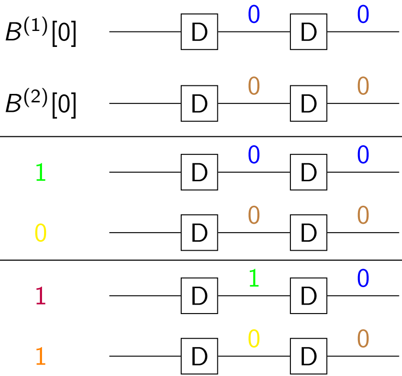
State transition II



Initial state:
$\Psi = [0, 0, 0, 0,]$

$\Psi = [0, 0, 0, 0,]$
$B^{(1)}[0] = 1$
$B^{(2)}[0] = 0$

State transition II



Initial state:

$\Psi = [0, 0, 0, 0,]$

$\Psi = [0, 0, 0, 0,]$

$B^{(1)}[0] = 1$

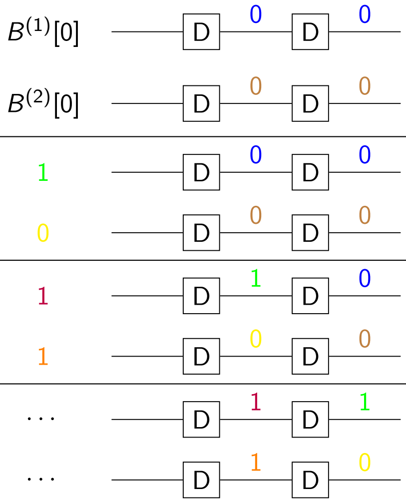
$B^{(2)}[0] = 0$

$\Psi = [1, 0, 0, 0,]$

$B^{(1)}[1] = 1$

$B^{(2)}[1] = 1$

State transition II



Initial state:

$\Psi = [0, 0, 0, 0,]$

$\Psi = [0, 0, 0, 0,]$

$B^{(1)}[0] = 1$

$B^{(2)}[0] = 0$

$\Psi = [1, 0, 0, 0,]$

$B^{(1)}[1] = 1$

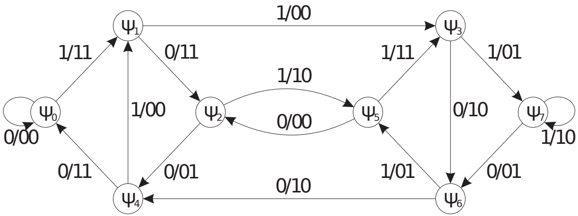
$B^{(2)}[1] = 1$

$\Psi = [1, 1, 1, 0,]$

$B^{(1)}[1] = \cdots$

$B^{(2)}[1] = \cdots$

State diagram



Every arrow is labeled with

- the input bits triggering that transition, and
- the output bits originating from that state given the corresponding input bits.

Index

Encoding

Straightforward once we know

- the initial state
- the state diagram

Example

Let us assume we start from state Ψ_0 and we want to encode the sequence [001]. The result is

$$[00\ 00\ 11]$$

Encoding

Straightforward once we know

- the initial state
- the state diagram

Example

Let us assume we start from state Ψ_0 and we want to encode the sequence [001]. The result is

$$[00\ 00\ 11]$$

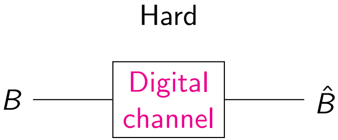
Header

After the *information sequence*, a *header* is transmitted to force the encoder to go back to its initial state.

information	header

Index

Decoding



- Metric: **Hamming** distance

Decoding

Hard

- Metric: **Hamming** distance

Soft

- Metric: **Euclidean** distance (at every step, squared difference)

Decoding

Hard

- Metric: **Hamming** distance

Soft

- Metric: **Euclidean** distance (at every step, squared difference)

In both cases:

- the goal is to find the *full* sequence most likely transmitted
- the solution is given by the Viterbi algorithm

Viterbi algorithm

Some keys

Viterbi algorithm

Some keys

- We need to assess all the trajectories starting a the initial state (usually Ψ_0).

Viterbi algorithm

Some keys

- We need to assess all the trajectories starting a the initial state (usually Ψ_0).
- For every possible transition, we compare its corresponding output with the observed one.

Viterbi algorithm

Some keys

- We need to assess all the trajectories starting a the initial state (usually Ψ_0).
- For every possible transition, we compare its corresponding output with the observed one.
- At every time instant, for every possible state, we need to find the path reaching it with the smallest accumulated cost.

Viterbi algorithm

Some keys

- We need to assess all the trajectories starting a the initial state (usually Ψ_0).
- For every possible transition, we compare its corresponding output with the observed one.
- At every time instant, for every possible state, we need to find the path reaching it with the smallest accumulated cost.
- Whenever two paths reach the same state, we keep the one with the smallest *accumulated* cost.

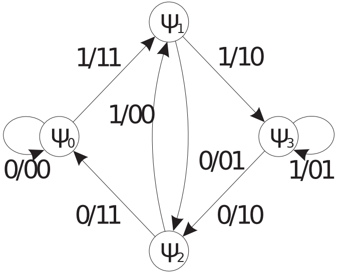
Viterbi algorithm

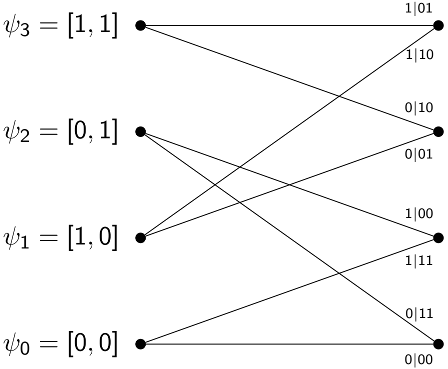
Some keys

- We need to assess all the trajectories starting a the initial state (usually Ψ_0).
- For every possible transition, we compare its corresponding output with the observed one.
- At every time instant, for every possible state, we need to find the path reaching it with the smallest accumulated cost.
- Whenever two paths reach the same state, we keep the one with the smallest *accumulated* cost.
- Decoding must end up in the initial state since a *header* is appended to every transmitted sequence in order to enforce this.

Error-free example

- Input sequence: 1 1 0 1 0 1 0 0
- Received sequence: 11 10 10 00 01 00 01 11

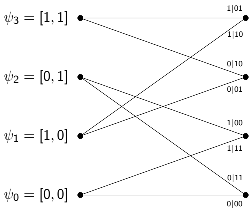


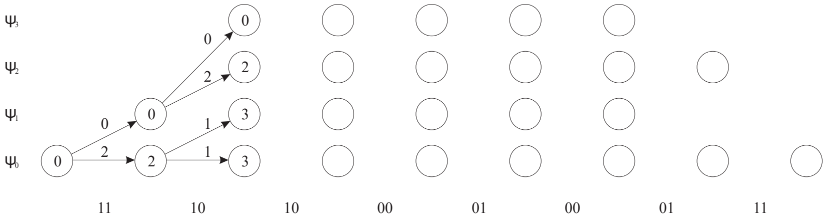


(**Trellis** representation)

Error-free example

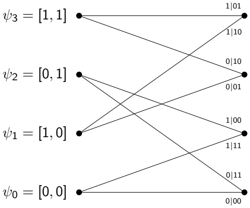
- Two first iterations

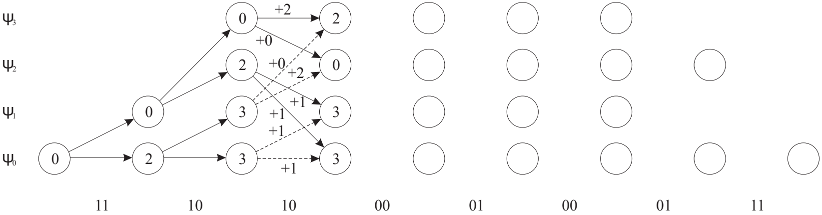




Error-free example

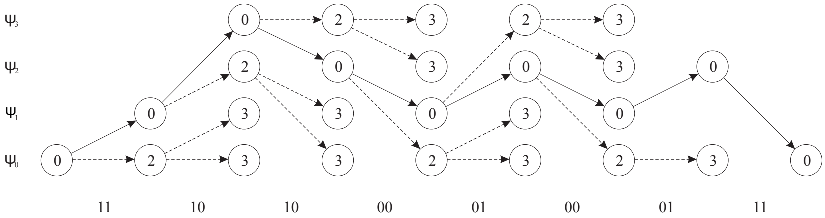
- Third iteration





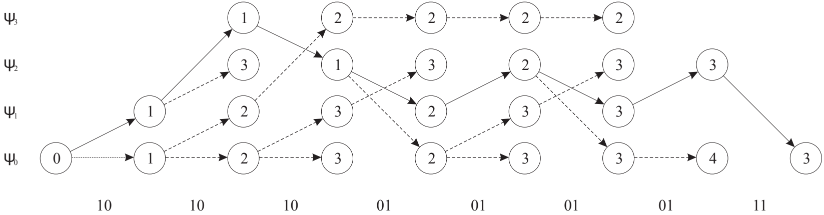
Error-free example

- Final result



Example with errors

- Received sequence: 10 10 10 01 01 01 01 11
- Final result



- The states sequence $\Psi_0\Psi_1\Psi_3\Psi_2\Psi_1\Psi_2\Psi_1\Psi_2\Psi_0$ is associated with the input sequence 11010100

Performance

- Soft decoding

$$P_e \approx \kappa_2 \mathrm{Q}\left(\sqrt{\frac{2D_{min}E_s}{N_0}}\right)$$

where κ_2 is the number of bit errors (in the decoded sequence) caused by the sequence associated with D_{min}.

Performance

- Soft decoding

$$P_e \approx \kappa_2 \mathrm{Q}\left(\sqrt{\frac{2D_{min}E_s}{N_0}}\right)$$

where κ_2 is the number of bit errors (in the decoded sequence) caused by the sequence associated with D_{min}.

- Hard decoding

$$P_e \approx \kappa_2 \sum_{i=\lfloor (D_{min}-1)/2 \rfloor + 1}^{nz} \binom{nz}{i} \epsilon^i (1-\epsilon)^{nz-i}$$

where z is the length of the trajectory associated with D_{min} and ϵ is the bit error probability.

Finding the minimum distance D_{min}

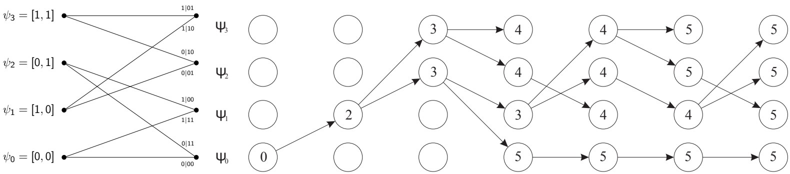
A convolutional code is linear, and hence **the encoded sequence that is closest to the all-zeros sequence determines** D_{min}.

Finding the minimum distance D_{min}

A convolutional code is linear, and hence **the encoded sequence that is closest to the all-zeros sequence determines** D_{min}.

Goal

We seek the sequence of states (path) that starts at the all-zeros sequence and goes back to it with the smallest accumulated cost.

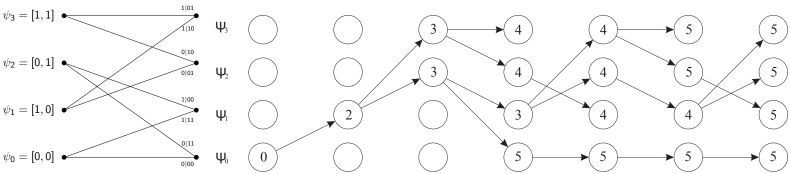


Finding the minimum distance D_{min}

A convolutional code is linear, and hence **the encoded sequence that is closest to the all-zeros sequence determines** D_{min}.

Goal

We seek the sequence of states (path) that starts at the all-zeros sequence and goes back to it with the smallest accumulated cost.



D_{min} allows computing the remaining parameters that have an impact on the performance

$$D_{min} = 5 \Rightarrow \begin{cases} \kappa_2 = 1 \\ z = 3 \end{cases}$$

Soft decoding

For the sake of simplicity, let us assume antipodal modulation i.e., $A[l] = \pm A$.

Notation

$$B_{0,:} = \{B[0], B[1], B[2], \cdots\} \equiv \text{input bits}$$
$$q_{0,:} = \{q[0], q[1], q[2], \cdots\} \equiv \text{soft estimates}$$

2 possibilities

Soft decoding

For the sake of simplicity, let us assume antipodal modulation i.e., $A[l] = \pm A$.

Notation

$$B_{0,:} = \{B[0], B[1], B[2], \cdots\} \equiv \text{input bits}$$
$$q_{0,:} = \{q[0], q[1], q[2], \cdots\} \equiv \text{soft estimates}$$

2 possibilities

- **Sequence** soft decoding: it minimizes the sequence error probability

$$\hat{B}_{0,:} = \arg\max_{B_{0,:}} p(q_{0,:}|B_{0,:})$$

Soft decoding

For the sake of simplicity, let us assume antipodal modulation i.e., $A[l] = \pm A$.

Notation

$$B_{0,:} = \{B[0], B[1], B[2], \cdots\} \equiv \text{input bits}$$
$$q_{0,:} = \{q[0], q[1], q[2], \cdots\} \equiv \text{soft estimates}$$

2 possibilities

- **Sequence** soft decoding: it minimizes the sequence error probability

$$\hat{B}_{0,:} = \arg\max_{B_{0,:}} p(q_{0,:}|B_{0,:})$$

- **Bitwise** soft decoding: it minimizes the bit error probability

$$\hat{B}[i] = \arg\max_{B[i]} p(B[i]|q_{0,:}), i = 0, 1, \cdots$$

Sequence soft decoding

ML rule ≡ MAP rule

$$\hat{B}_{0,:} = \arg\max_{B_{0,:}} p(q_{0,:}|B_{0,:}) = \arg\max_{B_{0,:}} \prod_{l} p(q[l]|B[l])$$
$$= \arg\min_{B_{0,:}} \sum_{l} -\log p(q[l]|B[l])$$

where

$$p(q[l]|B[l]) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(q[l]-A[l])^2}{2\sigma^2}\right).$$

Implemented by means of the Viterbi algorithm

Bitwise soft decoding

MAP rule

We apply the maximum a posteriori (MAP) rule at the bit level to compute

$$P(B[l] = 1 | q_{0,:})$$

and we decide

$$\hat{B}[l] = \begin{cases} 1 & \text{if } p(B[l] = 1 | q_{0,:}) > P(B[l] = 0 | q_{0,:}) \\ 0 & \text{otherwise} \end{cases}$$

Implemented by means of **BCJR** (Bahl, Cocke, Jelinek and Raviv) algorithm

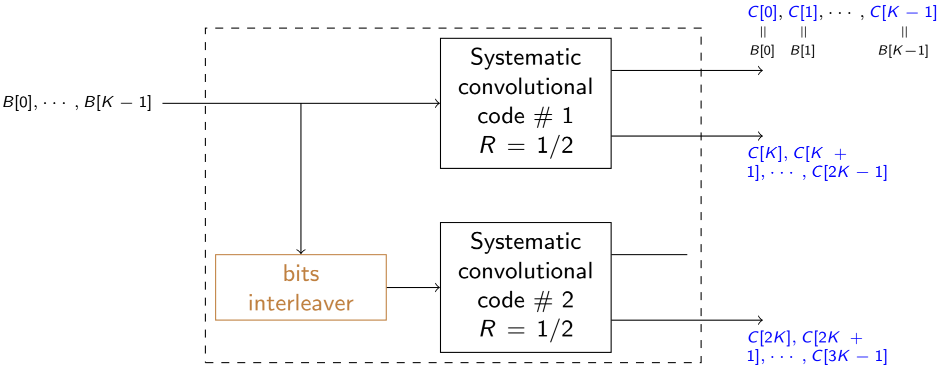
Index

Turbo codes

- They are built by composing two convolutional codes that operate over bits ordered differently.

Turbo codes

- They are built by composing two convolutional codes that operate over bits ordered differently.
- Main elements:
  - 2 convolutional codes
  - bit interleaver



Remarks

- The interleaver is used to increase the memory of the system without increasing the decoding complexity.

Remarks

- The interleaver is used to increase the memory of the system without increasing the decoding complexity.
- Good performance in the low-SNR region (at about 0.7 dBs from Shannon limit).

Remarks

- The interleaver is used to increase the memory of the system without increasing the decoding complexity.
- Good performance in the low-SNR region (at about 0.7 dBs from Shannon limit).
- Regarding decoding...
  - It is iterative since the decoding of the original sequence and the *shuffled* one must agree.
  - Relies on BCJR algorithm.