Chap. 2 Turbo Codes

☐ **Concatenated codes** : Forney (1966)

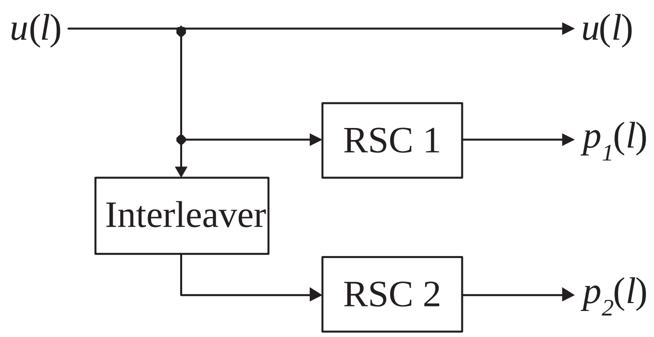
- Serially concatenated code
- Tradeoff between coding gain and complexity
- Example: RS code + convolutional code

☐ **Important issues**

- Performance with optimum decoding
  - Weight distribution is important at low SNR.
- Decoding complexity
  - Performance with practical decoding

□ Turbo codes

- Parallel concatenated convolutional codes (PCCC)
  - two RSC(recursive systematic convolutional) codes
  - random interleaver

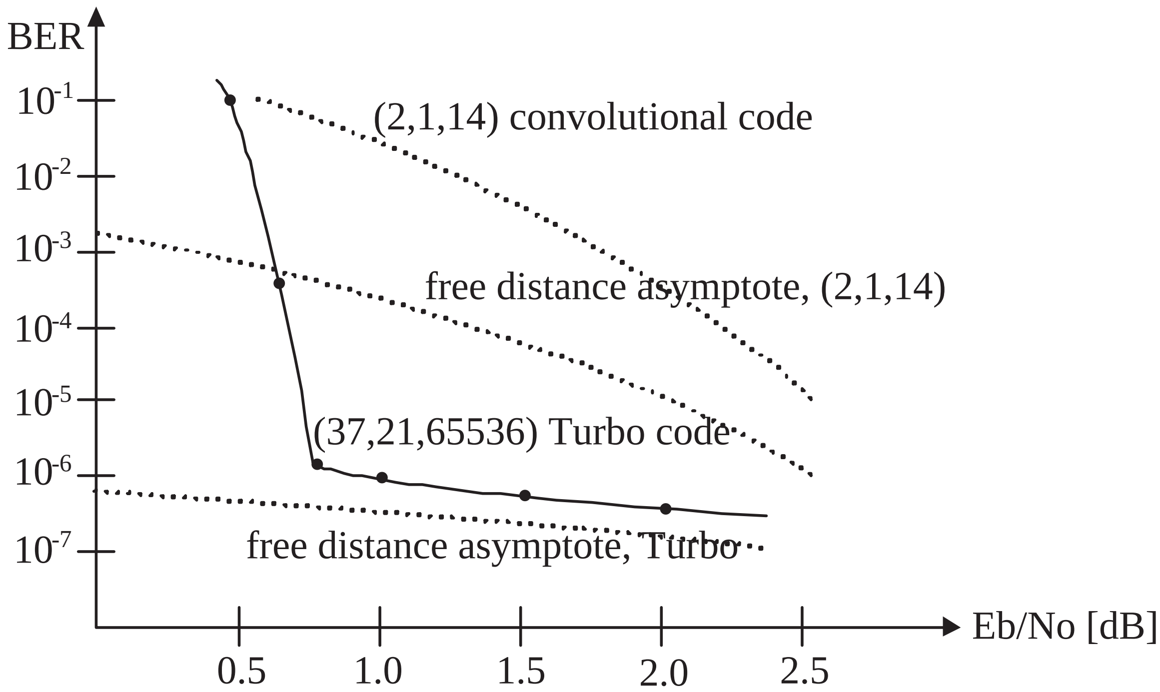


- Turbo codes:
  - G. Berrou, A. Glavieux and P. Thitimajshima (1993)
  - achieve near-capacity performance over the additive white Gaussian noise channel.

Questions:

1) Does the iterative decoding scheme always converge to the optimal solution?
2) Assuming optimal or near-optimal decoding, why do the turbo codes perform so well?

- Simulated performance of a turbo code



Note:

1) Turbo codes achieve BER$= 10^{-5}$ at $E_b/N_o = 0.7$ dB

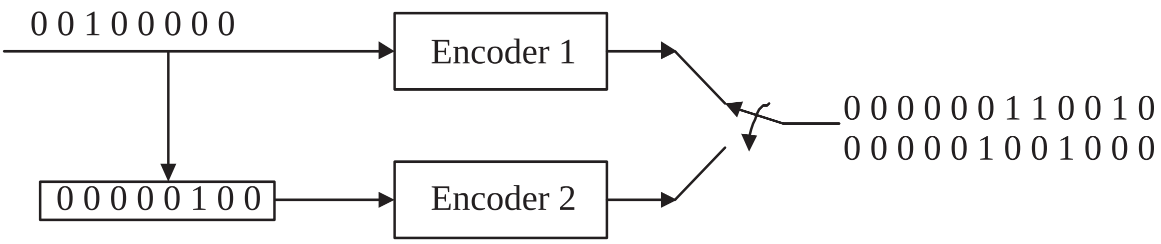
2) Error floor phenomenon: the flattening of the performance curve for moderate to high SNR's

□ Distance properties of Turbo codes

- Linear code:

distance distribution = weight distribution.

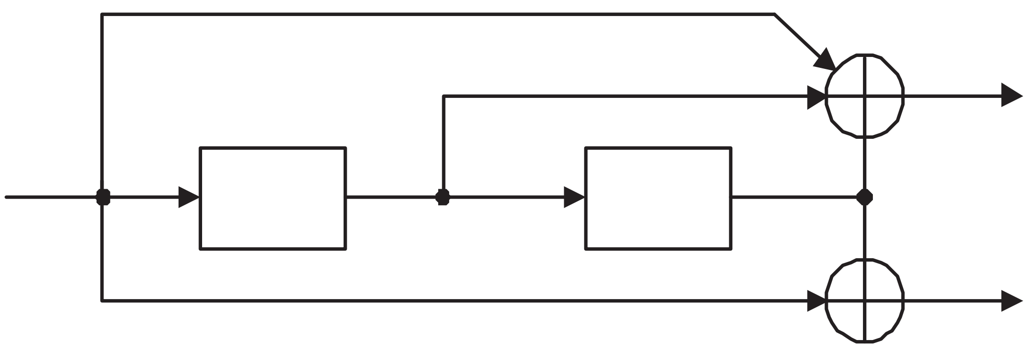
- Consider an input of low weight



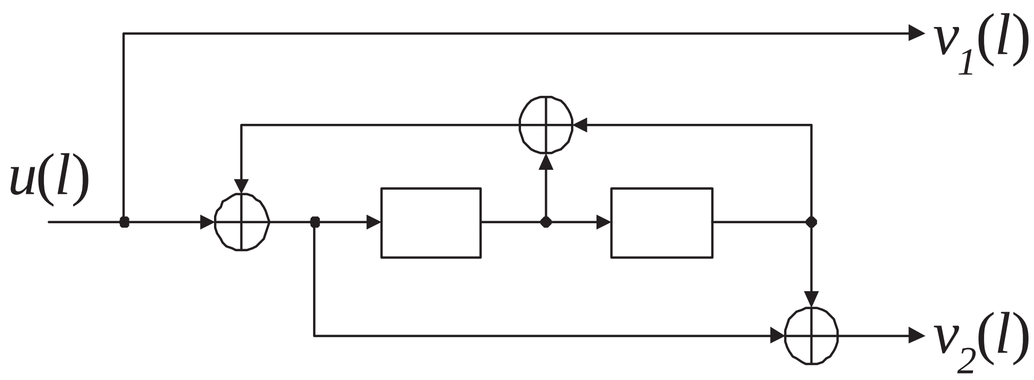
- If low weight input implies low weight output, then the distance properties will be poor.

$\Longrightarrow$ Problem !!

Example



- Solution: Recursive (systematic) encoders



  – Low weight input: high weight output

  – Random interleaver:

    If Encoder 1 produces low wight,
    Encoder 2 produces high weight *with high probability*

- Principle:

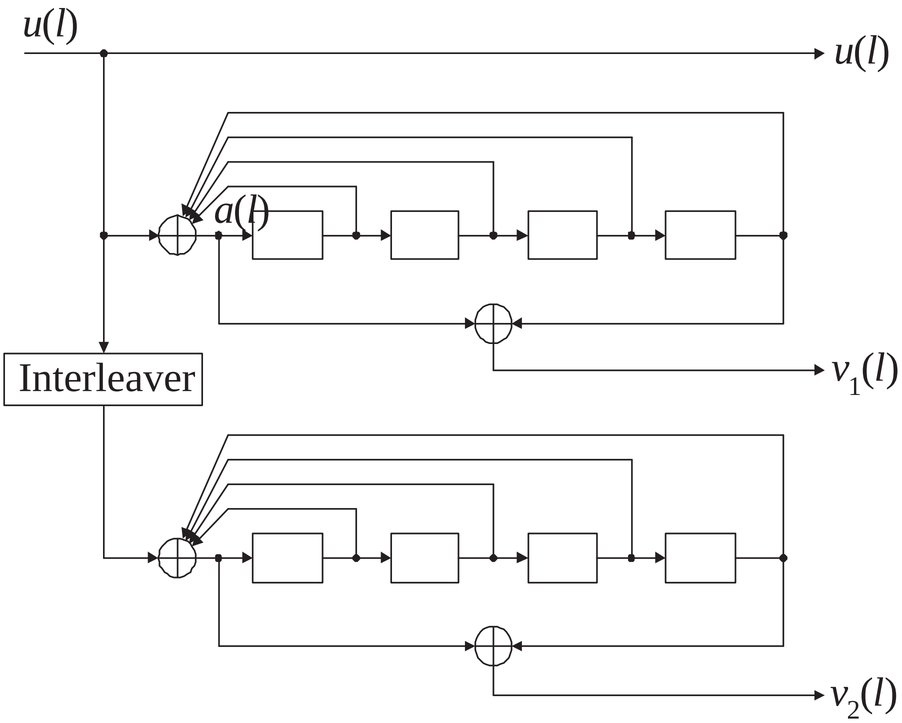
  Instead of aiming for a high minimum distance, *the goal is to design a code with few codewords of low weight.*

Example:

Transfer function matrix:

$$G(D) = \begin{bmatrix} 1 & \dfrac{1+D^4}{1+D+D^2+D^3+D^4} \end{bmatrix}$$

Encoder structure:



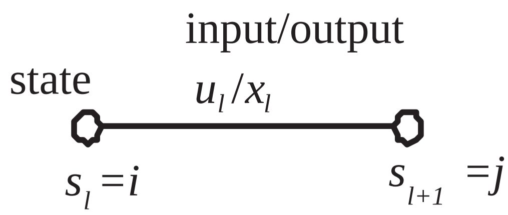
Input-output relation:

$$\begin{aligned} a(l) &= u(l) + a(l-1) + a(l-2) + a(l-3) + a(l-4) \\ v_1(l) &= a(l) + a(l-4) \end{aligned}$$

□

□ Maximum A Posteriori Symbol Decoding (MAP)

- BCJR Algorithm: Bahl, Cocke, Jelinek and Raviv (1974)
- Trellis (for linear block or convolutional codes)
  - Node and branch



  - *Encoding function* : deterministic unless there are parallel transitions.

$$x_l = f(u_l, S_l)$$

- Notation
  - S_l = state at time l
  - $p_{ij} \triangleq \Pr(S_{l+1} = j \,|S_l = i) = \Pr(u_l)$
  - $q_{ij}(x) \triangleq \Pr(f(u_l, S_l) = x \,|S_l = i, S_{l+1} = j)$
  - $\mathbf{x} = (x_1, x_2, \cdots, x_N)$: transmitted sequence
  - $\mathbf{r} = (r_1, r_2, \cdots, r_N)$: received sequence

- The MAP decoder by the BCJR algorithm computes
  - the *a posteriori state probability* $\Pr(S_{l+1} = j \,|\, \mathbf{r})$
  - the *state transition probability* $\Pr(S_l = i, S_{l+1} = j \,|\, \mathbf{r})$

Note: Equivalently, it computes

$$\begin{aligned}
\Pr(S_{l+1} = j, \mathbf{r}) &= \Pr(S_{l+1} = j|\mathbf{r}) \Pr(\mathbf{r}), \\
\Pr(S_l = i, S_{l+1} = j, \mathbf{r}) &= \Pr(S_l = i, S_{l+1} = j \,|\, \mathbf{r}) \Pr(\mathbf{r})
\end{aligned}$$

- Define

$$\begin{aligned}
\alpha_l(j) &\triangleq \Pr(S_{l+1} = j, \mathbf{r}_1^l) \quad \text{where} \quad \mathbf{r}_1^l = (r_1, r_2, \cdots, r_l); \\
\beta_l(j) &\triangleq \Pr(\mathbf{r}_{l+1}^N | S_{l+1} = j) \quad \text{where} \quad \mathbf{r}_{l+1}^N = (r_{l+1}, \cdots, r_N); \\
\gamma_l(j, i) &\triangleq \Pr(S_{l+1} = j, r_l | S_l = i).
\end{aligned}$$

Then

$$\begin{aligned}
\Pr(S_{l+1} = j, \mathbf{r}) &= \Pr(S_{l+1} = j, \mathbf{r}_1^l, \mathbf{r}_{l+1}^N) \\
&= \Pr(S_{l+1} = j, \mathbf{r}_1^l) \underbrace{\Pr(\mathbf{r}_{l+1}^N | S_{l+1} = j, \mathbf{r}_1^l)}_{= \Pr(\mathbf{r}_{l+1}^N | S_{l+1} = j)} \\
&= \alpha_l(j)\, \beta_l(j)
\end{aligned}$$

and

$$\begin{aligned}
\Pr(S_l = i, S_{l+1} = j, \mathbf{r}) &= \Pr(S_l = i, S_{l+1} = j, \mathbf{r}_1^{l-1}, r_l, \mathbf{r}_{l+1}^N) \\
&= \Pr(S_l = i, \mathbf{r}_1^{l-1}) \Pr(S_{l+1} = j, r_l | S_l = i) \\
&\qquad \times \Pr(\mathbf{r}_{l+1}^N | S_{l+1} = j) \\
&= \alpha_{l-1}(i)\, \gamma_l(j, i)\, \beta_l(j)
\end{aligned}$$

- **Update for $\alpha_l(j)$:**

$$
\begin{aligned}
\alpha_l(j) &\triangleq \Pr(S_{l+1} = j, \mathbf{r}_1^l) \\
&= \sum_{\text{state } i} \Pr(S_l = i, S_{l+1} = j, \mathbf{r}_1^l) \\
&= \sum_i \Pr(S_l = i, \mathbf{r}_1^{l-1}) \Pr(S_{l+1} = j, r_l | S_l = i)
\end{aligned}
$$

  – Recursion formula:

$$
\therefore \boxed{\alpha_l(j) = \textstyle\sum_i \alpha_{l-1}(i)\, \gamma_l(j, i)}
$$

  – Initial conditions for $\alpha_l(j)$:

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

  for a trellis code started in the zero state at time $l = 0$

- **Update for $\beta_l(j)$:**

$$\begin{aligned}\beta_l(j) &\triangleq \Pr(\mathbf{r}_{l+1}^N|S_{l+1}=j)\\ &= \sum_i \Pr(S_{l+2}=i,\mathbf{r}_{l+1}^N|S_{l+1}=j)\\ &= \sum_i \Pr(S_{l+2}=i,r_{l+1}|S_{l+1}=j)\ \Pr(\mathbf{r}_{l+2}^N|S_{l+2}=i)\end{aligned}$$

  – Recursion formula:

$$\therefore \quad \boxed{\beta_l(j) = \textstyle\sum_i \beta_{l+1}(i)\,\gamma_{l+1}(i,j)}$$

  – Boundary conditions for $\beta_l(j)$:

$$\beta_N(j) = \begin{cases} 1, & j=0 \\ 0, & j\neq 0 \end{cases}$$

  for a trellis code terminated in the zero state

- **Update for $\gamma_l(j,i)$:**
  - Note that

$$
\begin{aligned}
\gamma_l(j,i) &\triangleq \Pr(S_{l+1}=j, r_l \mid S_l=i) \\
&= \sum_{x_l} \Pr(S_{l+1}=j, x_l, r_l \mid S_l=i) \\
&= \sum_{x_l} \Pr(S_{l+1}=j \mid S_l=i) \Pr(x_l|S_l=i, S_{l+1}=j) \underbrace{\Pr(r_l|x_l)}_{\text{channel}} \\
&= \sum_{x_l} p_{ij}\, q_{ij}(x_l)\, p_n(r_l - x_l)
\end{aligned}
$$

where

$$\Pr(r_l|x_l) = p_n(r_l - x_l)$$

on the AWGN channel.

  - Recursion formula:

$$\boxed{\gamma_l(j,i) \;=\; \sum_{x_l} p_{ij}\, q_{ij}(x_l)\, p_n(r_l - x_l)}$$

over the AWGN channel.

□ MAP Algorithm

1) Initialize

$$\begin{aligned} \alpha_0(0) = 1; \quad \alpha_0(j) = 0 \qquad &\text{for } j \neq 0 \\ \beta_N(0) = 1; \quad \beta_N(j) = 0 \qquad &\text{for } j \neq 0. \end{aligned}$$

Let $l = 1$.

2) Calculate $\gamma_l(j, i)$ and $\alpha_l(j)$ (Forward direction)

3) If $l < N - 1$, let $l = l + 1$ and go to Step 2);
Else $l = N - 1$ and go to Step 4).

4) Calculate $\beta_l(j)$. (Backward (or Reverse) direction)
Using $\gamma_l(j, i)$, $\alpha_l(j)$ and $\beta_l(j)$, calculate

$$\begin{aligned} \Pr(S_{l+1} = j, \mathbf{r}) &= \alpha_l(j)\beta_l(j); \\ \Pr(S_l = i, S_{l+1} = j, \mathbf{r}) &= \alpha_{l-1}(j)\gamma_l(j, i)\beta_l(j). \end{aligned}$$

5) If $l > 0$, let $l = l - 1$ and go to Step 4).

6) Terminate the algorithm and output all the values

$$\Pr(S_{l+1} = j, \mathbf{r}) \text{ and } \Pr(S_l = i, S_{l+1} = j, \mathbf{r}).$$

Note: For an (n, k, ν) convolutional code,

$$\#\ \text{memories} \ \propto \ \underbrace{2^{k\nu}}_{\#\ \text{of states}} \ N$$

where N is the number of information blocks of length k.

Applications

1) A posteriori information symbol probability:

$$\Pr(u_l = u|\mathbf{r}) \ = \ \sum_{(i,j)\in A(u)} \Pr(S_l = i, S_{l+1} = j \,|\, \mathbf{r})$$

where $A(u)$ = the set of all transitions $i \to j$ caused by $u_l = u$.

2) A posteriori probability of the transmitted output symbol x_l

□ Iterative Decoding of Turbo Codes

- The MAP decoder computes

$$\Pr(u_l = u|\mathbf{r}) = \sum_{(i,j)\in A(u)} \Pr(S_l = i, S_{l+1} = j|\mathbf{r}).$$

- Given $\mathbf{r}$, the LLR for u_l can be computed as

$$\begin{aligned} L(\hat{u}_l) &= \log \frac{\Pr(u_l = 0|\mathbf{r})}{\Pr(u_l = 1|\mathbf{r})} \\ &= \log \frac{\sum_{(i,j)\in A(u_l=0)} \alpha_{l-1}(i)\ \gamma_l(j,i)\ \beta_l(j)}{\sum_{(i,j)\in A(u_l=1)} \alpha_{l-1}(i)\ \gamma_l(j,i)\ \beta_l(j)} \end{aligned}$$

Note:

1) A posteriori information symbol probability:

$$\begin{aligned} \Pr(u_l = u|\mathbf{r}) &= \sum_{(i,j)\in A(u)} \Pr(S_l = i, S_{l+1} = j|\mathbf{r}) \\ &= \sum_{(i,j)\in A(u)} \frac{\Pr(S_l = i, S_{l+1} = j, \mathbf{r})}{\Pr(\mathbf{r})} \end{aligned}$$

where

$$\Pr(S_l = i, S_{l+1} = j, \mathbf{r}) = \alpha_{l-1}(i)\gamma_l(j,i)\beta_l(j).$$

2) Determination of $\gamma_l(j,i)$:

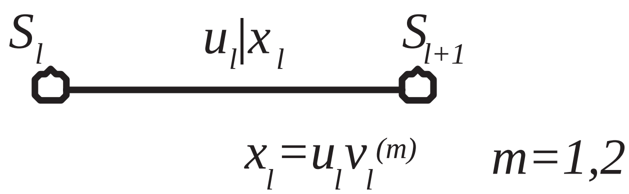
$$
\begin{aligned}
\gamma_l(j,i) &= \Pr(S_{l+1}=j, r_l | S_l = i) \\
&= \sum_{x_l} \Pr(S_{l+1}=j|S_l=i)\, \underbrace{\Pr(x_l|S_l=i, S_{l+1}=j)}_{=\, q_{ij}(x_l)} \Pr(r_l|x_l) \\
&= p_{ij}\ \Pr(r_l^{(0)}, r_l^{(m)} | u_l, v_l^{(m)})
\end{aligned}
$$

where

$$
q_{ij}(x_l) = \begin{cases} 1 & \text{if } x_l = (u_l\ v_l^{(m)}) \\ 0 & \text{otherwise} \end{cases}
$$

and

$$
\begin{cases} u_l & : \ \text{the systematic bit at time } l \\ v_l^{(m)} & : \ \text{the parity bit of the } m\text{th encoder for } m = 1,2 \end{cases}
$$



Note that

$$
\Pr(r_l^{(0)} r_l^{(m)} | u_l v_l^{(m)}) = \Pr(r_l^{(0)}|u_l) \Pr(r_l^{(m)}|v_l^{(m)})
$$

memoryless

and

$$
\begin{aligned}
p_{ij} &= \Pr(S_{l+1}=j|S_l=i) \quad \text{for } (i,j) \in A(u_l = u) \\
&= \Pr(u_l = u).
\end{aligned}
$$

Therefore,

$$\boxed{\gamma_l(j,i) = \Pr(u_l = u) \cdot \Pr(r_l^{(0)} | u_l) \cdot \Pr(r_l^{(m)} | v_l^{(m)})}$$

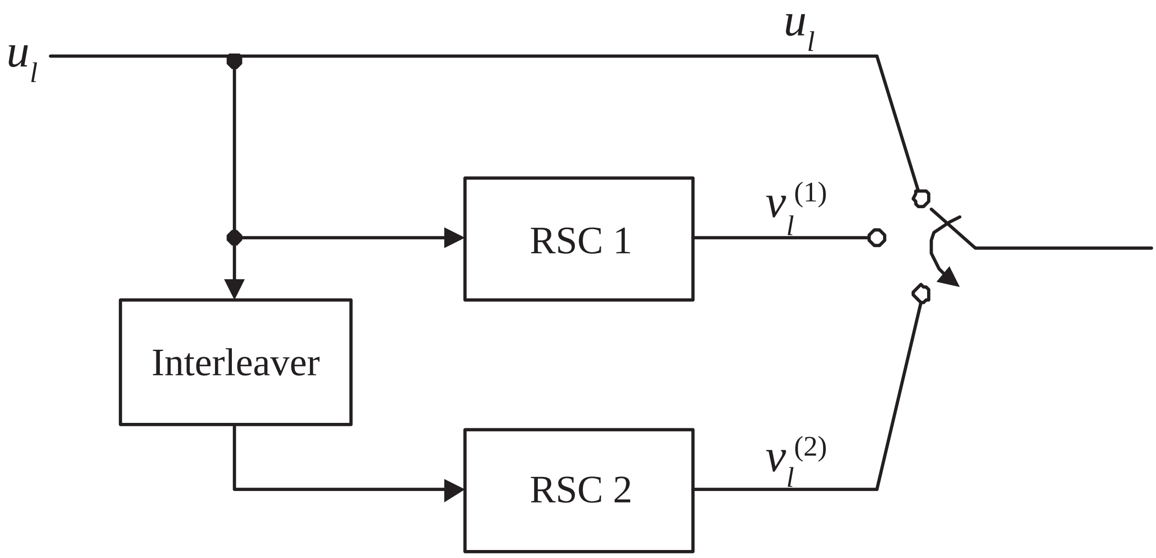
- Using the above expression for $\gamma_l(j,i)$, the LLR $L(\hat{u}_l)$ can be expressed as

$$\begin{aligned} L(\hat{u}_l) &= \log \frac{\sum_{(i,j)\in A(u_l=0)} \Pr(r_l^{(m)}|v_l^{(m)})\alpha_{l-1}(i)\beta_l(j)}{\sum_{(i,j)\in A(u_l=1)} \Pr(r_l^{(m)}|v_l^{(m)})\alpha_{l-1}(i)\beta_l(j)} \\ &\quad + \log \frac{\Pr(u_l=0)}{\Pr(u_l=1)} \\ &\quad + \log \frac{\Pr(r_l^{(0)}|u_l=0)}{\Pr(r_l^{(0)}|u_l=1)} \\ &= L_{e,l}^{(m)} + L(u_l) + L_s(u_l) \end{aligned}$$

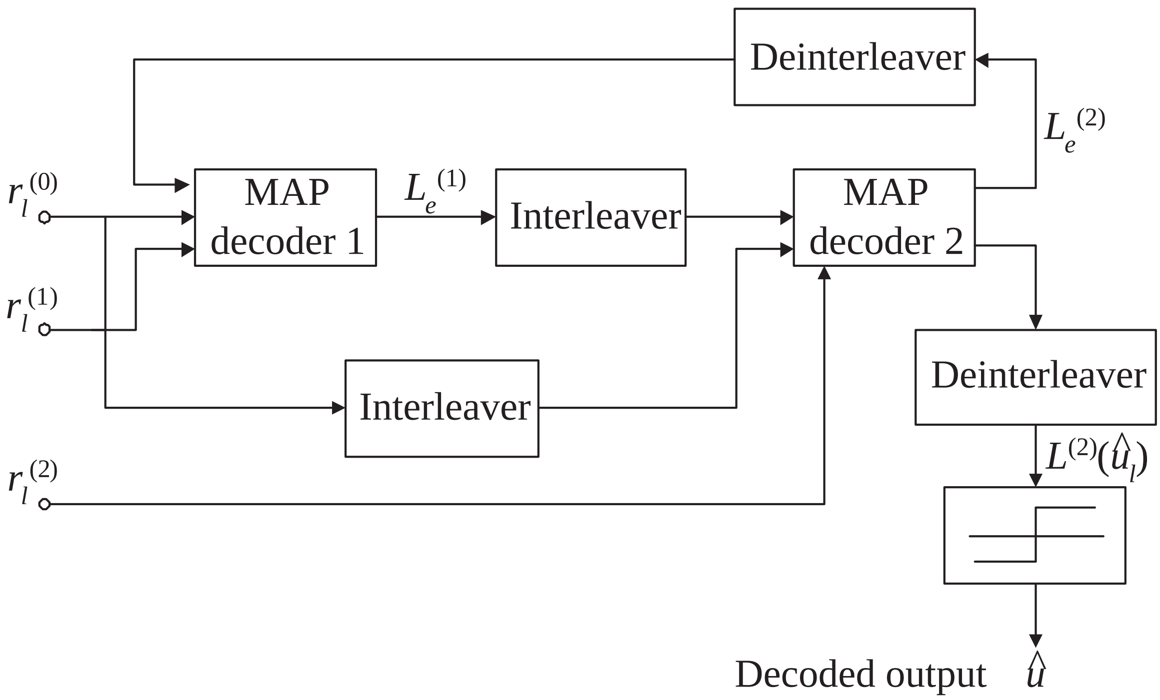
where

$$\begin{aligned} L_{e,l}^{(m)} &= \text{the extrinsic information from the } m\text{th decoder} \\ L(u_l) &= \text{the a priori log-likelihood ratio of the systematic bit } u_l \\ L_s(u_l) &= \text{the log-likelihood of the a posteriori probability} \\ &\quad\ \text{of the systematic bit, given } r_l^{(0)} \\ &= \log \frac{\Pr(u_l=0|r_l^{(0)})}{\Pr(u_l=1|r_l^{(0)})} \\ &= \frac{2}{\sigma^2} r_l^{(0)} \qquad (\textit{channel value}) \end{aligned}$$

- Turbo encoder



- Turbo decoder



Note:

1) First iteration:

$$L(u_l) = 0$$

for all l, since u_l is assumed to be equally likely to be 0 or 1

Decoder 1: Decode $\mathbf{r}^{(0)}, \mathbf{r}^{(1)}$ with $L(u_l) = 0$ for all l

Decoder 2: Decode $\mathbf{r}^{(0)}, \mathbf{r}^{(2)}$ with

$$L(u_l) \longleftarrow L_{e,l}^{(1)}$$

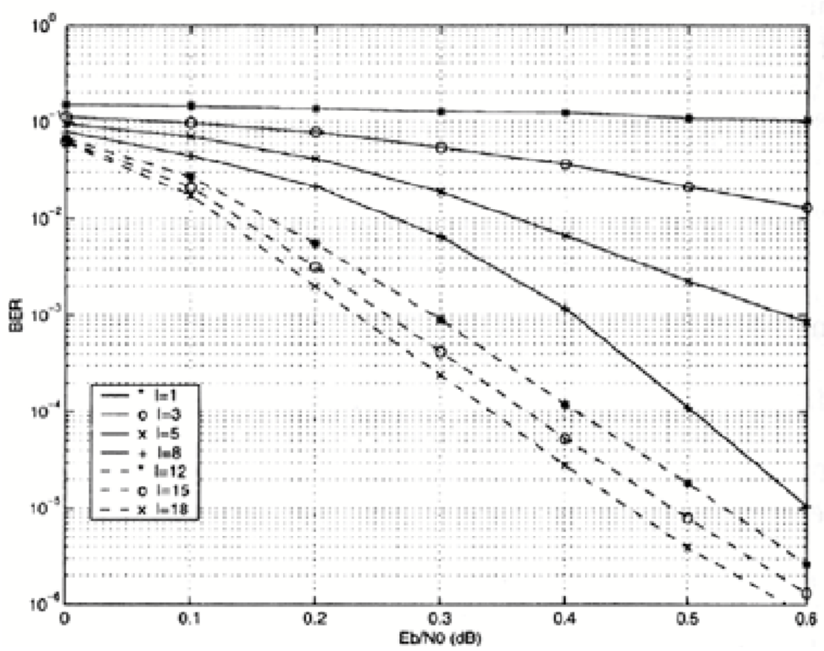
2) If we set $L(u_l) = L^{(1)}(\hat{u}_l)$, then

$$\begin{aligned} L^{(2)}(\hat{u}_l) &= L_{e,l}^{(2)} + L^{(1)}(\hat{u}_l) + L_s(u_l) \\ &= L_{e,l}^{(2)} + \overbrace{L_{e,l}^{(1)} + \underbrace{L(u_l) + L_s(u_l)}_{\text{overemphasized terms}}} + L_s(u_l) \end{aligned}$$

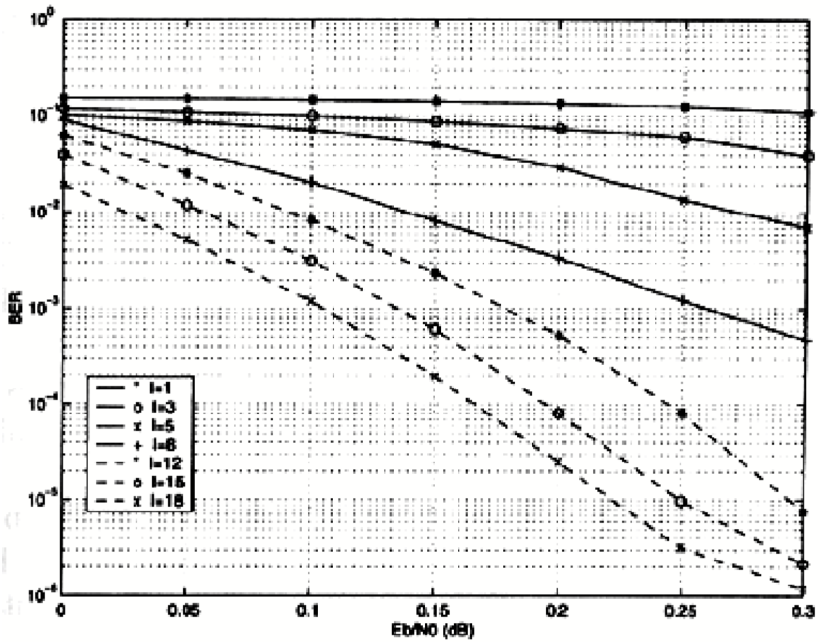
As the turbo code continues to iterate, the log-likelihood ratio accumulates L_s and the systematic bit becomes overemphasized. Therefore,

$$\boxed{L^{(2)}(\hat{u}_l) = L_{e,l}^{(2)} + L_{e,l}^{(1)} + L_s(u_l)}$$

- The BER performance of a 16-state, rate-$1/3$ turbo code with MAP algorithm on an AWGN channel, where $\mathbf{g}_0 = (37), \mathbf{g}_1 = (21)$.
  - the number I of iterations (Fig. 6.3, 6.4)

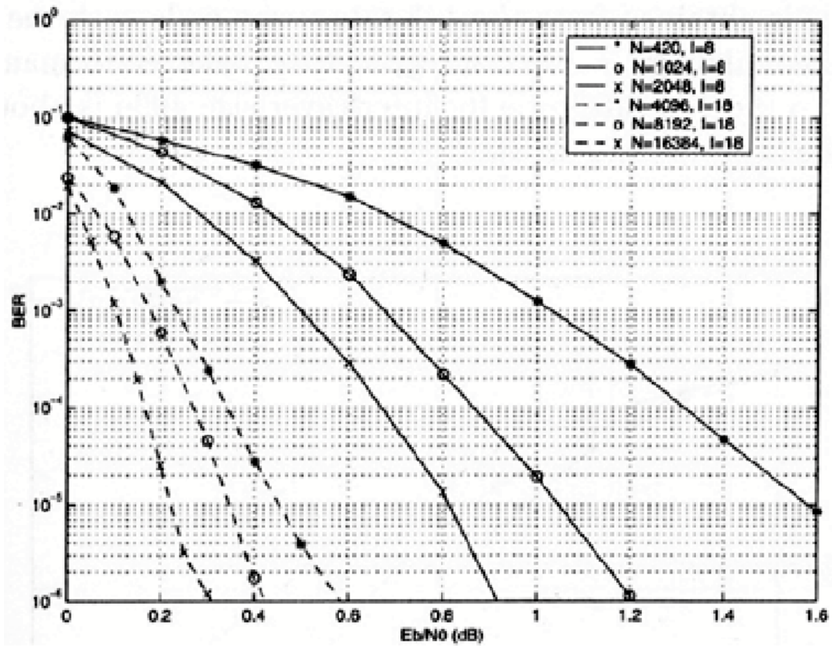


[interleaver size $N = 4,096$ bits]



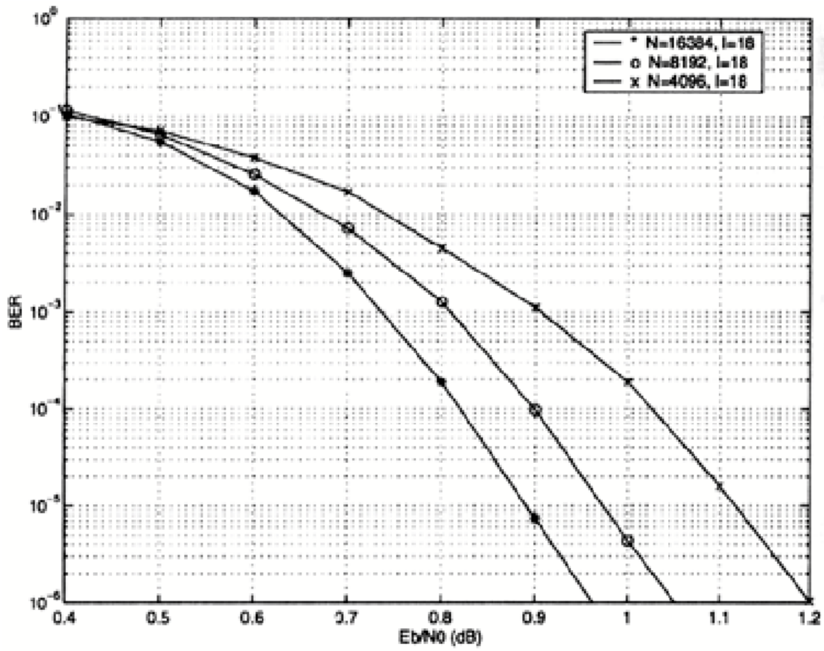
[interleaver size $N = 16,384$ bits]

– interleaver size N (Fig. 6.5)

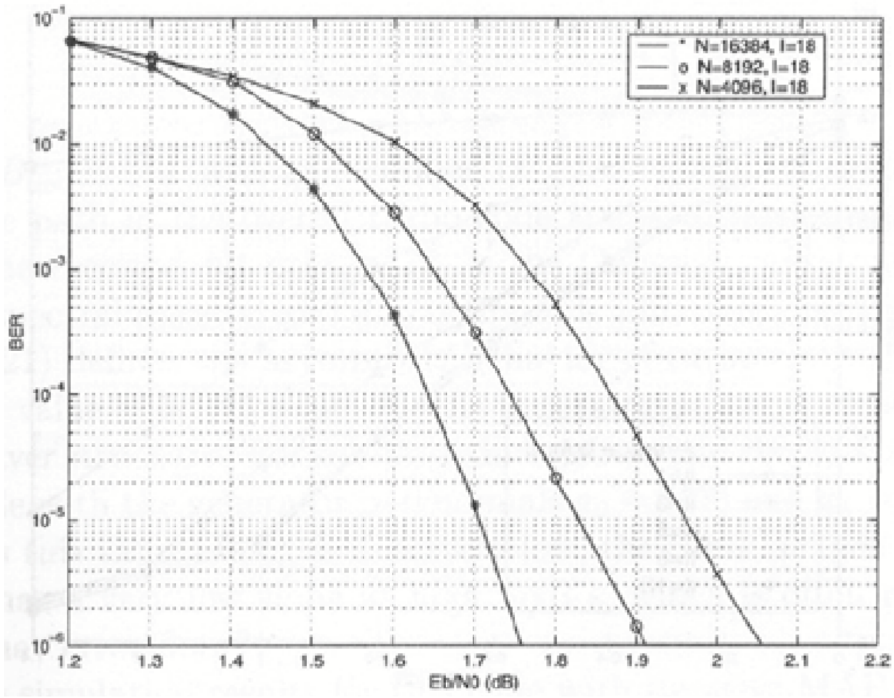


[Number of iterations $I = 8, 18$]

– Puncturing component codes; $\frac{1}{3} \rightarrow \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}$, etc. (Fig. 6.6, 6.7)



[rate $1/2$, interleaver size N, $I = 18$]



[rate $2/3$, interleaver size N, $I = 18$]

(See the book by Vucetic and Yuan)

– Choice of component codes

□ The Log-MAP Algorithm and Max-Log-MAP Algorithm

Define

$$\begin{aligned}
\alpha_l(j) &\longrightarrow & \overline{\alpha}_l(j) &\triangleq \log \alpha_l(j) \\
\beta_l(j) &\longrightarrow & \overline{\beta}_l(j) &\triangleq \log \beta_l(j) \\
\gamma_l(j,i) &\longrightarrow & \overline{\gamma}_l(j,i) &\triangleq \log \gamma_l(j,i).
\end{aligned}$$

Then

$$\alpha_l(j) = \sum_i \alpha_{l-1}(i)\gamma_l(j,i) \rightarrow \overline{\alpha}_l(j) = \log \sum_i e^{\overline{\alpha}_{l-1}(i)+\overline{\gamma}_l(j,i)}$$

$$\alpha_0(j) = \begin{cases} 1, & j = 0 \\ 0, & j \neq 0 \end{cases} \rightarrow \overline{\alpha}_0(j) = \begin{cases} 0, & j = 0 \\ -\infty, & j \neq 0 \end{cases}$$

$$\beta_l(j) = \sum_i \beta_{l+1}(i)\gamma_{l+1}(i,j) \rightarrow \overline{\beta}_l(j) = \log \sum_i e^{\overline{\beta}_{l+1}(i)+\overline{\gamma}_{l+1}(i,j)}$$

$$\beta_N(j) = \begin{cases} 1, & j = 0 \\ 0, & j \neq 0 \end{cases} \rightarrow \overline{\beta}_N(j) = \begin{cases} 0, & j = 0 \\ -\infty, & j \neq 0 \end{cases}$$

Using these expressions, we have

$$L(\hat{u}_l) = \log \frac{\sum_{(i,j)\in A(u_l=0)} \alpha_{l-1}(i)\gamma_l(j,i)\beta_l(j)}{\sum_{(i,j)\in A(u_l=1)} \alpha_{l-1}(i)\gamma_l(j,i)\beta_l(j)}$$

$$\longrightarrow \quad L(\hat{u}_l) = \log \frac{\sum_{(i,j)\in A(u_l=0)} e^{\overline{\alpha}_{l-1}(i)+\overline{\gamma}_l(j,i)+\overline{\beta}_l(j)}}{\sum_{(i,j)\in A(u_l=1)} e^{\overline{\alpha}_{l-1}(i)+\overline{\gamma}_l(j,i)+\overline{\beta}_l(j)}}.$$

Note that

$$\log(e^{\delta_1}+e^{\delta_2}) = \max(\delta_1,\delta_2) + \underbrace{\log(1+e^{-|\delta_2-\delta_1|})}_{=f_c(\delta_2-\delta_1)\ \text{“correction function”}}$$

$$\cong \max(\delta_1,\delta_2) \implies \text{“Max-Log-MAP”}$$

In general,

$$\log(e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_n}) \approx \max_i \delta_i$$

Max-Log MAP :

$$\begin{aligned} L(\hat{u}_l) \approx & \max_{(i,j)\in A(u_l=0)} \left[\overline{\alpha}_{l-1}(i)+\overline{\gamma}_l(j,i)+\overline{\beta}_l(j)\right] \\ & - \max_{(i,j)\in A(u_l=1)} \left[\overline{\alpha}_{l-1}(i)+\overline{\gamma}_l(j,i)+\overline{\beta}_l(j)\right] \end{aligned}$$

Note: Let $\Delta = e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_{n-1}} \triangleq e^{\delta}$. Then

$$\begin{aligned} \log(e^{\delta_1}+e^{\delta_2}+\cdots+e^{\delta_n}) &= \log(\Delta+e^{\delta_n}), \\ &= \max(\log\Delta,\delta_n)+f_c(|\log\Delta-\delta_n|) \\ &= \max(\delta,\delta_n)+f_c(|\delta-\delta_n|) \end{aligned}$$

"Recursion for the Log-MAP"

Comparison of the Decoder Complexity

	MAP	Log-MAP	Max-Log-MAP	SOVA
Additions	$2 \cdot 2^k \cdot 2^\nu + 6$	$6 \cdot 2^k \cdot 2^\nu + 6$	$4 \cdot 2^k \cdot 2^\nu + 8$	$2 \cdot 2^k \cdot 2^\nu + 8$
Multiplications	$5 \cdot 2^k \cdot 2^\nu + 8$	$2^k \cdot 2^\nu$	$2 \cdot 2^k \cdot 2^\nu$	$2^k \cdot 2^\nu$
max ops		$4 \cdot 2^\nu - 2$	$4 \cdot 2^\nu - 2$	$2 \cdot 2^\nu - 1$
look-ups		$4 \cdot 2^\nu - 2$		
exponentials	$2 \cdot 2^k \cdot 2^\nu$			

(n, k) convolutional code of memory order ν

□ Weight Enumerating Function

- An $[n, k]$ linear block code $\mathcal{C}$
  - The code $\mathcal{C}$ is a subspace of $\mathbb{F}_q^n$, where $\mathbb{F}$ is the finite field with q elements.
  - $n =$ code length or block length
  - $k =$ dimension

- Weight enumerator functions for $\mathcal{C}$
  - Minimum distance $= d$
  - Weight distribution: $\{A_0, A_1, \cdots, A_n\}$

    $$A_0 = 1, \quad A_i = 0 \;\; \text{for } 0 < i < d$$

    where $A_i =$ the number of codewords of Hamming weight i

  - Weight enumerator or weight enumerating function (WEF)

    $$A(X) = \sum_{i=0}^{n} A_i X^i$$

- Input-output weight enumerating function (IOWEF)
  - *Input-output weight enumerating function (IOWEF)*

$$A^{IO}(W,X) = \sum_{w,x} A_{w,x} W^w X^x$$

where $A_{w,x}$ = the number of codewords of weight x generated by input information of weight w.

  - *Conditional input-output weight enumerating function*:

The IOWEF can be expressed as

$$A^{IO}(W,X) = \sum_{w} W^w A_w^{IO}(X)$$

where $A_w^{IO}(X)$ is called the *conditional input-output weight enumerating function* defined by

$$A_w^{IO}(X) = \sum_{x} A_{w,x} X^x.$$

The conditional IOWEF $A_w^{IO}(X)$ denotes the weights of the codewords generated by input information of weight w.

The relation between the IOWEF and the conditional IOWEF is

$$A_w^{IO}(X) = \frac{1}{w!} \frac{\partial^w}{\partial W^w} A^{IO}(W,X)\Big|_{W=0}.$$

- Weight enumerator functions for *systematic codes*
  - *Input-redundancy weight enumerating function (IRWEF)*

$$A^{IR}(W,Z) = \sum_{w,z} A_{w,z} W^w Z^z$$

where

$$A_{w,z} = \text{the number of codewords with input information weight } w \text{ and parity check information weight } z$$

  - *Conditional input-redundancy weight enumerating function*:

The IRWEF can be expressed as

$$A^{IR}(W,Z) = \sum_{w} W^w A_w^{IR}(Z)$$

where $A_w^{IR}(Z)$ is called the *conditional input-redundancy weight enumerating function* defined by

$$A_w^{IR}(Z) = \sum_{z} A_{w,z} Z^z.$$

The conditional IRWEF $A_w^{IR}(Z)$ denotes the parity-check information weights of the codewords generated by input information of weight w.

The relation between the IRWEF and the conditional IRWEF is

$$A_w^{IR}(Z) = \frac{1}{w!} \frac{\partial^w}{\partial W^w} A^{IR}(W,Z)\bigg|_{W=0}.$$

□ Performance Bounds

- *Pairwise error probability* (or error event probability):

$$\Pr(\mathbf{c} \to \mathbf{e}) = Q\left(\sqrt{\frac{2dRE_b}{N_o}}\right) \triangleq P_d$$

where

$$\begin{aligned} R &= k/n, \\ E_b &= \text{signal energy per information bit}, \\ N_o &= \text{single sided power spectral density of the AWGN}, \\ d &= d_H(\mathbf{c}, \mathbf{e}). \end{aligned}$$

- *Word error probability* :

$$\begin{aligned} P_W &\le \sum_{d=d_{\min}}^{n} A_d P_d \quad \text{(by the union bound)} \\ &= \sum_{d=d_{\min}}^{n} A_d\, Q\left(\sqrt{2dR\frac{E_b}{N_o}}\right) \\ &\le \sum_{d=d_{\min}}^{n} \frac{1}{2} A_d\, e^{-dR\frac{E_b}{N_o}} \\ &= \frac{1}{2}\left(A(X) - 1\right)\Big|_{X=e^{-R\frac{E_b}{N_o}}} \end{aligned}$$

Note: Upper bound on $Q(x)$

$$Q(x) \le \frac{1}{2}e^{-x^2/2}, \; x \ge 0$$

- *Bit error probability*:

$$\begin{aligned} P_b &= \frac{1}{k}\sum_{\mathbf{e}\neq\mathbf{c}} w_{\mathbf{c}\to\mathbf{e}} \Pr(\hat{\mathbf{c}}=\mathbf{e}) \\ &\leq \frac{1}{k}\sum_{d=1}^{n}\sum_{w=1}^{k} wA_{w,d}P_d \\ &= \sum_{d=d_{\min}}^{n} B_d P_d \\ &= \sum_{d=d_{\min}}^{n} B_d\, Q\left(\sqrt{2dR\frac{E_b}{N_o}}\right) \end{aligned}$$

where

$$\begin{aligned} w_{\mathbf{c}\to\mathbf{e}} &= \text{the Hamming weight of the information part in } \mathbf{c}-\mathbf{e} \\ \Pr(\hat{\mathbf{c}}=\mathbf{e}) &= \text{the probability that the decoder output is } \hat{\mathbf{c}}=\mathbf{e}, \\ &\quad\ \text{given that } \mathbf{c} \text{ was transmitted} \\ B_d &= \text{the } d\text{th } \textit{error coefficient}\text{, i.e., the average of bit errors} \\ &\quad\ \text{caused by transitions between the all-zero codeword} \\ &\quad\ \text{and codewords of weight } d\ (d \geq d_{\min}). \end{aligned}$$

Note that

$$B_d = \sum_{w} \frac{w}{k} A_{w,d}.$$

Therefore, P_b is upper bounded by

$$\begin{aligned} P_b &\le \sum_{d=d_{\min}}^{n} \frac{1}{2} B_d \, e^{-dR\frac{E_b}{N_o}} \\ &= \sum_{w=1}^{k} \frac{w}{2k} W^w A_w(X) \Big|_{W=1,\ X=e^{-R\frac{E_b}{N_o}}} \\ &= \frac{1}{2k} W \frac{\partial A^{IO}(W,X)}{\partial W} \Big|_{W=1,\ X=e^{-R\frac{E_b}{N_o}}}. \end{aligned}$$

Example: Consider the $[7,4]$ systematic Hamming code $\mathcal{C}$ with generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

- The WEF of the code $\mathcal{C}$ is

$$A(X) = 1 + 7X^3 + 7X^4 + X^7.$$

- The IOWEF of the code $\mathcal{C}$ is given by

$$\begin{aligned} A^{IO}(W,X) &= 1 + 3WX^3 + WX^4 + 3W^2X^3 + 3W^2X^4 \\ &\quad + W^3X^3 + 3W^3X^4 + W^4X^7. \end{aligned}$$

- Therefore, the error coefficients of the code $\mathcal{C}$ are determined by

$$\begin{aligned} B_3 &= \frac{1}{4}\times 3+\frac{2}{4}\times 3+\frac{3}{4}\times 1 = 3; \\ B_4 &= \frac{1}{4}\times 1+\frac{2}{4}\times 3+\frac{3}{4}\times 3 = 4; \\ B_7 &= \frac{4}{4}\times 1 = 1 \end{aligned}$$

and

$$B_d = 0 \quad \text{for any positive integer } d \neq 3, 4, 7.$$

- The IRWEF of the code $\mathcal{C}$ is given by

$$\begin{aligned} A^{IR}(W,Z) &= 1+W(3Z^2+Z^3)+W^2(3Z+3Z^2) \\ &\quad +W^3(1+3Z)+W^4Z^3. \end{aligned}$$

- The conditional IRWEF of the code $\mathcal{C}$ is

$$\begin{aligned} A_0^{IR}(Z) &= 1, \\ A_1^{IR}(Z) &= 3Z^2+Z^3, \\ A_2^{IR}(Z) &= 3Z+3Z^2, \\ A_3^{IR}(Z) &= 1+3Z, \\ A_4^{IR}(Z) &= Z^3. \end{aligned}$$

$\square$

☐ Performance Bounds on Convolutional codes

- Assumption
  - Finite information length N
  - (n, k, ν) convolutional code of rate $R = k/n$
  - ML decoding on the AWGN channel

- Bit error rate (or Bit error probability)

$$P_b \leq \sum_{i=1}^{2^N-1} \frac{w_i}{N} Q\left(\sqrt{2d_i R \frac{E_b}{N_o}}\right)$$

where

$$\begin{aligned} w_i &\triangleq \text{information weight of the } i\text{th codeword;} \\ d_i &\triangleq \text{total Hamming weight of the } i\text{th codeword.} \end{aligned}$$

- Average information weight per codeword:

$$\bar{w}_d = \frac{W_d}{N_d}$$

where

$$\begin{aligned} W_d &\triangleq \text{total information weight of all codewords of weight } d \\ &= \sum_{d=w+z} w A_{w,z} \end{aligned}$$

$$N_d \triangleq \text{Number, or multiplicity, of codewords of weight } d$$

- Therefore, P_b is upper bounded by

$$P_b \leq \sum_{d=d_{\text{free}}}^{n(\nu+N)} \frac{N_d\, \bar{w}_d}{N}\, Q\left(\sqrt{2dR\frac{E_b}{N_o}}\right).$$

Note:

1) The *error coefficient* is given by

$$B_d = \frac{N_d \bar{w}_d}{N}.$$

2) N_d includes the codewords due to multiple detours for $d \geq 2\, d_{\text{free}}$

Case $N \to \infty$

- $N_d^0 =$ the number of codewords of weight d caused by the information sequences whose first one occurs at time 0

Then

$$\lim_{N\to\infty} \frac{N_d}{N} = N_d^0 \qquad (\because \text{time-invariant})$$

$$\lim_{N\to\infty} \bar{w}_d = \lim_{N\to\infty} \frac{W_d}{N_d} = \frac{W_d^0}{N_d^0} \triangleq \bar{w}_d^0$$

where

$W_d^0 =$ the total information weight of all codewords with weight d that are caused by information sequences whose first one occurs at time 0.

- Therefore, P_b is upper bounded by

$$\begin{aligned} P_b &\le \sum_{d=d_{\text{free}}}^{n(\nu+N)} N_d^0\, \bar{w}_d^0\, Q\left(\sqrt{2dR\frac{E_b}{N_o}}\right) \\ &= \sum_{d=d_{\text{free}}}^{n(\nu+N)} W_d^0\; Q\left(\sqrt{2dR\frac{E_b}{N_o}}\right). \end{aligned}$$

where W_d^0 corresponds to the error coefficient B_d.

- **Conclusion:**

In order to find good convolutional codes with ML decoders,

  - Maximize the free distance d_{free}.
  - Minimize the number $N^0_{d_{\text{free}}}$ of free-distance paths for a given rate and constraint length.

□ Performance Bounds on Turbo codes

- In general, the bit error probability is given by

$$P_b \le \sum_{i=1}^{2^N-1} \frac{w_i}{N} Q\left(\sqrt{2d_i R \frac{E_b}{N_o}}\right)$$

where w_i is the information weight of the ith codeword.

Note:

1) Convolutional codes:
The encoding function f is linear and time-invariant (LTI).

$$\mathbf{c} = f(\mathbf{u}) \;\Rightarrow\; D\mathbf{c} = f(D\mathbf{u})$$

2) Turbo codes:
f is linear, but time-varying.

$$D\mathbf{c} \ne f(D\mathbf{u}) \quad \textit{with high probability.}$$

- The bit error probability can also be expressed as

$$P_b \le \sum_{d=d_{\text{free}}}^{n(\nu+N)} \frac{N_d\, \bar{w}_d}{N} Q\left(\sqrt{2dR \frac{E_b}{N_o}}\right)$$

where $\bar{w}_d$ is *the average information weight per codeword of weight* d.

- For turbo codes with pseudo-random interleavers,

$$N_d \bar{w}_d \ll N, \quad \text{i.e.,} \quad \frac{N_d \bar{w}_d}{N} \ll 1$$

since the pseudo-random interleavers map low-weight parity sequences in the first constituent encoder to high-weight parity sequences in the second constituent encoder.

Note:

$$\frac{N_d}{N} = \text{effective multiplicity of codewords of weight } d.$$

- For moderate and high SNRs, turbo codes have the asymptotic performance approaching

$$P_b \approx \frac{N_{\text{free}}\, \bar{w}_{\text{free}}}{N} Q\left(\sqrt{2 d_{\text{free}} R \frac{E_b}{N_o}}\right) \triangleq P_{\text{free}}$$

where

N_{free} = the number, or multiplicity, of codewords of weight d_{free};

$\bar{w}_{\text{free}}$ = the average weight of the information sequences causing free-distance codewords;

P_{free} = free-distance asymptote.

Example:

1) Turbo codes with $(37, 21, 65536)+$ pseudorandom interleavers

 – $37: \quad g_0(D) = D^4 + D^3 + D^2 + D + 1$

 – $21: \quad g_1(D) = D^4 + 1$

 – $N_{\text{free}} = 3, \quad d_{\text{free}} = 6, \quad \bar{w}_{\text{free}} = 2$
 Note that each of these paths was caused by an input sequence of weight 2.

 – Therefore,

$$P_{\text{free}} = \frac{3 \cdot 2}{65536} Q\left(\sqrt{2 \cdot 6 \cdot 0.5 \frac{E_b}{N_o}}\right)$$

 where the corresponding effective multiplicity is given by

$$\frac{N_{\text{free}}}{N} = \frac{3}{65536}.$$

2) Maximum free-distance $(2, 1, 14)$ convolutional code

 – $d_{\text{free}} = 18, \quad N^0_{\text{free}} = 18, \quad \bar{W}^0_{\text{free}} = 137 \ (= N^0_{\text{free}} \bar{w}^0_{\text{free}})$

 – Therefore,

$$P_{\text{free}} = 137\ Q\left(\sqrt{2 \cdot 18 \cdot 0.5 \frac{E_b}{N_o}}\right).$$

□

Remark on the performance of turbo codes:

1) *The slope of the asymptote is essentially determined by the free distance of the code.*

2) The *error floor* observed with turbo codes is due to the fact that they have *a relatively small free distance* and consequently a relatively flat free distance asymptote.

3) Increasing the length of the interleaver while preserving the free distance will lower the asymptote without changing its slope by reducing the effective multiplicity.

4) If the size of the interleaver is fixed, then the "error floor" can be modified by increasing the free distance of the code while preserving the error coefficient.
($\Rightarrow$ changing the slope of the free distance asymptote.)

5) For a fixed interleaver size, choosing the feedback polynomial to be a primitive polynomial results in an increased free distance and thus a sleeper asymptote.

□ A Turbo code with a Rectangular Interleaver

- Rectangular interleaver: $120 \times 120 = 14,400 = N$
- $(37, 21, 14400)$ turbo code $(R = 1/2)$
  - $d_{\text{free}} = 12$
  - $N_{\text{free}} = 28,900$
    Each of the free-distance paths is caused by an information sequence of weight 4. $\Rightarrow$ $\bar{w}_{\text{free}} = 4$.
  - Therefore,

$$
\begin{aligned}
P_{\text{free}} &= \frac{28,900 \times 4}{14,400} Q\left(\sqrt{2 \cdot 12 \cdot 0.5 \cdot \frac{E_b}{N_o}}\right) \\
&\approx 8\, Q\left(\sqrt{12\, \frac{E_b}{N_o}}\right)
\end{aligned}
$$

Note: At BER$= 10^{-5}$,

1) $(37, 21, 14400)+$ block interleaver: SNR$\simeq 2.7$dB
2) $(37, 21, 65536)+$ pseudorandom interleaver: SNR$\simeq 0.7$dB
3) $(37, 21, 1024)+$ pseudorandom interleaver: SNR$\simeq 2.5$dB

Note:

Increasing the size of the block interleaver does not result in a significant reduction in the effective multiplicity of the free-distance codewords.

□ Calculation of N_{free} in the Rectangular Interleaver

Assumption

- RSC with transfer function:

$$G(D) = \left[1 \quad \frac{1+D^4}{1+D+D^2+D^3+D^4}\right]$$

- Interleaver size: $N = 120 \times 120$
- Rate $= 1/2$ (with puncturing)

Note that

$$\begin{aligned}(1+D^5)\,G(D) &= \left[1+D^5 \quad 1+D+D^4+D^5\right]\\ (1+D^{10})\,G(D) &= \left[1+D^{10} \quad 1+D+D^4+D^6+D^9+D^{10}\right]\end{aligned}$$

There are *four information patterns* causing codewords with free distance:

1) Pattern A:

$$\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & & & & & 0 \\ 0 & & & & & 0 \\ 0 & & & & & 0 \\ 0 & & & & & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{array} \qquad \begin{array}{lcccccccccccccccc} \text{Inf.}: & 1 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ \mathbf{p}_1: & 1 & \cancel{1} & 0 & \cancel{0} & 1 & \cancel{1} & 0 & \cdots & 0 & 1 & \cancel{1} & 0 & \cancel{0} & 1 & \cancel{1} \\ \mathbf{p}_2: & \cancel{1} & 1 & \cancel{0} & 0 & \cancel{1} & 1 & \cancel{0} & \cdots & 0 & \cancel{1} & 1 & \cancel{0} & 0 & \cancel{1} & 1 \end{array}$$

The weight of the corresponding codewords: $w = 4 + 4 + 4 = 12$.

The number of codewords of weight 12 due to these patterns:

$$(\sqrt{N}-5) \times (\sqrt{N}-5) = 13,225$$

2) Pattern B:

$$\begin{array}{cccccccc}
1&0&0&0&0&1&&\\
0&&&&&0&\big|&\\
\vdots&&&&&\vdots&\big|&9\\
0&&&&&0&\big|&\\
1&0&0&0&0&1&&
\end{array}
\qquad
\begin{array}{l}
\text{Inf.}: \ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ \cdots\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\\
\mathbf{p}_1: \ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ \cdots\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\\
\mathbf{p}_2: \ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1
\end{array}$$

The number of codewords of weight 12 due to these patterns:

$$0.5 \times (\sqrt{N} - 10) \times (\sqrt{N} - 5) = 6,325$$

where the factor 0.5 comes from the puncturing patterns.

3) Pattern C:

$$\begin{array}{ccccccc}
1&0&\cdots&0&1&&\\
0&&&&0&\big|&\\
\vdots&&&&\vdots&\big|&4\\
0&&&&0&\big|&\\
1&\multicolumn{3}{c}{\underbrace{0\ \ \cdots\ \ 0}_{9}}&1&&
\end{array}$$

The number of codewords of weight 12 due to these patterns:

$$0.5 \times (\sqrt{N} - 10) \times (\sqrt{N} - 5) = 6,325$$

where the factor 0.5 comes from the puncturing patterns.

4) Pattern D:

$$\begin{array}{cccccl} 1 & 0 & \cdots & 0 & 1 & \\ 0 & & & & 0 & \multirow{3}{*}{\left.\vphantom{\begin{array}{c}0\\ \vdots\\ 0\end{array}}\right| 9} \\ \vdots & & & & \vdots & \\ 0 & & & & 0 & \\ 1 & 0 & \cdots & 0 & 1 & \\ & \multicolumn{3}{c}{\overline{\quad\ 9\quad\ }} & & \end{array}$$

The number of codewords of weight 12 due to these patterns:

$$0.25 \times (\sqrt{N} - 10) \times (\sqrt{N} - 10) = 3,025$$

where the factor 0.25 comes from the puncturing patterns.

Combining the above results in 1), 2), 3) and 4), N_{free} is given by

$$\begin{aligned} N_{\text{free}} &= 13,225 + 2 \cdot 6,325 + 3,025 \\ &= 28,900. \end{aligned}$$

Note:

1) $N_{\text{free}} \sim N \quad \Rightarrow \quad N_{\text{free}}/N$ does not change significantly even if N increases.

2) The free-distance asymptote and error floor can not be improved significantly in the case of block interleavers, even if N increases.

□ Distance Spectrum of Turbo Codes

- Relatively low free-distance $\Rightarrow$ Error floor occurs.

- Sparse distance spectrum $\Rightarrow$ Outstanding performance of turbo codes (at low SNRs), when a pseudorandom interleaver is used.

- Distance spectrum of $(37, 21, 65536)$ turbo codes (in the average sense):

d	N_d	W_d
6	4.5	9
8	11	22
10	20.5	41
12	75	150

(when only weight-2 information sequences are considered.)

Note that

$$W_d = N_d \bar{w}_d.$$

- Distance spectrum of $(2, 1, 14)$ convolutional codes:

d	N_d^0	W_d^0
18	33	187
20	136	1034
22	835	7857
24	4787	53994
26	27941	361762
28	162513	2374453
30	945570	15452996
32	5523544	99659236

Note: "spectrally dense" and $(N_d \simeq N \times N_d^0)$

The contribution of the higher-distance spectral lines to the overall BER is greater than the contribution of the free-distance term for SNR < 2.7 dB ($\Rightarrow 10^{-6} \leq$ BER).

□ Turbo Codes with Random Interleavers

- Consider the performance of a turbo code, *averaged over all possible pseudorandom interleavers of a given length*.

- Need to calculate the conditional WEF, since

$$P_b \le \sum_{w=1}^{N} \frac{w}{2N} W^w A_w^{IR}(Z)\Big|_{W=Z=e^{-R\frac{E_b}{N_o}}}$$

where N = interleaver length.

- *Uniform interleaver*
  - $N!$ pseudo-random interleavers with uniform distribution (i.e., with probability $\frac{1}{N!}$)
  - An input sequence $\mathbf{u}$ of $w_H(\mathbf{u}) = w$
    $\longrightarrow$ an input sequence $\mathbf{u}'$ of $w_H(\mathbf{u}') = w$
  - The probability that $\mathbf{u}$ is mapped to $\mathbf{u}'$ is given by

$$\Pr(\mathbf{u} \to \mathbf{u}') = \frac{w!(N-w)!}{N!} = \frac{1}{\binom{N}{w}}.$$

- Input-redundancy weight enumerator function (IRWEF):

$$\begin{aligned} A^{IR}(W,Z) &= \sum_w \sum_z A_{w,z} W^w Z^z \\ &= \sum_w W^w \underbrace{\sum_z A_{w,z} Z^z}_{\triangleq A_w(Z)} \end{aligned}$$

- Conditional WEF:

$$A_w(Z) = \frac{A_w^{C_1}(Z)\, A_w^{C_2}(Z)}{\binom{N}{w}}$$

where $A_w^{C_i}(Z)$ = the conditional WEF of ith component code, (averaged over the ensemble of interleavers of length N).

Note that

$$A_w^{C_1}(Z) = A_w^{C_2}(Z) = A_w^{C}(Z).$$

- The conditional WEF $A_w(Z)$ can be approximated as

$$A_w(Z) \;\approx\; \frac{w!}{(n!)^2}\, N^{2n-w}\, [A(w, Z, n)]^2$$

where

n = maximum number of concatenated single error paths within the block length N,

$A(w, Z, n)$ = redundancy weight enumerating function of the component code generated by concatenating n single error paths with total information weight w.

- Bit error probability:

$$P_b \le \sum_{w=w_{\min}}^{N} \frac{w \cdot w!}{2(n!)^2} N^{2n-w-1} W^w \left[A(w,Z,n)\right]^2 \Big|_{W=Z=e^{-R\frac{E_b}{N_o}}}$$

where $w_{\min}$ is the minimum information weight in error paths of the component codes.

- Spectral thinning of the distance spectrum:
  - $\lim_{N\to\infty} A_{w,z} \;=\;$ finite
  - Since each spectral line is a finite sum of $A_{w,z}$ terms, each spectral line converges to a finite value as the interleaver size increases.
  - *Spectral thinning* : convergence of each spectral line to a finite value as $N \;\to\; \infty$.

    $\Rightarrow$ Good performance for low SNR's
    (i.e., achieve near-capacity performance)

□ Primitive polynomials and free distance

Lemma 1 *For an average turbo code, as $N \to \infty$,*

1) The free-distance codewords are caused by information sequences of weight 2.

2) The free-distance of an average turbo code is maximized by constituent encoders that have the largest output-weight for weight-2 information sequences.

Note: For an RSC with transfer function matrix given by

$$G(D) = \begin{bmatrix} 1 & \frac{g_1(D)}{g_0(D)} \end{bmatrix},$$

Lemma 1 tells us that

$g_0(D)$ should be primitive.

Example:

- $(37, 21, 400)$ turbo code:

$$g_0(D) = 1 + D + D^2 + D^3 + D^4 \quad \Rightarrow \quad \text{period} = 5$$

The corresponding turbo code has $d_{\text{free}} = 6$

- $(23, 35, 400)$ turbo code:

$$g_0(D) = 1 + D + D^4 \quad \Rightarrow \quad \text{period} = 15$$

The corresponding turbo code has $d_{\text{free}} = 10$