Channel Coding Theory:

Convolutional Codes.

Laura Cottatellucci

EURECOM

laura.cottatellucci@eurecom.fr

Outlines

I. Motivations
II. Definition of Convolutional Codes
III. Representations of Convolutional Codes
IV. Soft-Input Decoding
V. Free distance
VI. Performance Bounds
VII. Soft Output Decoding
VIII. References

Why to Study Convolutional Codes

(1) **Convolutional codes can be transformed into linear block codes by trellis termination.**

(2) **Convolutional codes exhibit favorable properties:**

- **Efficient encoding of long codewords using shift registers.**
- **Efficient soft-input decoding using Viterbi-algorithm.**
- **Efficient soft-output decoding using BCJR-algorithm (Bahl, Cocke, Jelinek, Raviv).**

(3) **Used in deep space/satellite and mobile communications.**

(4) **The classic turbo codes are based on convolutional codes.**

Convolutional Codes & Linear Block Codes

A linear code is obtained by selecting 2^k vectors in $GF(2^n)$ such that their symbols satisfy the linear constraints

$$\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G},$$

in other words there is a dependency among symbols of a codeword. This dependency can be enforced by a linear time invariant system.

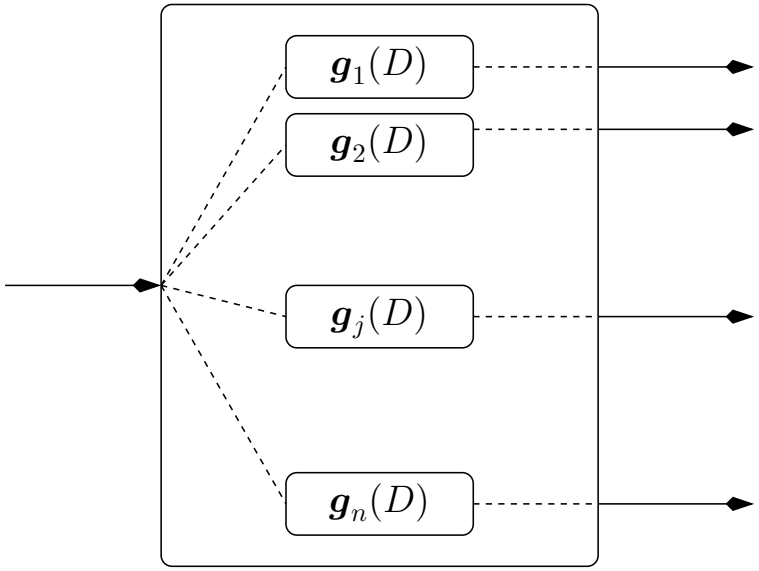
Example

Consider a single-input n-output linear invariant systems with finite memory defined by the impulse responses

$$\boldsymbol{g}_j = (g_{0,j}, g_{1,j}, \ldots, g_{\nu,j}), \qquad 1 \leq j \leq n$$

or, equivalently,

$$\boldsymbol{g}_j(D) = g_{0,j} + g_{1,j}D + \ldots + g_{\nu,j}D^{\nu}, \qquad 1 \leq j \leq n.$$



Example (Cntd)

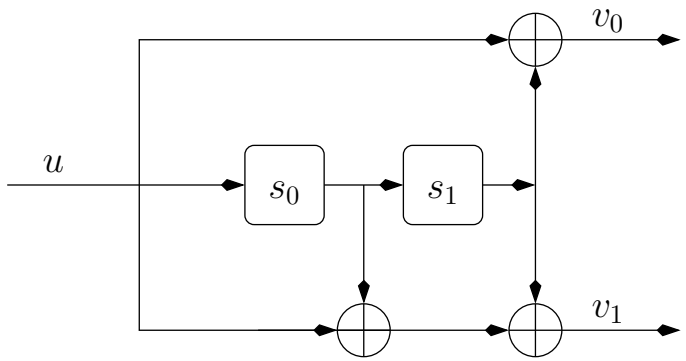
Using the polynomial representation (in this context D-transform) of $\boldsymbol{g}_j$ and $\boldsymbol{u}$, $\boldsymbol{g}_j(D)$ and $\boldsymbol{u}(D)$, respectively, the codeword polynomial in the D-transform domain is

$$\boldsymbol{v}(D) = \boldsymbol{u}(D)[\boldsymbol{g}_1(D), \boldsymbol{g}_2(D), \ldots, \boldsymbol{g}_n(D)]$$

Let us consider the code with generator sequences* (octal notation) $(5, 7)_8$

$$\boldsymbol{v}(D) = \boldsymbol{u}(D)[\, 1 + D^2 \;\; 1 + D + D^2 \,]$$

Block diagram of the encoder $(5, 7)_8$.



We can generalize this structure by considering a linear time-invariant system with k-inputs and n-outputs.

* Note that when talking about convolutional codes we refer to sequences instead of polynomials, e.g. generator sequences instead of generator polynomials.

Example (Cntd 2)

Encoding of a sequence $\boldsymbol{u} = (u_0, u_1, \ldots, u_{K-1})$ of length K

- Append to $\boldsymbol{u}$ a sequence of zeros 0_ν of length ν.
- Input the sequence $\widetilde{\boldsymbol{u}} = (\boldsymbol{u}, 0_\nu)$ to the linear invariant system $\boldsymbol{g}_j$ to obtain a sequence of length $K + \nu$. The convolution of the sequence $\boldsymbol{u}(D)$ with the impulsive response $\boldsymbol{g}_j(D)$ is also obtained by multiplying $\widetilde{\boldsymbol{u}}$ by the Toeplitz $K \times (K + \nu)$ matrix

$$\boldsymbol{G}_j = \begin{bmatrix} g_{0,j} & g_{1,j} & \ldots & g_{\nu,j} & 0 & \ldots & \ldots & \ldots & \ldots & 0 \\ 0 & g_{0,j} & \ldots & g_{\nu-1,j} & g_{\nu,j} & \ldots & \ldots & \ldots & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots & \ldots & 0 & g_{0,j} & g_{1,j} & \ldots & g_{\nu,j} \end{bmatrix}$$

- The codeword $\boldsymbol{v}$ of the single-input n-output system is obtained by concatenating the n output sequences

$$\widetilde{\boldsymbol{u}}\boldsymbol{G}_1, \widetilde{\boldsymbol{u}}\boldsymbol{G}_2, \ldots, \widetilde{\boldsymbol{u}}\boldsymbol{G}_n.$$

The length of $\boldsymbol{v}$ is $N = (n(K + \nu))$.

Convolutional Codes: Definition

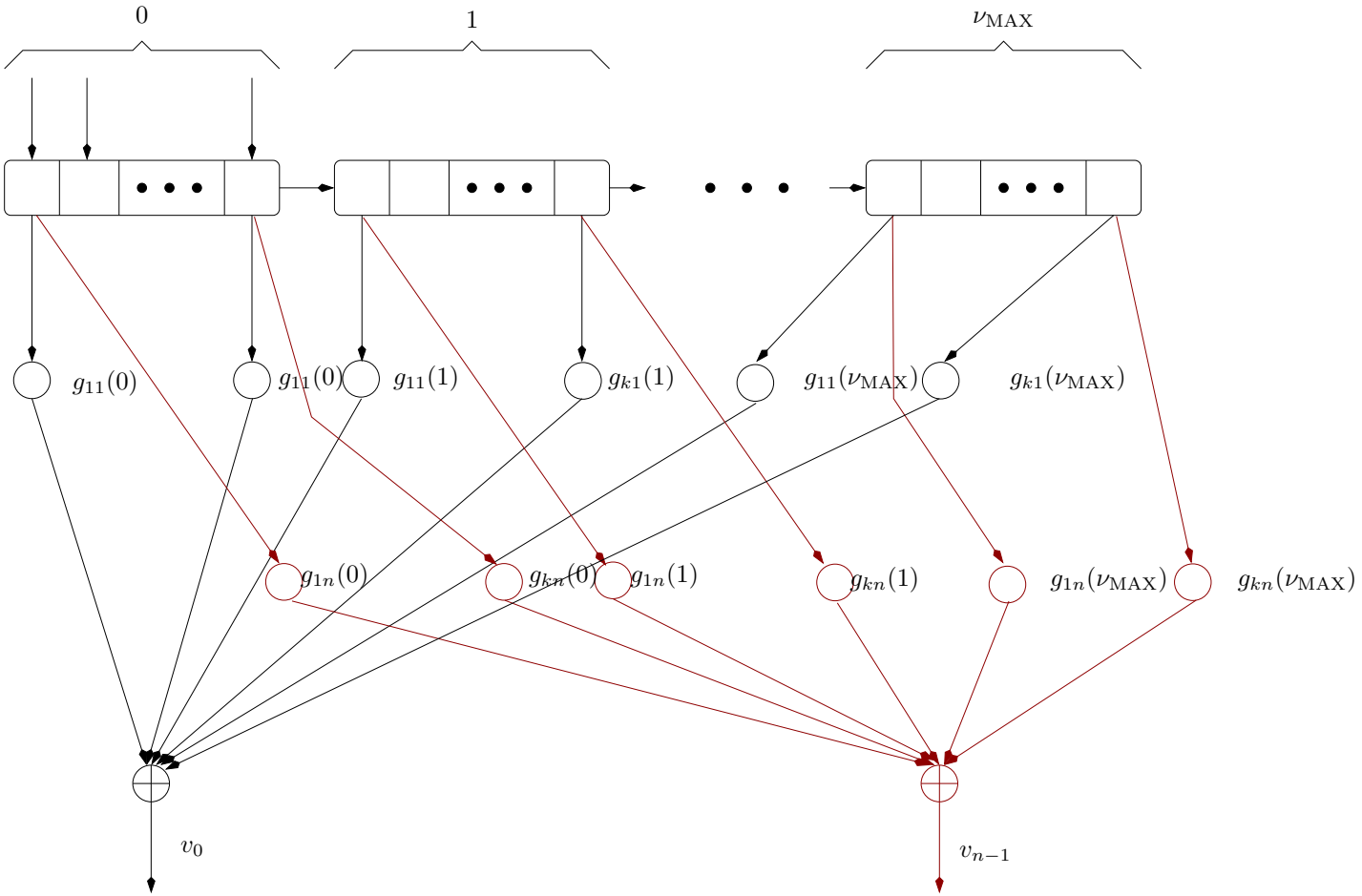
Let a k-input n-output linear time-invariant system be defined by the matrix of the impulse responses in the D-domain

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

and let $\nu_1, \nu_2, \dots \nu_k$ be the maximum degree of the polynomials in rows from 1 to k of $\boldsymbol{G}(D)$ with $\nu_{\mathrm{MAX}} = \max(\nu_1, \nu_2, \dots, \nu_k)$.
Furthermore, let $\boldsymbol{u}_1(D), \boldsymbol{u}_2(D), \dots, \boldsymbol{u}_k(D)$ be the k information sequences input of the linear time-invariant system. If each input sequence is of length $K + \nu_{\mathrm{MAX}}$, with ν_{MAX} tailing zeros, the convolutional code is obtained by concatenation of the n sequences

$$\boldsymbol{v}(D) = (\ \boldsymbol{u}_1(D)\ \ \boldsymbol{u}_2(D)\ \ \dots\ \ \boldsymbol{u}_k(D)\ )\boldsymbol{G}(D)$$

Block Diagram of a Feedforward Encoder

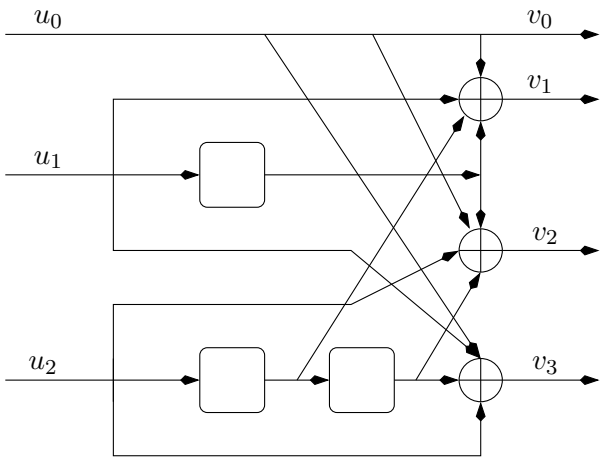


$g_{\ell m}(j)$ **is the coefficient of the monomial** X^j **in** $\boldsymbol{g}_{\ell m}(D)$.

Example

A $(4, 3)$ binary nonsystematic feedforward convolutional encoder

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



Generator Matrix

$$\boldsymbol{G}_0 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad \boldsymbol{G}_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad \boldsymbol{G}_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{G} = \begin{pmatrix} \boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots \\ \boldsymbol{0} & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \boldsymbol{0} & \boldsymbol{0} & \dots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{G}_0 & \boldsymbol{G}_1 & \boldsymbol{G}_2 & \boldsymbol{0} & \dots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

Equivalent Definition of Convolutional Codes

An (n, k, ν) convolutional code is the row-space of the encoder generator matrix G.

Some Definitions

The code rate is $R = \frac{kK}{n(K+\nu_{\mathrm{MAX}})} \underset{\nu_{\mathrm{MAX}} \ll K}{\approx} \frac{k}{n}$.

ν_{MAX} **is the memory order length of the convolutional code.**

The overall constraint length ν **of the encoder is defined as** $\nu = \sum_{i=1}^{k} \nu_i$, **that is,** ν **is the sum of the length of all shift registers.**

A rate $R = \frac{k}{n}$ **convolutional encoder with overall constraint length** ν **is referred to as an** (n, k, ν) **encoder.**

Systematic Convolutional Codes

A convolutional encoder is systematic if, in each generated segment of n digits, the first k are a replica of the corresponding message digits.

On the generator matrix this property is reflected in the following structure of the matrices $\boldsymbol{G}_i$

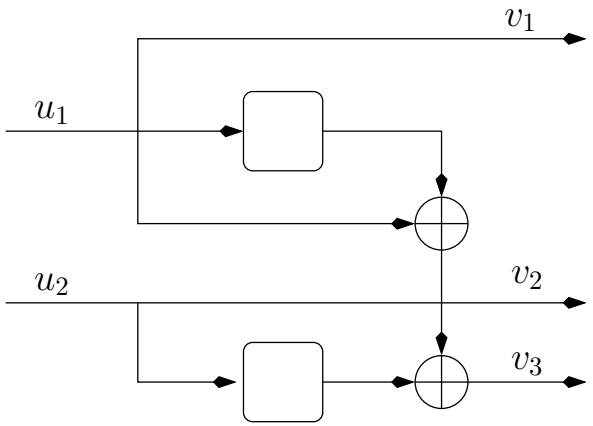
$$\boldsymbol{G}_0 = [\, \boldsymbol{I}_k \;\; \boldsymbol{P}_1 \,] \qquad \boldsymbol{G}_j = [\, \boldsymbol{0} \;\; \boldsymbol{P}_j \,] \quad 1 \le j \le \nu_{\mathrm{MAX}}$$

where $\boldsymbol{P}_i$, $1 \le i \le \nu_{\mathrm{MAX}}$, are the $k \times (n-k)$ parity check matrices.

Example

$$\boldsymbol{G}_0 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\boldsymbol{G}_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$



Representation as Finite State Machine

An encoder is a time invariant system with finite memory

$\Rightarrow$ **Its output depends only on its input and its state (content of registers)**

$\Rightarrow$ **It can be represented as a finite-state machine (FSM)**

$$\boldsymbol{s}_{i+1} = \boldsymbol{A}\boldsymbol{s}_i + \boldsymbol{B}\boldsymbol{u}_i^{(c)}$$
$$\boldsymbol{v}_i^{(c)} = \boldsymbol{C}\boldsymbol{s}_i + \boldsymbol{D}\boldsymbol{u}_i^{(c)}$$

where $\boldsymbol{s}_i$, $\boldsymbol{u}_i^{(c)}$, **and** $\boldsymbol{v}_i^{(c)}$ **are the** ν**-dimensional vector of the state at in time interval** i**, the** k**-dimensional vector of the input at time interval** i**, and the** n**-dimensional vector of the output at time instant** i**, respectively.** $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ **are matrices in** $GF(2)$ **of dimensions** $\nu \times \nu$, $\nu \times k$, $n \times \nu$, **and** $n \times k$, **respectively.**

Finite State Machine & Generator Matrix

The generator matrix can be obtained from the finite state machine representation

- **By using the D-transform of the FSM representation**

$$\boldsymbol{s} = \boldsymbol{A}D\boldsymbol{s} + \boldsymbol{B}D\boldsymbol{u}^{(c)}$$
$$\boldsymbol{v}^{(c)} = \boldsymbol{C}\boldsymbol{s} + \boldsymbol{D}\boldsymbol{u}^{(c)}$$

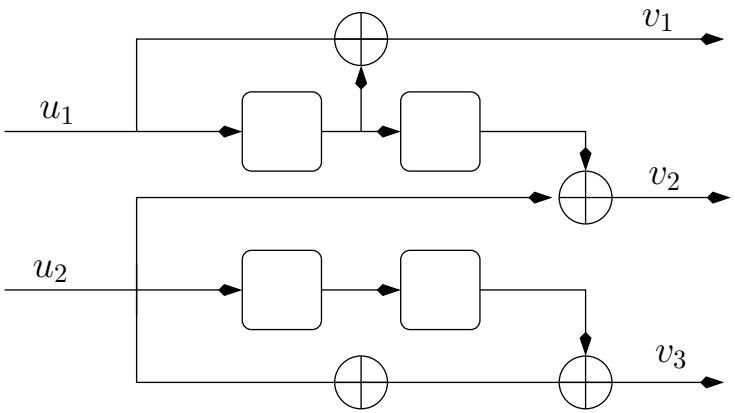
- **By expressing $v^{(c)}$ as a function of $u^{(c)}$**

$$\boldsymbol{s} = (\boldsymbol{I} - \boldsymbol{A}D)^{-1}\boldsymbol{B}D\boldsymbol{u}^{(c)}$$
$$\boldsymbol{v}^{(c)} = [\boldsymbol{C}(\boldsymbol{I}D^{-1} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}]\boldsymbol{u}^{(c)}$$

Therefore,

$$\boldsymbol{G}(D) = [\boldsymbol{C}(\boldsymbol{I}D^{-1} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D}]^T$$

Example



$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad \boldsymbol{D} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

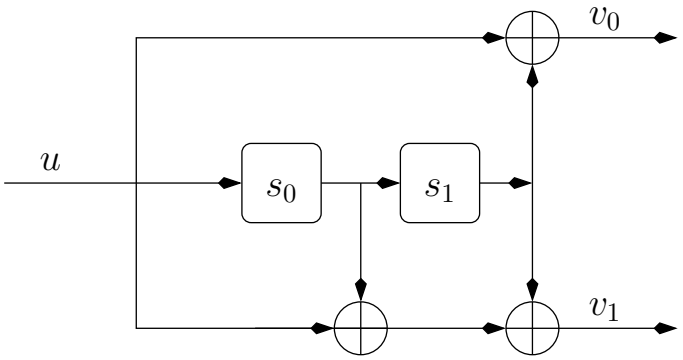
$$\boldsymbol{G}^T(D) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} D^{-1} & 0 & 0 & 0 \\ 1 & D^{-1} & 0 & 0 \\ 0 & 0 & D^{-1} & 0 \\ 0 & 0 & 1 & D^{-1} \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

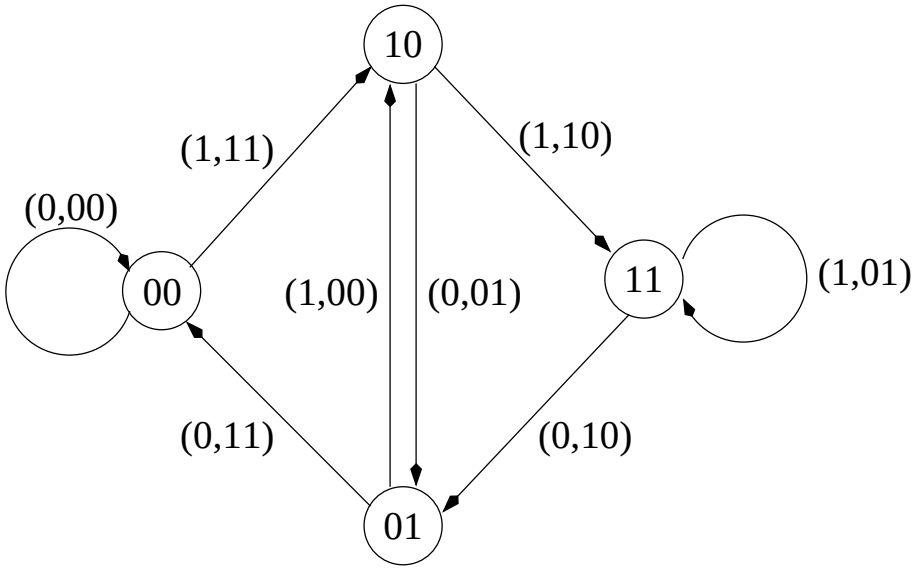
$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} D & 0 & 0 & 0 \\ D^2 & D & 0 & 0 \\ 0 & 0 & D & 0 \\ 0 & 0 & D^2 & D \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} D+1 & 0 \\ D^2 & 1 \\ 0 & 1+D+D^2 \end{pmatrix}$$

State Diagram Representation

The state diagram representation of a convolutional code $\mathcal{C}$ is a graph with 2^ν vertices corresponding to all possible states and 2^n edges connecting those vertices/states between which a transition is possible. Each edge (s, s') is labelled with the pair (u, v) where u is the input generating the transition and v is the resulting output.

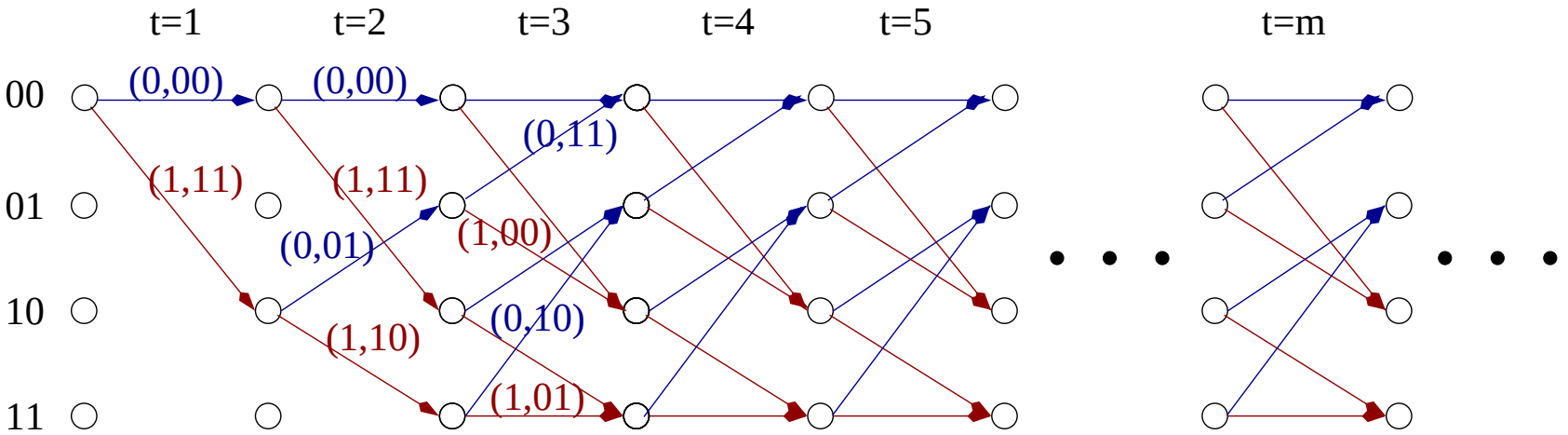
Example





Trellis Representation

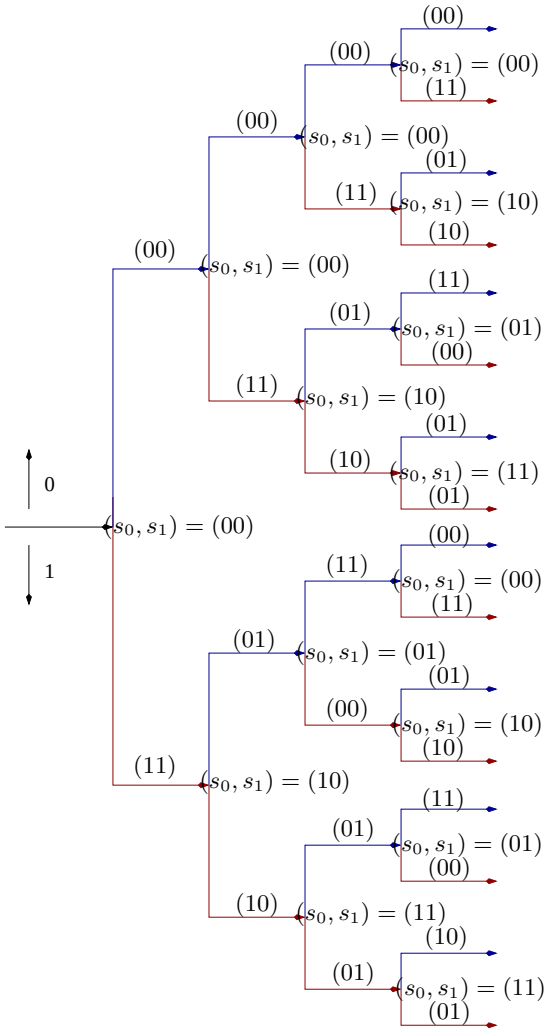
It is obtained from the state diagram by introducing the time dimension and representing the transition for the 2^ν states at time ℓ to the 2^ν states at time $\ell+1$
for each time instant.



- **In a steady state (not at the beginning or at the end of the trellis) the structure of the trellis is identical at each stage.**
- **Number of outgoing/incoming edges for each state:** 2^k.
- **Number of states:** 2^ν.

Tree Diagram

An equivalent representation of a convolutional code that makes explicit the time dependency is the tree diagram. However, its dimensions become quickly too large.



Maximum Likelihood Decoder

Assume

1. **Binary input and Q-ary output discrete memoryless channel (DMC).**
2. **Information sequences $\boldsymbol{u} = (\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{h-1})$ of length $K = kh$, equivalently $\boldsymbol{u} = (u_0, \ldots, u_{K-1})$, being $u_i,\ i = 0, \ldots K-1$ information bits.**
3. **Codewords $\boldsymbol{v} = (\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{h+\nu_{\mathrm{MAX}}-1})$ of length $N = n(h + \nu_{\mathrm{MAX}})$, ($\nu_{\mathrm{MAX}}$ is the memory order length), equivalently $\boldsymbol{v} = (v_0, \ldots, v_{N-1})$.**

The maximum likelihood decoder chooses the codeword $\widehat{\boldsymbol{v}}$ that maximizes the likelihood/log-likelihood function, i.e.

$$\widehat{\boldsymbol{v}} = \underset{\boldsymbol{v}}{\operatorname{argmax}} P(\boldsymbol{r}|\boldsymbol{v}) = \underset{\boldsymbol{v}}{\operatorname{argmax}} \prod_{\ell=0}^{h+\nu_{\mathrm{MAX}}-1} P(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = \underset{\boldsymbol{v}}{\operatorname{argmax}} \prod_{\ell=0}^{N-1} P(r_\ell|v_\ell)$$

$$= \underset{\boldsymbol{v}}{\operatorname{argmax}} \log P(\boldsymbol{r}|\boldsymbol{v}) = \underset{\boldsymbol{v}}{\operatorname{argmax}} \sum_{\ell=0}^{h+\nu_{\mathrm{MAX}}-1} \log P(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = \underset{\boldsymbol{v}}{\operatorname{argmax}} \sum_{\ell=0}^{N-1} \log P(r_\ell|v_\ell)$$

Maximum Likelihood Decoder (Cntd)

Metric of the codeword v: $$M(\boldsymbol{r}|\boldsymbol{v}) = \log P(\boldsymbol{r}|\boldsymbol{v})$$

Branch metric: $$M(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = \log P(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell)$$

Bit metric: $$M(r_\ell|v_\ell) = \log P(r_\ell|v_\ell)$$

$$M(\boldsymbol{r}|\boldsymbol{v}) = \sum_{\ell=0}^{h+\nu_{\mathrm{MAX}}-1} M(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = \sum_{\ell=0}^{N-1} M(r_\ell|v_\ell)$$

Partial path metric for the first t branches of a path

$$M([\boldsymbol{r}|\boldsymbol{v}]_t) = \sum_{\ell=0}^{t-1} M(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = \sum_{\ell=0}^{nt-1} M(r_\ell|v_\ell)$$

The Viterbi Algorithm

The Viterbi algorithm applied to the received sequence r finds the path through the trellis with the largest metric, that is, the maximum likelihood path.

It processes r recursively. At each unit time

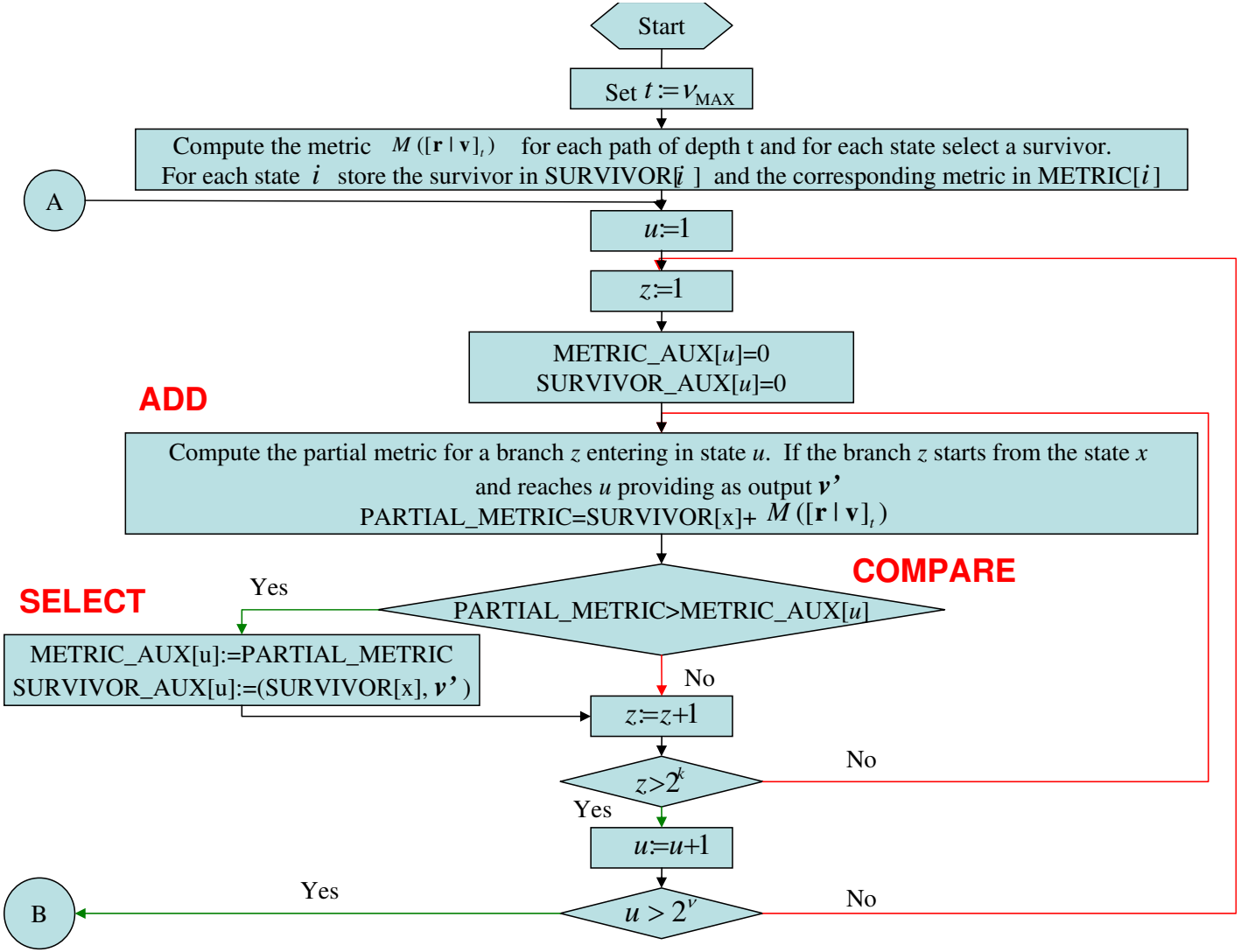
1. **It adds 2^k branch metrics to each previously stored path metric (ADD operation).**
2. **It compares the metrics of all 2^k paths entering each state (COMPARE operation).**
3. **It selects the path with the largest metric called the survivor (SELECT operation). The survivor at each state is stored along with its metric.**

Step 1. **Beginning at time unit $t = \nu_{\mathrm{MAX}}$ compute the partial metric for the single path entering each state. Store the survivor and its metric for each state.**

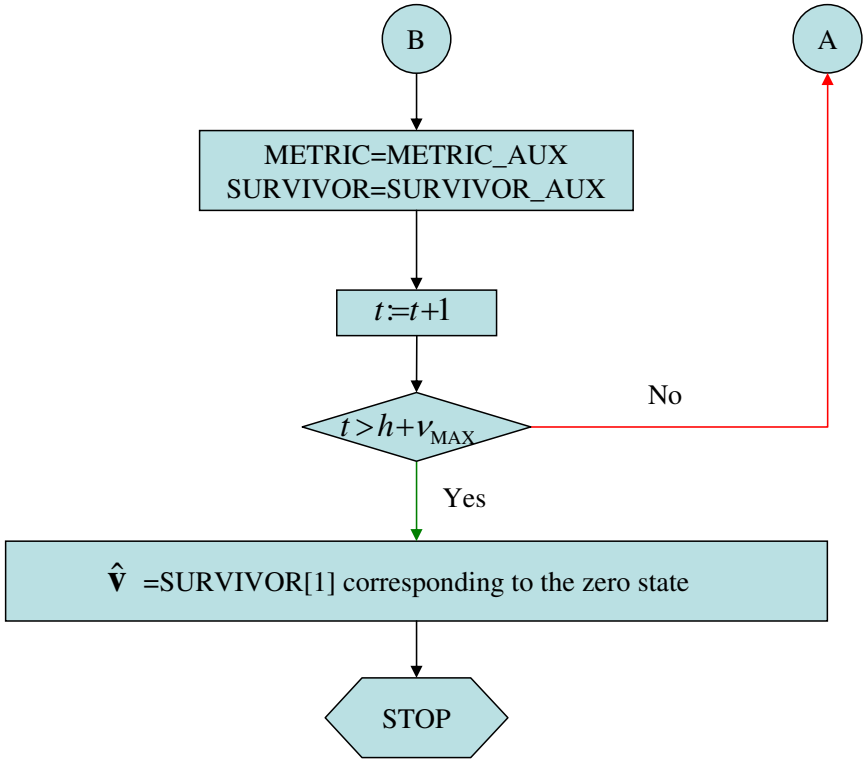
Step 2. **Increase t by 1. Compute the partial metric for all 2^k paths entering that state to the metric of the connecting survivor at the previous time unit. For each state, compare the metrics of all 2^k paths entering that state. Select the path with the largest metric, store it along with its metric and eliminate all other paths.**

Step 3. **If $t < h + \nu_{\mathrm{MAX}}$, repeat step 2; otherwise, stop.**

Viterbi Algorithm: Flow Chart



Viterbi Algorithm: Flow Chart (Cntd)



(*) This flow chart does not take into account the trellis termination.

Viterbi Algorithm and Maximum Likelihood Decoding

Theorem: **The final survivor $\widehat{v}$ in the Viterbi algorithm is the maximum likelihood path, that is**

$$M(\boldsymbol{r}|\widehat{\boldsymbol{v}}) \geq M(\boldsymbol{r}|\boldsymbol{v}) \qquad \forall \boldsymbol{v} \neq \widehat{\boldsymbol{v}}.$$

Proof:

Assume that the maximum likelihood path is eliminated by the Viterbi algorithm at time t. This implies that the partial path metric of the survivor exceeds the partial path metric of the maximum likelihood path at this point. If we append the remaining portion of the maximum likelihood path onto the survivor at time unit t, the total metric of this path will exceed the total metric of the maximum likelihood path. This contradicts the definition of maximum likelihood path. Hence, the maximum likelihood path cannot be eliminated by the Viterbi algorithm and must be the final survivor.

Viterbi Algorithm: Remarks

The branch metric

$$M(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = \log P(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell)$$

can be replaced by

$$\widetilde{M}(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = c_2[\log P(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) + c_1]$$

with $c_1 \in \mathbb{R}$ and $c_2 \in \mathbb{R}^+$ (real positive)

Often c_1 and c_2 are chosen as follows:

- **c_1 is chosen to make the smallest metric equal to zero.**
- **c_2 is chosen such that all metrics can be approximated by integers (slight sub-optimality due to the approximation by integers).**

Viterbi Algorithm: Metrics for a BSC

For a BSC

$$P(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = p^{d(\boldsymbol{r}_\ell,\boldsymbol{v}_\ell)}(1-p)^{n-d(\boldsymbol{r}_\ell,\boldsymbol{v}_\ell)}$$

can be replaced by

$$\widetilde{M}(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = c_2[\log P(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) + c_1]$$

with $c_1 \in \mathbb{R}$ and $c_2 \in \mathbb{R}^+$ (real positive)

Often c_1 and c_2 are chosen as follows:

- **c_1 is chosen to make the smallest metric equal to zero.**
- **c_2 is chosen such that all metrics can be approximated by integers (slight sub-optimality due to the approximation by integers).**

Viterbi Algorithm: Metrics for a BSC (cntd)

For a BSC

$$P(\boldsymbol{r}|\boldsymbol{v}) = p^{d(\boldsymbol{r},\boldsymbol{v})}(1-p)^{N-d(\boldsymbol{r},\boldsymbol{v})}$$

being $d(\boldsymbol{r},\boldsymbol{v})$ the Hamming distance between $\boldsymbol{r}$ and $\boldsymbol{v}$. Thus,

$$\log P(\boldsymbol{r}|\boldsymbol{v}) = d(\boldsymbol{r},\boldsymbol{v}) \log\left(\frac{p}{1-p}\right) + N\log(1-p).$$

Since for $p < \frac{1}{2}$, $\log\frac{p}{1-p} < 0$ and $N\log(1-p)$ are constant for all $\boldsymbol{v}$ the maximum likelihood detector minimizes$^{(*)}$ the Hamming distance.

For the Viterbi algorithm, the branch metric is $d(\boldsymbol{r}_\ell, \boldsymbol{v}_\ell)$, the bit metric is $d(r_\ell, v_\ell)$.

The survivors at time unit t and state s' minimizes the partial path metric for the first t branches, i.e. the Hamming distance, for all the paths that reach s' at time unit t.

$^{(*)}$ The minimization instead of the maximization is due to the fact that $\log\frac{p}{1-p} < 0$.

Viterbi Algorithm: Metrics for a BIAWGN

We assume a binary antipodal modulation.

$\Rightarrow$ **The elements of the codeword** $\boldsymbol{v} = (v_0, v_1, \ldots, v_{N-1})$ **assume values** ± 1 **according to the mapping** $1 \to +1$ **and** $0 \to -1$

$\Rightarrow$ **Conditional probability density function of the received symbol** r_ℓ **given the transmitted bit** v_ℓ

$$p(r_\ell|v_\ell) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2\sigma^2}(r_\ell - v_\ell)^2}.$$

Then,

$$\begin{aligned} M(\boldsymbol{r}|\boldsymbol{v}) &= \ln p(\boldsymbol{r}|\boldsymbol{v}) = \ln \prod_{\ell=0}^{N-1} p(r_\ell|v_\ell) = \sum_{\ell=0}^{N-1} \ln p(r_\ell|v_\ell) \\ &= -\frac{1}{2\sigma^2} \sum_{\ell=0}^{N-1} (r_\ell - v_\ell)^2 - \ln \sqrt{2\pi\sigma} = \frac{1}{2\sigma^2} \sum_{\ell=0}^{N-1} r_\ell v_\ell - \frac{1}{2\sigma^2} \sum_{\ell=0}^{N-1} (r_\ell^2 + 1) - \ln \sqrt{2\pi\sigma} \\ &= C_1 \boldsymbol{r}\boldsymbol{v}^T + C_2 \end{aligned}$$

with $C_1 = \frac{1}{2\sigma^2}$ **and** $C_2 = -\frac{1}{2\sigma^2}(\|\boldsymbol{r}\|^2 + N) - \ln \sqrt{2\pi\sigma}.$

Viterbi Algorithm: Metrics for a BIAWGN (Cntd)

Metric of the codeword v: $\widetilde{M}(\boldsymbol{r}|\boldsymbol{v}) = \boldsymbol{r}\boldsymbol{v}^T$

Branch metric: $\widetilde{M}(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = \boldsymbol{r}_\ell\boldsymbol{v}_\ell^T$

Bit metric: $\widetilde{M}(r_\ell|v_\ell) = r_\ell v_\ell^T$

In a BSC the codeword metric is the Hamming distance between received signal and some codeword...

In a BIAWGN the codeword metric is the correlation between the received signal and any codeword or, equivalently their Euclidian distance!

Free Distance

As for the linear block the performance (error detection and error correction capabilities) depends on the Hamming distances between pairs of codewords.

The free distance of a convolutional code is defined as

$$d_{\text{free}} = \min_{\boldsymbol{U}',\boldsymbol{U}''} \{d(\boldsymbol{V}', \boldsymbol{V}'') : \boldsymbol{U}' \neq \boldsymbol{U}''\}$$

where $\boldsymbol{V}'$ and $\boldsymbol{V}''$ are the codewords corresponding to the information sequences $\boldsymbol{U}'$ and $\boldsymbol{U}''$

Because of the linearity of the convolutional codes

$$\begin{aligned} d_{\text{free}} &= \min_{\boldsymbol{U}',\boldsymbol{U}''} \{w(\boldsymbol{V}' + \boldsymbol{V}'') : \boldsymbol{U}' \neq \boldsymbol{U}''\} \\ &= \min_{\boldsymbol{U}',\boldsymbol{U}''} \{w(\boldsymbol{V}) : \boldsymbol{U} = \boldsymbol{U}' + \boldsymbol{U}'' \neq \boldsymbol{0}\} \\ &= \min_{\boldsymbol{U}',\boldsymbol{U}''} \{w(\boldsymbol{UG}) : \boldsymbol{U} = \boldsymbol{U}' + \boldsymbol{U}'' \neq \boldsymbol{0}\} \end{aligned}$$

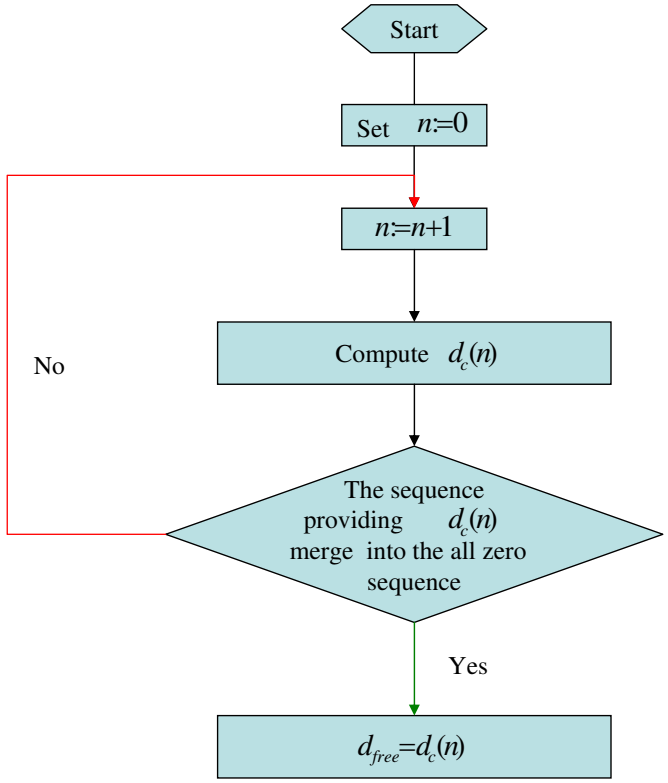
d_{free} is the minimum weight codeword produced by any finite-length nonzero information sequence. Equivalently, it is the minimum weight of all finite paths in the state diagram that diverge from and merge again with the all-zero state.

Column Distance and Free Distance

Consider a pair of encoded sequences up to the depth ℓ into the code trellis and assume that they are different at the first branch. The ℓ-th order column distance $d_c(\ell)$ is the minimum distance between all pairs of such sequences.

For a non catastrophic encoder it can be shown that

$$\lim_{\ell \to \infty} d_c(\ell) = d_{\text{free}}$$



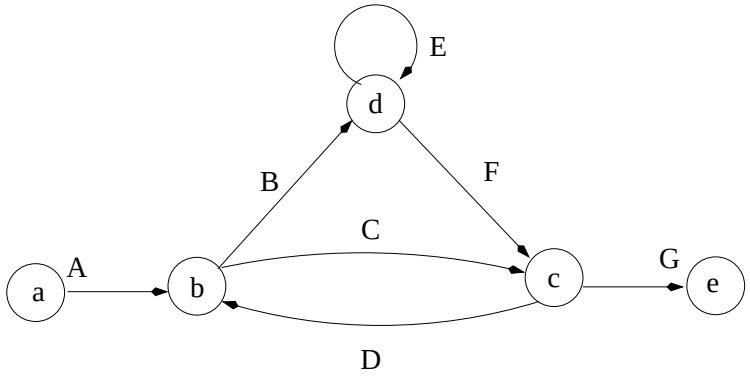
Transfer Function of Directed Graphs

A directed graph or digraph is an ordered pair $\mathcal{G} = (V, A)$ where

- **V is a set whose elements are called vertices or nodes;**
- **A is a set of ordered pairs of vertices, called directed edges, or arcs, or arrows.**

The transfer function between a pair of vertices is defined as the sum of the labels of all paths of any length connecting the two vertices.

Transfer Function of Directed Graphs



Let $x_i,\ i = b, c, d$ be the value of the accumulated path from the initial vertex a to the vertex $i = b, c, d$.
State equations (one for each node different from a and e)

$$\begin{cases} x_b = A + Dx_c, \\ x_c = Cx_b + Fx_d, \\ x_d = Bx_b + Ex_d \end{cases}$$

Transfer Function: Example (cntd)

State Equations

$$\boldsymbol{x} = \boldsymbol{T}\boldsymbol{x} + \boldsymbol{x}_0 \qquad \Rightarrow \qquad \boldsymbol{x} = (\boldsymbol{I} - \boldsymbol{T})^{-1}\boldsymbol{x}_0 = (\boldsymbol{I} + \boldsymbol{T} + \boldsymbol{T}^2 + \boldsymbol{T}^3 + \ldots)\boldsymbol{x}_0$$

with

$$\boldsymbol{x} = \begin{pmatrix} x_b \\ x_c \\ x_d \end{pmatrix} \quad \boldsymbol{T} = \begin{pmatrix} 0 & D & 0 \\ C & 0 & F \\ B & 0 & E \end{pmatrix} \quad \boldsymbol{x}_0 = \begin{pmatrix} A \\ 0 \\ 0 \end{pmatrix}$$

The transfer function from a to e is given by

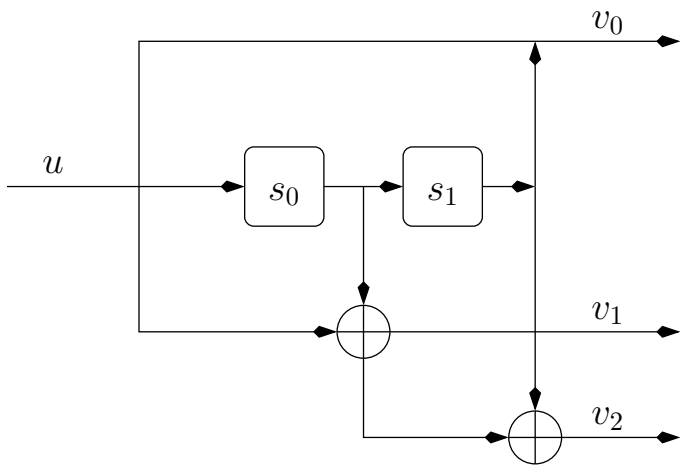
$$T(a,e) = Gx_c = G\frac{\det\begin{pmatrix} 1 & A & 0 \\ -C & 0 & -F \\ -B & 0 & 1-E \end{pmatrix}}{\det\begin{pmatrix} 1 & -D & 0 \\ -C & 1 & -F \\ -B & 0 & 1-E \end{pmatrix}} = \frac{ACG(1-E) - ABFG}{1 - E - BFD - CD + CDE}$$

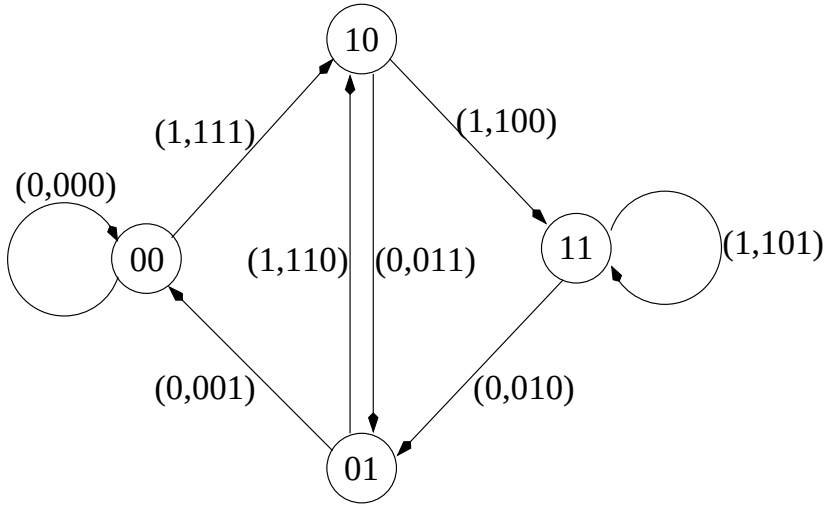
Weight Enumerating Function

Object: To count all paths that start from all-zero state, diverge at depth 1 in the trellis and subsequently merge again in the all-zero state

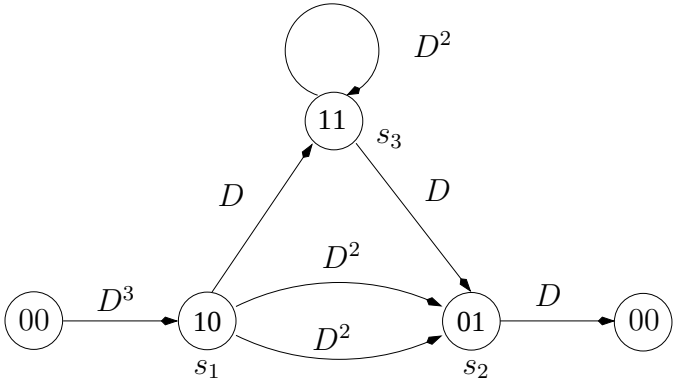
1. **Redraw the state diagram of convolutional codes with the following changes:**
   - **Eliminate the loop in the all-zero state.**
   - **Replace the label (u, v) with $D^{w(v)}$.**
   - **Split the all-zero state into two states, one representing the input node and the other representing the output node of the state diagram.**
2. **Calculate the transfer function from the input node to the output node in the digraph modified in 1.**
3. **Eventually, expand the denominator of the transfer function in a Laurent series to obtain a series in D: the weight enumerating function (WEF). The coefficient of the monomial D^ℓ is the number of output sequences in the graph having distance ℓ from the all-zero sequence.**
4. **If the monomial with lowest exponent is D^ℓ assign ℓ to d_{free}**

Example


v_0
u
s_0
s_1
v_1
v_2


10
(1,111)
(1,100)
(0,000)
00
(1,110)
(0,011)
11
(1,101)
(0,001)
(0,010)
01

Example (cntd)



$$\begin{cases} s_1 = D^2 s_2 + D^3, \\ s_2 = D^2 s_1 + D s_3, \\ s_3 = D s_1 + D^2 s_3 \end{cases}$$

$$s_2 = \frac{\det \begin{pmatrix} 1 & D^3 & 0 \\ -D^2 & 1 & -D \\ -D & 0 & 1-D^3 \end{pmatrix}}{\det \begin{pmatrix} 1 & -D^2 & 0 \\ -D^2 & 1 & -D \\ -D & 0 & 1-D^3 \end{pmatrix}} = \frac{D^5(2-D^2)}{1-(D^2+2D^4-D^6)}$$

Laurent expansion

$$\frac{1}{1-(D^2+2D^4-D^6)} = 1 + D^2 + 3D^4 + 4D^6 + 9D^8 + 14D^{10} + \ldots$$

Weight Enumerating Function

$$A = Ds_2 = D^6(2 + D^2 + 5D^4 + 5D^6 + 14D^8 + 19D^{10} + \ldots)$$

From the WEF we determine the free distance: $d_{\text{free}} = 6$.

Input Output Weight Enumerating Function

Object: For all the paths that start from all-zero state, diverge at depth 1 in the trellis, and subsequently merge again in all-zero state

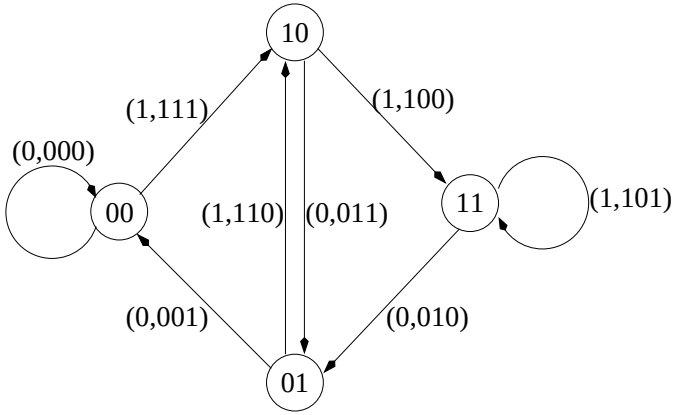
- **We want to determine the Hamming distance from all-zero sequence.**
- **We want to determine the length of the paths.**
- **We want to determine the weights of the input sequences that generate such paths.**

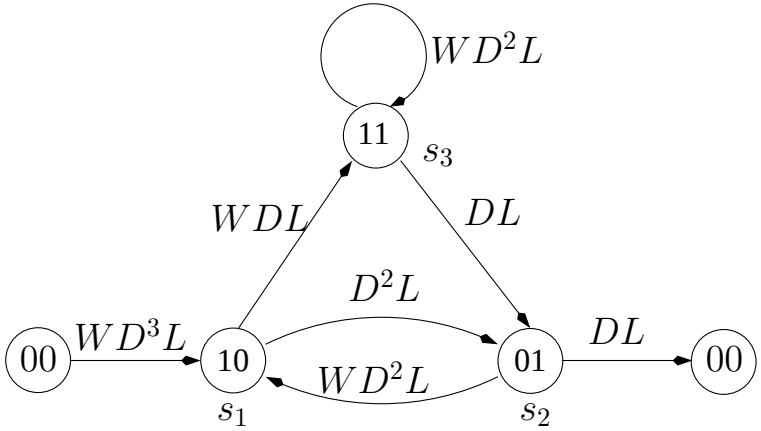
We proceed as for determining the weight enumerating function. The only difference is in the labelling of the modified graph.

The modified graph is labelled as follows. Each label contains 3 indeterminates :

- W **: its exponent is the weight of the input data frame causing the transition.**
- D **: it is the exponent of the weight of the output sequence.**
- L **: it is present in each edge with exponent 1 and counts the length of the path.**

Example





$$\begin{cases} s_1 = WD^2Ls_2 + WD^3L, \\ s_2 = D^2Ls_1 + DLs_3, \\ s_3 = WDLs_1 + WD^2Ls_3 \end{cases}$$

$$s_2 = \frac{\det\begin{pmatrix} 1 & WD^3L & 0 \\ -D^2L & 0 & -DL \\ -WDL & 0 & 1-WD^2L \end{pmatrix}}{\det\begin{pmatrix} 1 & -WD^2L & 0 \\ -D^2L & 1 & -DL \\ -WDL & 0 & 1-WD^2L \end{pmatrix}} = \frac{WD^3L(WD^2L^2 + D^2L(1-WD^2L))}{1 - WD^2L - WD^2L(WD^2L^2 + D^2L(1-WD^2L))}$$

Performance bounds

Framework:

- **BSC with error probability ε.**
- **Non systematic feedforward encoder**

$$G(D) = (\, 1 + D \;\; 1 + D^2 \;\; 1 + D + D^2 \,)$$

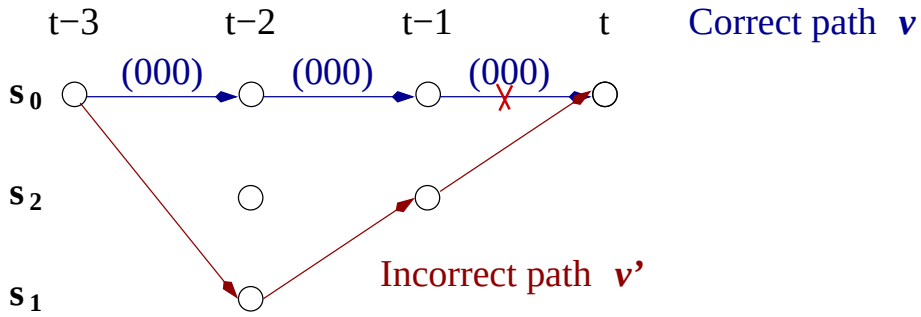
having IOWEF

$$\begin{aligned} A(W, D, L) &= \frac{D^7 W L^3}{1 - DWL(1 + D^2 L)} \\ &= D^7 W L^3 + D^8 W^2 L^4 + D^9 W^3 L^5 + D^{10}(W^2 L^2 + W^4 L^6) + \ldots \end{aligned}$$

- **Transmission of the all-zero codeword $\boldsymbol{v} = \mathbf{0}$.**
- **Focus on the first event error, i.e. the first error made at an arbitrary time unit t when the all-zero path (the correct path) is eliminated for the first time at time t in favor of a competitor (incorrect path).**

Object: Determine the bound on the first event error at time unit t.

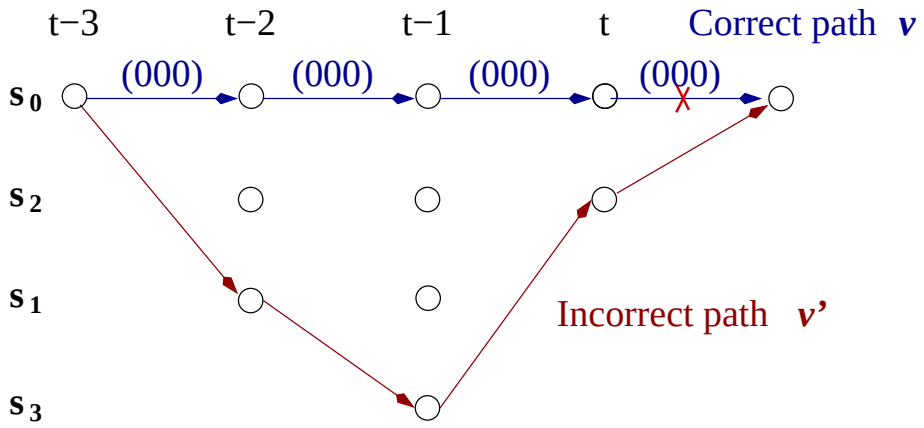
Performance: First Event Error at Time Unit t



From $A(W, D, L)$ we know that there is a path of length 3 diverging from the all-zero path, merging with it again, and having concatenated labels of weight 7. The Viterbi algorithm selects v' if the binary received sequence agrees with the incorrect path in four or more of the positions with 1's.

$$P_7 = P[\text{four or more 1's in seven positions}]$$
$$= \sum_{e=4}^{7} \binom{7}{e} \varepsilon^e (1-\varepsilon)^{7-e}.$$

Performance: First Event Error at Time Unit t (cntd)



Similarly, from $A(W, D, L)$ we know that there is a path of length 4 diverging from the all-zero path, merging with it again, and having concatenated labels of weight 8. The probability that the Viterbi algorithm selects the incorrect path is

$$P_8 = \frac{1}{2}P[\text{four 1's in eight positions}] + P[\text{five or more 1's in eight positions}]$$
$$= \frac{1}{2}\binom{8}{4}\varepsilon^4(1-\varepsilon)^4 + \sum_{e=5}^{8}\binom{8}{e}\varepsilon^e(1-\varepsilon)^{8-e}.$$

First Event Error at Time Unit t (cntd)

In general, assuming that the incorrect path has weight d, a first event error is made with probability

$$
\begin{aligned}
P_d &= \begin{cases} \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} \varepsilon^e (1-\varepsilon)^{d-e} & \text{for } d \text{ odd}, \\ \frac{1}{2}\binom{d}{\frac{d}{2}} \varepsilon^{\frac{d}{2}}(1-\varepsilon)^{\frac{d}{2}} + \sum_{e=\frac{d}{2}+1}^{d} \binom{d}{e} \varepsilon^e (1-\varepsilon)^{d-e} & \text{for } d \text{ even.} \end{cases} \\
&< \begin{cases} \varepsilon^{\frac{d}{2}}(1-\varepsilon)^{\frac{d}{2}} \sum_{e=\frac{d+1}{2}}^{d} \binom{d}{e} & \text{for } d \text{ odd}, \\ \varepsilon^{\frac{d}{2}}(1-\varepsilon)^{\frac{d}{2}} \sum_{e=\frac{d}{2}}^{d} \binom{d}{e} & \text{for } d \text{ even} \end{cases} \\
&< \varepsilon^{\frac{d}{2}}(1-\varepsilon)^{\frac{d}{2}} \sum_{e=0}^{d} \binom{d}{e} \\
&= 2^d \varepsilon^{\frac{d}{2}}(1-\varepsilon)^{\frac{d}{2}}
\end{aligned}
$$

First Event Error at Time Unit t (cntd)

Because all incorrect paths of length t or less can cause a first event error at time t, the first event error probability at time unit t, $P_f(E,t)$ is upperbounded, using the union bound, by the sum of the error probabilities of each of these paths. By considering also all incorrect paths of length greater than t branches

$$P_f(E,t) < \sum_{d=d_{\text{free}}}^{+\infty} A_d P_d$$

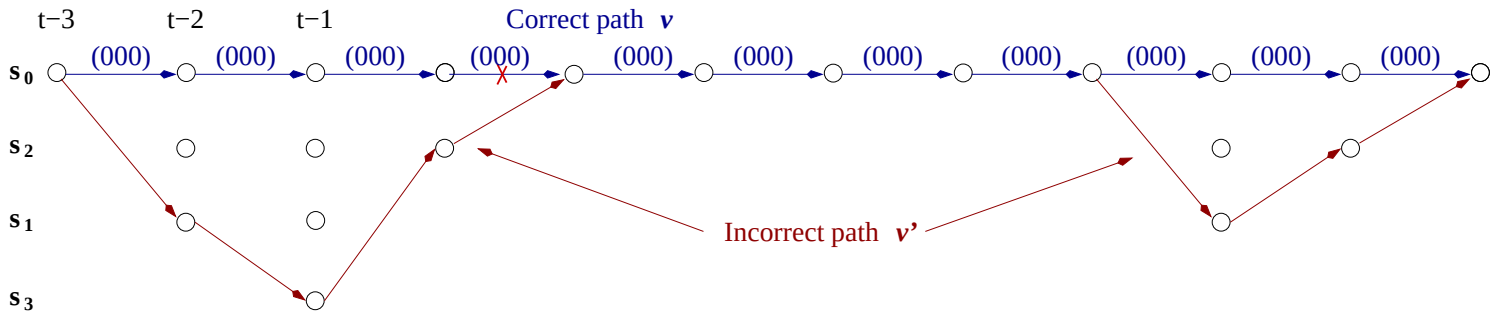
being A_d the number of codewords of weight d. For any t

$$\begin{aligned} P_f(E) &< \sum_{d=d_{\text{free}}}^{+\infty} A_d (2\sqrt{\varepsilon(1-\varepsilon)})^d \\ &= A(D)|_{D=2\sqrt{\varepsilon(1-\varepsilon)}} \end{aligned}$$

being $A(D)$ the WEF.

Event-Error Probability

A codeword can diverge and merge again with the all-zero path several times. We want to take into account such situations.



The event-error probability at time unit t is bounded by the first event error probability

$$P(E,t) \leq P_f(E,t)$$

By applying to $P(E,t)$ the same bounds we applied to $P_f(E,t)$

$$P(E) < \sum_{d=d_{\text{free}}}^{\infty} A_d P_d = A(D)|_{D=2\sqrt{\varepsilon(1-\varepsilon)}}.$$

For small ε the bound is dominated by the first term of $A(D)$ such that

$$P(E) \approx A_d (2\sqrt{\varepsilon(1-\varepsilon)})^{d_{\text{free}}} \approx A_{d_{\text{free}}} 2^{d_{\text{free}}} \varepsilon^{\frac{d_{\text{free}}}{2}}.$$

Bit Weight Enumerating Function

Let us introduce the bit conditional weight enumerating function

$$B_w(D) = \sum_d B_{w,d} D^d$$

with $B_{wd} = \frac{w}{k} A_{wd}$ and A_{wd} the number of codewords with weight d and weight of the corresponding information sequence w. Then, B_{wd} is the total number of nonzero information bits associated with codewords of weight d produced by information sequences of weight w divided by the number of information bits k for unit time.

The bit IOWEF is given by

$$\begin{aligned} B(W, D) &= \sum_{w,d} B_{w,d} W^w D^d \\ &= \sum_w B_w(D) W^w \end{aligned}$$

The bit WEF and bit IOWEF

$$B(D) = \sum_d B_d D^d = B(W, D)|_{W=1}$$

with $B_d = \sum_w B_{wd}$.

Relation between the codeword IOWEF $A(D, W)$ **and the bit IOWEF** $B(D)$.

$$\begin{aligned} B(D) &= \sum_d B_d D^d = \sum_{w,d} \frac{w}{k} A_{w,d} D^d \\ &= \frac{1}{k} \left. \frac{\partial(\sum_{w,d} A_{wd} W^w D^d)}{\partial W} \right|_{W=1} = \frac{1}{k} \left. \frac{\partial A(W, D)}{\partial W} \right|_{W=1}. \end{aligned}$$

Bit Error Probability for BSC

The bit error probability is bounded by

$$P_e(E) < \sum_{d_{\text{free}}}^{+\infty} B_d P_d$$

being B_d the total number of information bits of all weight d codewords, divided by the number of information bits k for unit time.
Furthermore,

$$P_e(E) < \sum_{d=d_{\text{free}}} B_d P_d = \sum_{d=d_{\text{free}}} B_d (2\sqrt{\varepsilon(1-\varepsilon)})^d$$
$$= B(D)_{D=2\sqrt{\varepsilon(1-\varepsilon)}} = \frac{1}{k}\frac{\partial}{\partial w} A(W,D)\Big|_{W=1, D=2\sqrt{\varepsilon(1-\varepsilon)}}$$

For small ε, the previous sum is dominated by the first term and

$$P_e(E) \approx B_{d_{\text{free}}} (2\sqrt{\varepsilon(1-\varepsilon)})^{d_{\text{free}}} \approx B_{d_{\text{free}}} 2^{d_{\text{free}}} \varepsilon^{\frac{d_{\text{free}}}{2}}.$$

Asymptotic Coding Gain: Hard Decision Case

For a BSC originated by an AWGN channel

$$\varepsilon = Q\left(\sqrt{\frac{2E_s}{N_0}}\right) \approx \frac{1}{2}\mathrm{e}^{\frac{E_s}{N_0}} \qquad \text{and}$$

$$P_e(E) \approx B_{d_{\text{free}}} 2^{\frac{d_{\text{free}}}{2}} \mathrm{e}^{(\frac{d_{\text{free}}}{2})(E_s/N_0)}$$

Defining the energy per information bit as $E_b = \frac{E_s}{R}$ the bit error probability over a BSC with coding and large $\frac{E_b}{N_0}$

$$P_e(E) \approx B_{d_{\text{free}}} 2^{\frac{d_{\text{free}}}{2}} \mathrm{e}^{-\left(\frac{Rd_{\text{free}}}{2}\right)\left(\frac{E_b}{N_0}\right)}.$$

The bit error probability over a BSC without coding is

$$\widetilde{P}_e(E) = \varepsilon = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \approx \frac{1}{2}\mathrm{e}^{-\frac{E_b}{N_0}}.$$

For large SNR the exponent is dominant. Since $\widetilde{P}_e(E)$ and $P_e(E)$ differ in the factor $\frac{Rd_{\text{free}}}{2}$

$$\gamma = 10 \log_{10} \frac{Rd_{\text{free}}}{2}$$

is called asymptotic coding gain.

Note: The coding gain becomes smaller when $\frac{E_b}{N_0}$ becomes smaller. If $\frac{E_b}{N_0}$ becomes so small that $C < R$ reliable communication is not possible and uncoded systems outperform coded systems.

Event- and Bit-Error Probabilities for DMC

Let us consider a BIAWGN channel with finite Q-ary output quantization (DMC).

The event-error probability is bounded by

$$P(E) < A(D)|_{D=D_0}.$$

and the bit-error probability is bounded by

$$P_e(E) < B(D)|_{D=D_0}.$$

being D_0 the Bhattacharyya parameter

$$D_0 \stackrel{\triangle}{=} \sum_{0 \le j \le Q-1} \sqrt{P(j|0)P(j|1)}.$$

Event- and Bit-Error Probabilities for BIAWGN

$$P(E) < \sum_{d=d_{\text{free}}}^{\infty} A_d Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right)$$

$$P_e(E) < \sum_{d=d_{\text{free}}}^{\infty} B_d Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right)$$

using the bound $Q(x) = \frac{1}{2\pi}\int_x^{+\infty} \exp\left(-\frac{u^2}{2}\right) \mathrm{d}u < \mathrm{e}^{-\frac{x^2}{2}}$

$$P(E) < \sum_{d=d_{\text{free}}}^{\infty} A_d \mathrm{e}^{-\frac{dRE_b}{N_0}} = A(D)\big|_{D=\mathrm{e}^{-\frac{RE_b}{N_0}}}$$

$$P_e(E) < \sum_{d=d_{\text{free}}}^{\infty} B_d \mathrm{e}^{-\frac{dRE_b}{N_0}} = B(D)\big|_{D=\mathrm{e}^{-\frac{RE_b}{N_0}}} = \frac{1}{k}\frac{\partial}{\partial W} A(W,D)\Big|_{W=1, D=\mathrm{e}^{-\frac{RE_b}{N_0}}}$$

For large $\frac{E_b}{N_0}$

$$P_e(E) \approx B_{d_{\text{free}}} \mathrm{e}^{-\frac{d_{\text{free}} RE_b}{N_0}}.$$

The asymptotic coding gain in the soft decision case is given by

$$\gamma = 10 \log_{10} R d_{\text{free}}$$

The asymptotic coding gain in the soft decision case is 3 dB higher than the asymptotic channel gain in the hard decision case.

Maximum Likelihood versus Maximum a Posteriori Probability

Maximum Likelihood (ML): It maximizes the probability $P(\boldsymbol{r}|\widehat{\boldsymbol{v}} = \boldsymbol{v})$. For equally probable codewords it also maximizes $P(\widehat{\boldsymbol{v}} = \boldsymbol{v}|\boldsymbol{r})$, equivalently it minimizes $P(\widehat{\boldsymbol{v}} \neq \boldsymbol{v}|\boldsymbol{r})$, i.e. the word error probability (WER).
Maximum a Posteriori Probability (MAP): It maximizes $P(\widehat{u}_j = u_j|\boldsymbol{r})$, the a posteriori probability (APP), or equivalently it minimizes the BER.

There is a close relation between the ML and the MAP decoding. In fact,

$$\overbrace{P(\widehat{u}_j \neq u_j|\boldsymbol{r})}^{\text{MAP}} = \underbrace{\frac{d(\widehat{\boldsymbol{u}}, \boldsymbol{u})}{K}}_{\text{Encoder}} \underbrace{P(\widehat{\boldsymbol{u}} \neq \boldsymbol{u}|\boldsymbol{r})}_{\text{Code}} = \frac{d(\widehat{\boldsymbol{u}}, \boldsymbol{u})}{K} \overbrace{P(\widehat{\boldsymbol{v}} \neq \boldsymbol{v}|\boldsymbol{r})}^{\text{ML for equiprob. CWs}}$$

While the WER depends only on the code, the BER depends on the code and on the encoder (mapping information sequence to code sequence).
If the encoder is selected such that minimum weight codewords corresponds to minimum weight messages low BER corresponds to low WER.

L-values as Reliability Measures

- **Consider a channel $P(r|v)$ where $v \in \{\pm 1\}$ (BPSK).**
  The *a posteriori L-value* of v conditioned on r is

$$L(v|r) = \ln \frac{P(v=+1|r)}{P(v=-1|r)}.$$

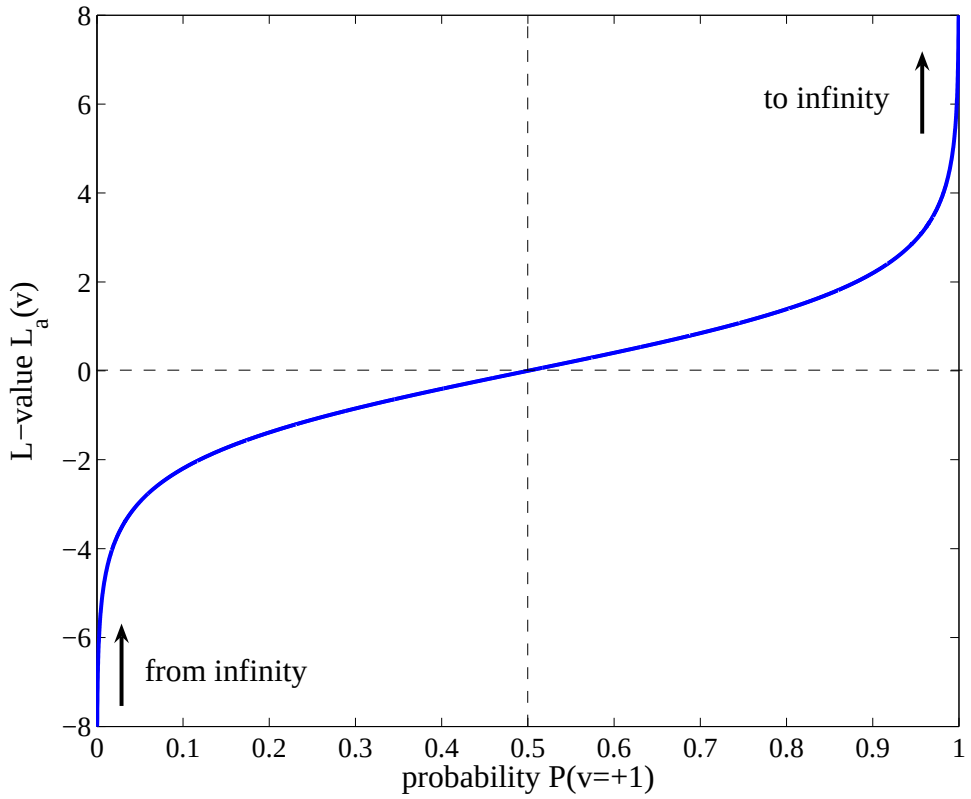
  The L-value is also known as soft value or reliability value.

- **$L(v|r)$ is a function of r but not of v.**

- **The MAP bit estimate is $\widehat{v} = \operatorname{sign}(L(v|r))$.**
  The value $|L(v|r)|$ measures the reliability of the decision.

A Priori Probabilities and L-values

$$L_a(v) = \ln \frac{P(v=+1)}{P(v=-1)} = \ln \frac{P(v=+1)}{1-P(v=+1)} = \ln \frac{1-P(v=-1)}{P(v=-1)}$$



L-values as Reliability Measures (cntd)

- **Applying Bayes' rule yields**

$$P(v=+1|r) = \frac{p(r|v=+1)}{p(r)} P(v=+1)$$

where $P(v=+1)$ **is the** *a priori probability* **that** $v=+1$ **was transmitted.**

- **The definition of the L-values and the Bayes rule yield**

$$L(v|r) = \underbrace{\ln \frac{P(v=+1)}{P(v=-1)}}_{L_a(v)} + \underbrace{\ln \frac{p(r|v=+1)}{p(r|v=-1)}}_{L_c(v|r)}$$

- **The** *a priori* **L-value** $L_a(v)$ **represents the prior knowledge on** v.
- **The** *channel* **L-value** $L_c(v|r)$ **represents the knowledge on** v **given** r.
- **A practical advantage of using L-values instead of probabilities: we can perform additions/subtractions instead of multiplications/divisions.**

L-values and AWGN Channel

- **For an AWGN channel we have** $r = v + z$ **and**

$$P(r|v) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(r-v)^2}{2\sigma^2}\right).$$

- **The definition of the channel L-values and the channel p.d.f. yield**

$$L_c(v|r) = \ln \frac{\exp\left(-\frac{(r-\sqrt{\mathcal{E}})^2}{2\sigma^2}\right)}{\exp\left(-\frac{(r+\sqrt{\mathcal{E}})^2}{2\sigma^2}\right)} = \frac{2\sqrt{\mathcal{E}}}{\sigma^2} r.$$

  The channel L-values are simply weighted versions of the channel observations.

- $L_c(v|r)$ **can be easily included into a metric for decoding.**
- **For** $v \in \{\pm\sqrt{E_s}\}$ **and** $N_0 = 2\sigma^2$, **we have** $L_c = 4\frac{E_s}{N_0} r$.

BCJR Algorithm

It computes the APP L-values $L(u_\ell) = \ln\left(\frac{P(u_\ell=+1|\boldsymbol{r})}{P(u_\ell=-1|\boldsymbol{r})}\right) \qquad \ell = 0, 1, \ldots h-1.$

The decoder output is

$$\widehat{u}_\ell = \begin{cases} +1 & \text{if } L(u_\ell) > 0, \\ -1 & \text{if } L(u_\ell) < 0. \end{cases} \qquad \ell = 0, 1, \ldots h-1.$$

Let U_ℓ^+ (U_ℓ^-) **be the set of all information sequences such that** $u_\ell = +1$ $(u_\ell = -1)$ **and** v **the corresponding codeword. By applying**

$$P(u_\ell|\boldsymbol{r}) = \frac{p(u_\ell = +1, \boldsymbol{r})}{p(\boldsymbol{r})} = \frac{\sum_{\boldsymbol{u}\in U_\ell^+} p(\boldsymbol{r}|\boldsymbol{v})P(\boldsymbol{u})}{p(\boldsymbol{r})}$$

we obtain

$$L(u_\ell) = \ln \frac{\sum_{\boldsymbol{u}\in U_\ell^+} p(\boldsymbol{r}|\boldsymbol{v})P(\boldsymbol{u})}{\sum_{\boldsymbol{u}\in U_\ell^-} p(\boldsymbol{r}|\boldsymbol{v})P(\boldsymbol{u})}$$

Except for very short block lengths h**, a straightforward computation of the APP L-values is prohibitive: Sum over a set of** 2^{h-1} **information sequences!**

For convolutional codes we can make use of the trellis structure.

BCJR Algorithm: Making Use of the Trellis Structure

Let Σ_ℓ^+ (Σ_ℓ^-) be the set of all state pairs $s_\ell = s'$ and $s_{\ell+1} = s$ that correspond to the input bit $u_\ell = +1$ ($u_\ell = -1$) at time ℓ.
Then,

$$P(u_\ell = +1|\boldsymbol{r}) = \frac{p(u_\ell = +1, \boldsymbol{r})}{p(\boldsymbol{r})} = \frac{\sum_{(s',s)\in\Sigma_\ell^+} p(s_\ell = s', s_{\ell+1} = s, \boldsymbol{r})}{p(\boldsymbol{r})}$$

and the APP L-value is given by

$$L(u_\ell) = \ln \frac{\sum_{(s',s)\in\Sigma_\ell^+} p(s_\ell = s', s_{\ell+1} = s, \boldsymbol{r})}{\sum_{(s',s)\in\Sigma_\ell^-} p(s_\ell = s', s_{\ell+1} = s, \boldsymbol{r})}$$

Sum over a set of $2^{\nu_{\mathrm{MAX}}}$ state pairs with $\nu_{\mathrm{MAX}} \ll h$ typically!

BCJR Algorithm: Recursive Structure

Let $\boldsymbol{r}_{t<\ell} = (\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{\ell-1})$ **and** $\boldsymbol{r}_{t>\ell} = (\boldsymbol{r}_{\ell+1}, \boldsymbol{r}_{\ell+2}, \ldots, \boldsymbol{r}_{h+\nu_{\mathrm{MAX}}})$.
Then,

$$\begin{aligned} p(s', s, \boldsymbol{r}) &= p(s', s, \boldsymbol{r}_{t<\ell}, \boldsymbol{r}_\ell, \boldsymbol{r}_{t>\ell}) \\ &= p(\boldsymbol{r}_{t>\ell}|s', s, \boldsymbol{r}_{t<\ell}, \boldsymbol{r}_\ell) p(s', s, \boldsymbol{r}_{t<\ell}, \boldsymbol{r}_\ell) \\ &= p(\boldsymbol{r}_{t>\ell}|s', s, \boldsymbol{r}_{t<\ell}, \boldsymbol{r}_\ell) p(s, \boldsymbol{r}_\ell|s', \boldsymbol{r}_{t<\ell}) p(s', \boldsymbol{r}_{t<\ell}) \\ &= \underbrace{p(\boldsymbol{r}_{t>\ell}|s)}\ \underbrace{p(s, \boldsymbol{r}_\ell|s')}\ \underbrace{p(s', \boldsymbol{r}_{t<\ell})} \\ &= \quad \beta_{\ell+1}(s) \quad \gamma_\ell(s', s) \quad \alpha_\ell(s') \end{aligned}$$

with

$$\begin{aligned} \alpha_\ell(s') &= p(s', \boldsymbol{r}_{t<\ell}) && \textbf{Forward Metric} \\ \beta_{\ell+1}(s) &= p(\boldsymbol{r}_{t>\ell}|s) && \textbf{Backward Metric} \\ \gamma_\ell(s', s) &= p(s, \boldsymbol{r}_\ell|s') && \textbf{Branch Metric} \end{aligned}$$

BCJR Algorithm: Forward Recursion

Let σ_ℓ the set of all states at time ℓ

$$\begin{aligned}\alpha_{\ell+1}(s) = p(s, \boldsymbol{r}_{t<\ell+1}) &= \sum_{s' \in \sigma_\ell} p(s', s, \boldsymbol{r}_{t<\ell+1}) \\ &= \sum_{s' \in \sigma_\ell} p(s, \boldsymbol{r}_\ell | s', \boldsymbol{r}_{t<\ell}) p(s', \boldsymbol{r}_{t<\ell}) \\ &= \sum_{s' \in \sigma_\ell} p(s, \boldsymbol{r}_\ell | s') p(s', \boldsymbol{r}_{t<\ell}) \\ &= \sum_{s' \in \sigma_\ell} \gamma_\ell(s', s) \alpha_\ell(s') \qquad (1)\end{aligned}$$

This recursion to compute the forward metric is called forward recursion. It is initialized by

$$\alpha_0(s) = \begin{cases} 1 & s = \mathbf{0}, \\ 0 & s \neq \mathbf{0}. \end{cases}$$

BCJR Algorithm: Backward Recursion

Similarly, for the backward metric, the backward recursion is given by

$$\beta_\ell(s') = \sum_{s \in \sigma_{\ell+1}} \gamma_\ell(s', s)\beta_{\ell+1}(s) \tag{2}$$

and it is initialized by

$$\beta_K(s) = \begin{cases} 1 & s = \mathbf{0}, \\ 0 & s \neq \mathbf{0}. \end{cases}$$

BCJR Algorithm: Branch Metric

$$\begin{aligned}\gamma_\ell(s',s) &= p(s,\boldsymbol{r}_\ell|s') = \frac{p(s,s',\boldsymbol{r}_\ell)}{p(s')} \\ &= \frac{P(s',s)}{P(s')}\frac{p(s',s,\boldsymbol{r}_\ell)}{P(s',s)} \\ &= P(s|s')p(\boldsymbol{r}_\ell|s',s) \\ &= P(u_\ell)p(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) \end{aligned} \tag{3}$$

where u_ℓ is the input bit and v_ℓ the output bits corresponding to the state transition $s' \to s$ at time ℓ.

Branch Metric for BIAWGN Channel

$$\gamma_\ell(s',s) = P(u_\ell)p(\boldsymbol{r}_\ell|\boldsymbol{v}_\ell) = P(u_\ell)\left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n \mathrm{e}^{-\frac{E_s}{N_0}\|\boldsymbol{r}_\ell-\boldsymbol{v}_\ell\|^2}.$$

The algorithm with forward and backward recursion (1) and (2) and branch metric (3) is called MAP algorithm.

Log-MAP Algorithm or Log-domain BCJR Algorithm

Let

$$\max^*(x,y) = \ln(\mathrm{e}^x + \mathrm{e}^y) = \max(x,y) + \ln(1+\mathrm{e}^{-|x-y|})$$

and

$$\max^*(x,y,z) = \ln(\mathrm{e}^x + \mathrm{e}^y + \mathrm{e}^z) = \max^*(\max^*(x,y),z).$$

The MAP algorithm for an BIAWGN channel is equivalent to the log-MAP algorithm where the APP L-values are given by

$$L(u_\ell) = \max^*_{(s',s)\in\Sigma_\ell^+}(\beta^*_{\ell+1}(s) + \gamma^*_\ell(s',s) + \alpha^*_\ell(s')) - \max^*_{(s',s)\in\Sigma_\ell^-}(\beta^*_{\ell+1}(s) + \gamma^*_\ell(s',s) + \alpha^*_\ell(s')) \quad (4)$$

with

- $$\alpha^*_{\ell+1}(s) = \max^*_{s'\in\sigma_\ell}(\gamma^*_\ell(s',s) + \alpha^*_\ell(s')) \qquad \ell = 0,1,\ldots,K-1 \quad (5)$$

  and

$$\alpha^*_0(s) = \ln\alpha_0(s) = \begin{cases} 0 & s = \mathbf{0}, \\ -\infty & s \neq \mathbf{0}. \end{cases} \quad (6)$$

Log-MAP Algorithm or Log-domain BCJR Algorithm (cntd)

- $$\beta_\ell^*(s') = \max\nolimits_{s\in\sigma_{\ell+1}}^*(\gamma_\ell^*(s',s) + \beta_{\ell+1}^*(s)) \qquad \ell = K-1, K-2, \ldots, 0 \tag{7}$$

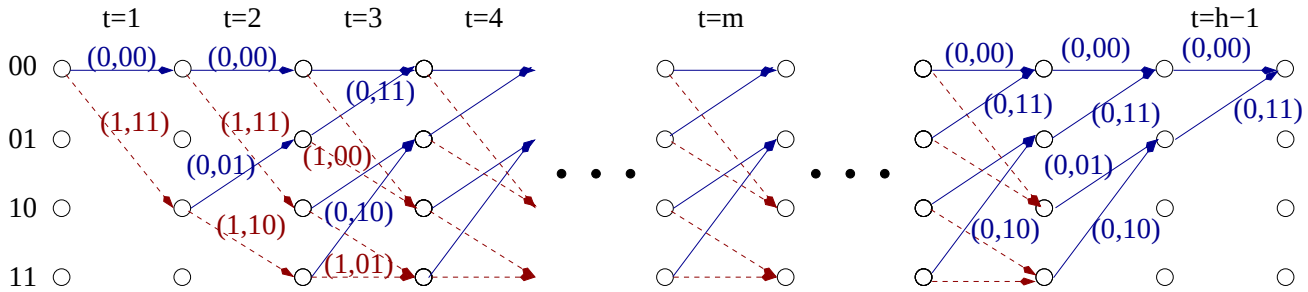
  and

$$\beta_K^*(s) = \ln \beta_K(s) = \begin{cases} 0 & s = \mathbf{0}, \\ -\infty & s \neq \mathbf{0}. \end{cases} \tag{8}$$

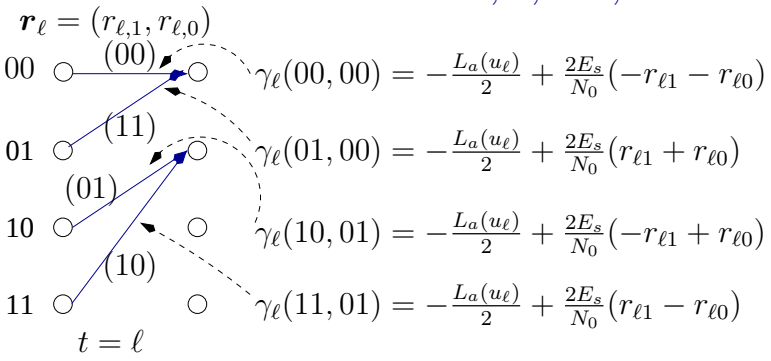
- $$\gamma_\ell^*(s') = \begin{cases} \frac{u_\ell}{2} L_a(u_\ell) + \frac{L_c}{2} \boldsymbol{r}_\ell \boldsymbol{v}_\ell^T & \ell = 0, 1, h-1, \\ \frac{L_c}{2} \boldsymbol{r}_\ell \boldsymbol{v}_\ell^T & \ell = h, h+1, \ldots K \end{cases} \tag{9}$$

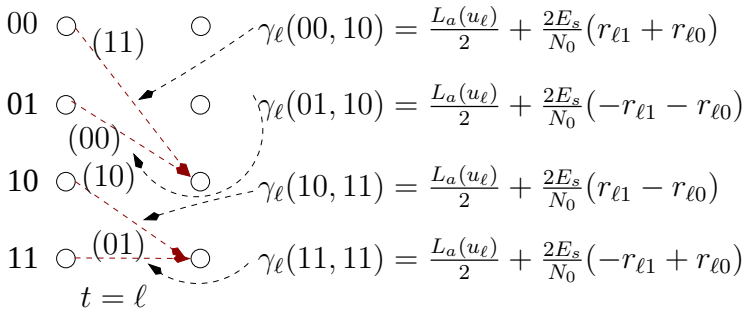
  where $L_a(u_\ell) = \ln\left(\frac{p(u_\ell=+1)}{p(u_\ell=-1)}\right),\ \ell = 0, 1, \ldots, h-1$ **is the a priori L-values of the information bits and** $L_c = \frac{4E_s}{N_0}$ **is the channel reliability factor.**

Example: Computation of the Branch Metrics

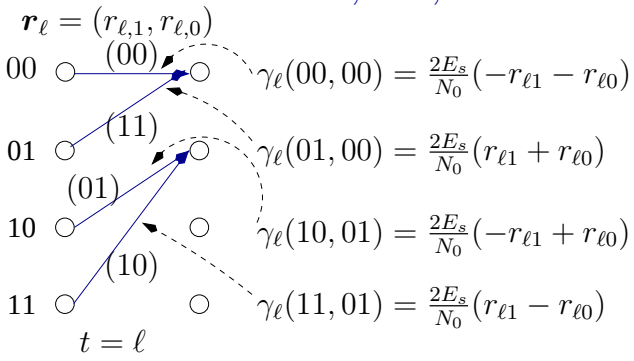


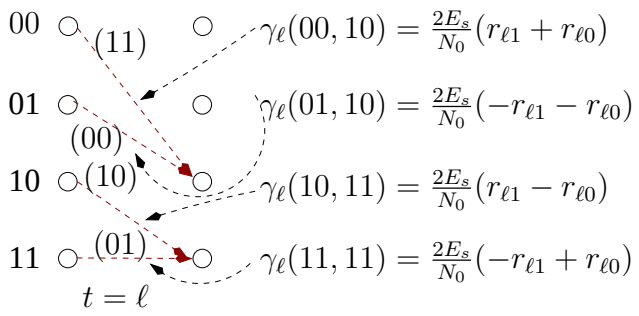
$\ell = 0, 1, \ldots, h-1$



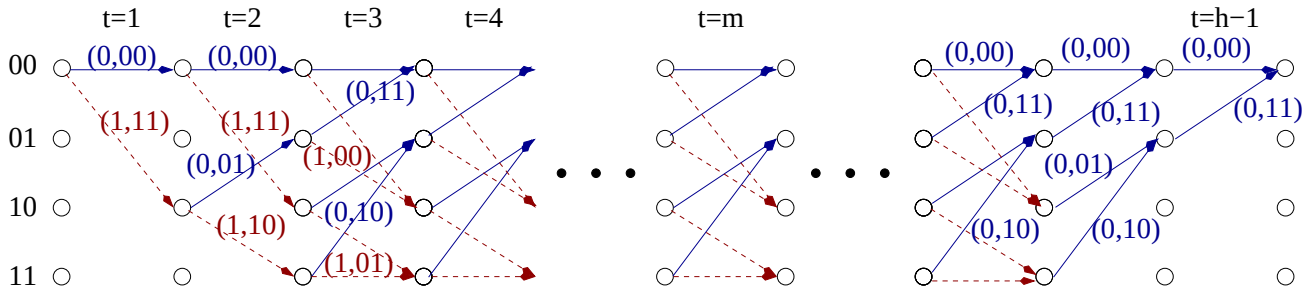


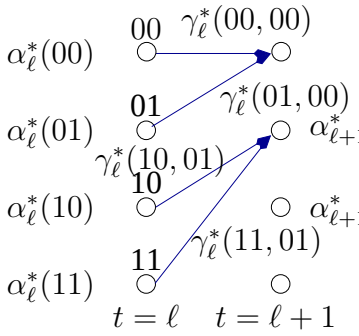
$\ell = h, \ldots, K$





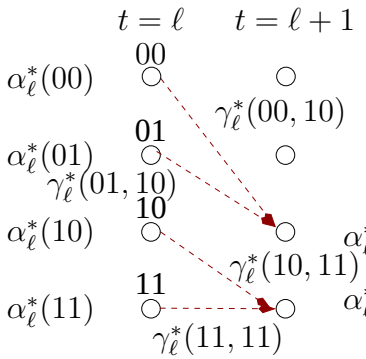
Example: Computation of the Forward Metrics





$$\alpha_{\ell+1}^*(00) = \max(\alpha_\ell^*(00)+\gamma_\ell^*(00,00), \alpha_\ell^*(01)+\gamma_\ell^*(01,00)) + \ln(1+\mathrm{e}^{-|\alpha_\ell^*(00)+\gamma_\ell^*(00,00)-\alpha_\ell^*(01)-\gamma_\ell^*(01,00)|})$$

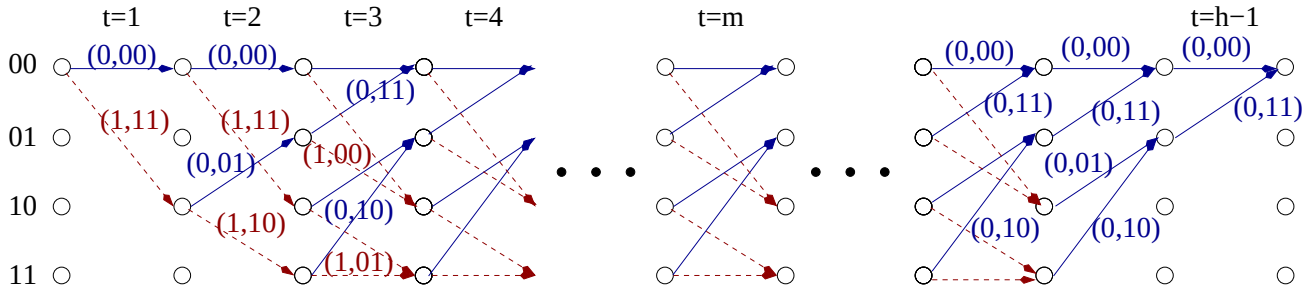
$$\alpha_{\ell+1}^*(01) = \max(\alpha_\ell^*(10)+\gamma_\ell^*(10,01), \alpha_\ell^*(11)+\gamma_\ell^*(11,01)) + \ln(1+\mathrm{e}^{-|\alpha_\ell^*(10)+\gamma_\ell^*(10,01)-\alpha_\ell^*(11)-\gamma_\ell^*(11,01)|})*$$

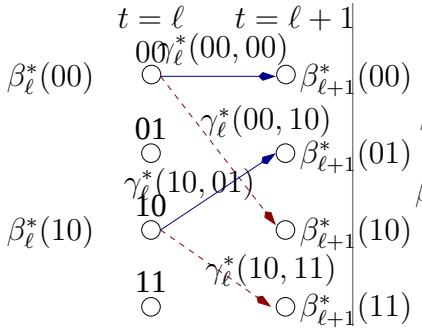


$$\alpha_{\ell+1}^*(10) = \max(\alpha_\ell^*(00)+\gamma_\ell^*(00,10), \alpha_\ell^*(01)+\gamma_\ell^*(01,10)) + \ln(1+\mathrm{e}^{-|\alpha_\ell^*(00)+\gamma_\ell^*(00,10)-\alpha_\ell^*(01)-\gamma_\ell^*(01,10)|})$$

$$\alpha_{\ell+1}^*(11) = \max(\alpha_\ell^*(10)+\gamma_\ell^*(10,11), \alpha_\ell^*(11)+\gamma_\ell^*(11,11)) + \ln(1+\mathrm{e}^{-|\alpha_\ell^*(10)+\gamma_\ell^*(10,11)-\alpha_\ell^*(11)-\gamma_\ell^*(11,11)|})$$

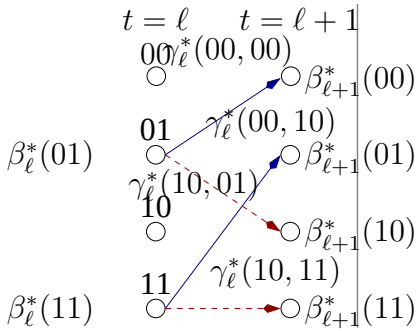
Example: Computation of the Backward Metrics





$$\beta^*_\ell(00) = \max^*(\beta^*_{\ell+1}(00) + \gamma^*_\ell(00,00), \beta^*_{\ell+1}(10) + \gamma^*_\ell(00,10))$$

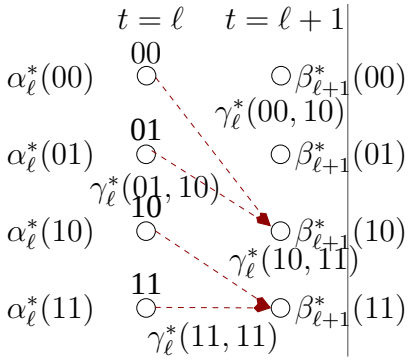
$$\beta^*_\ell(10) = \max^*(\beta^*_{\ell+1}(01) + \gamma^*_\ell(10,01), \beta^*_{\ell+1}(11) + \gamma^*_\ell(10,11))$$



$$\beta^*_\ell(01) = \max^*(\beta^*_{\ell+1}(00) + \gamma^*_\ell(01,00), \beta^*_{\ell+1}(10) + \gamma^*_\ell(01,10))$$

$$\beta^*_\ell(11) = \max^*(\beta^*_{\ell+1}(01) + \gamma^*_\ell(11,01), \beta^*_{\ell+1}(11) + \gamma^*_\ell(11,11))$$

Example: Computation of the APP L-Values

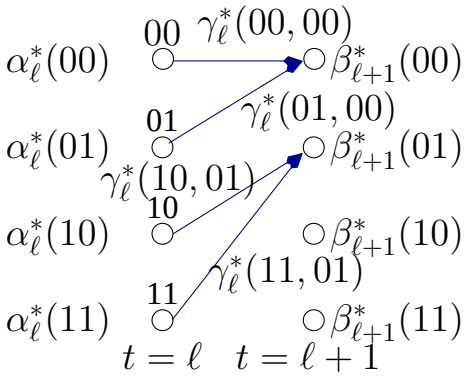


$$y_1^+ = \alpha^*_\ell(00) + \gamma^*_\ell(00, 10) + \beta^*_{\ell+1}(10)$$

$$y_2^+ = \alpha^*_\ell(01) + \gamma^*_\ell(01, 10) + \beta^*_{\ell+1}(10)$$

$$y_3^+ = \alpha^*_\ell(10) + \gamma^*_\ell(10, 11) + \beta^*_{\ell+1}(11)$$

$$y_4^+ = \alpha^*_\ell(11) + \gamma^*_\ell(11, 11) + \beta^*_{\ell+1}(11)$$



$$y_1^- = \alpha^*_\ell(00) + \gamma^*_\ell(00, 00) + \beta^*_{\ell+1}(00)$$

$$y_2^- = \alpha^*_\ell(01) + \gamma^*_\ell(01, 00) + \beta^*_{\ell+1}(00)$$

$$y_3^- = \alpha^*_\ell(10) + \gamma^*_\ell(10, 01) + \beta^*_{\ell+1}(01)$$

$$y_4^- = \alpha^*_\ell(11) + \gamma^*_\ell(11, 01) + \beta^*_{\ell+1}(01)$$

$$L(u_\ell) = \max^*(y_1^+, y_2^+, y_3^+, y_4^+) - \max^*(y_1^-, y_2^-, y_3^-, y_4^-)$$

Log-Domain BCJR Algorithm

Step 1 **Initialize the forward and backward metrics $\alpha_0^*(s)$ and $\beta_K^*(s)$ using (6) and (8).**

Step 2 **Compute the branch metrics $\gamma_\ell^*(s', s),\ \ell = 0, 1, \ldots, K-1,$ using (9).**

Step 3 **Compute the forward metrics $\alpha_{\ell+1}^*(s),\ \ell = 0, 1, \ldots, K-1,$ using (5).**

Step 4 **Compute the backward metrics $\beta_\ell^*(s'),\ \ell = K-1, K-2, \ldots, 0$ using (7).**

Step 5 **Compute the APP L-values $L(u_\ell),\ \ell = 0, 1, \ldots, h-1,$ using (4).**

Step 6 **(Optional) Compute the hard decision $\widehat{u}_\ell = \mathrm{sign}(L(u_\ell)),\ \ell = 0, 1, \ldots, h-1.$**

Remarks

- $$\max^*(x, y) = \ln(\mathrm{e}^x + \mathrm{e}^y) = \max(x, y) + \ln(1 + \mathrm{e}^{-|x-y|})$$

  The function $\max^*(x, y)$ can be implemented simply by a $\max(\cdot, \cdot)$ function plus a look-up table to evaluate $\ln(1 + \mathrm{e}^{-|x-y|})$. This is much simpler and provide greater numerical stability than a direct computation of $\mathrm{e}^x + \mathrm{e}^y$.

  $\Rrightarrow$ The log-domain BCJR algorithm is simpler and more stable than the MAP algorithm.

- **The computation of the branch metrics, the forward metrics, and the backward metrics have almost the same complexity of a Viterbi algorithm.**

  $\Rrightarrow$ The log-domain BCJR algorithm is roughly three times as complex as the Viterbi algorithm.

Remarks 2

- The branch metrics of a Viterbi algorithm depend only on the the correlation $r_\ell v_\ell^T$ and no knowledge about of the channel is needed. In the BCJR algorithm the constant $\frac{L_c}{2}$ captures the effects of the channel and cannot be dropped because it is in relation with L_a, the a priori L-values.
- A simpler algorithm than the log-domain BCJR can be obtained by neglecting the correction term $\ln(1+\mathrm{e}^{-|x-y|})$ in the $\max^*(\cdot,\cdot)$ function and by using the approximation

$$\max^*(x,y) \approx \max(x,y)$$

  This algorithm is called **max-log-MAP algorithm.**

- Because the $\max$ function plays the same role as the COMPARE-SELECT operation in the Viterbi algorithm, the forward recursion in the Max-log-MAP algorithm is equivalent to a forward Viterbi algorithm and the backward recursion in the Max-log-MAP algorithm is equivalent to a backward Viterbi algorithm.

Remarks 3

- The MAP, the log-MAP, and the max-log-MAP algorithms are examples of the so-called forward-backward algorithms.
- If the log-MAP algorithm provides as outputs the (real) APP L-values $L(u_\ell)$ instead of the hard decoder outputs based on $L(u_\ell)$, we say that it provides soft decoder outputs and it is a soft-input soft-output (SISO) decoder.
- The performance loss of the max-log-MAP algorithm compared to the log-MAP algorithm is more pronounced when it is used for iterative decoding since the approximation errors accumulate as additional iterations are performed.

BCJR Algorithm for DMC

- **For DMC it is convenient to use the MAP algorithm.**
- **A straightforward application of the MAP algorithm using the metrics $\alpha_\ell(s), \beta_{\ell+1}(s)$, and $\gamma_\ell(s', s)$ can lead to numerical precision problems.**
  To avoid numerical precision problems it is convenient to use the normalized forward and backward metrics $A_\ell(s)$ and $B_{\ell+1}(s)$ instead of $\alpha_\ell(s)$ and $\beta_{\ell+1}(s)$.
- **The normalization does not affect the calculation of the final APP L-values.**

Normalized Metrics

Let

$$a_\ell \stackrel{\triangle}{=} \sum_{s\in\sigma_\ell} \alpha_\ell(s) \quad \text{for} \quad \ell = 1, \ldots, K \qquad \text{and} \qquad b_\ell \stackrel{\triangle}{=} \sum_{s\in\sigma_\ell} \beta_\ell(s) \quad \text{for} \quad \ell = K-1, K-2 \ldots, 0$$

then

$$A_\ell(s) = \frac{\alpha_\ell(s)}{a_\ell} \quad \text{for} \quad \ell = 1, \ldots, K, \ \forall s \in \sigma_\ell \qquad \text{and} \qquad B_\ell(s) = \frac{\beta_\ell(s')}{b_\ell} \quad \text{for} \quad \ell = K-1, \ldots, 0, \ \forall s' \in \sigma_\ell.$$

References

- S. Lin and D. Costello, **Error Control Coding,** *Prentice Hall,* **1983. Chapter 11 and 12.**

- S. Benedetto and E. Biglieri, **Principles of Digital Transmission with Wireless Applications,** , **Chapter 11 and Appendix F.**

- G. Caire, Lecture notes for the course on Information Theory and Coding, 2003.