- Soft decision decoding (or equivalently binary-input additive white Gaussian channel)
  - The error of Ungerboeck codes (at high SNR) is dominated by the "two code words" whose pairwise Euclidean distance is equal to d_{free}. (This d_{free} represents Euclidean distance, not the Hamming distance defined previously.)

$$\Rightarrow \quad \text{Equivalently } x_j = s_{j,m} + w_j \text{ for } j = 1, \ldots, N$$

$$\text{where } \|\boldsymbol{s}_0 - \boldsymbol{s}_1\|^2 = \sum_{j=1}^{N} (s_{j,0} - s_{j,1})^2 = d_{\text{free}}^2$$

$$\Rightarrow \quad \hat{m} = \arg\max \left\{ P\left(\boldsymbol{x} \,|\boldsymbol{s}_0\right), P\left(\boldsymbol{x} \,|\boldsymbol{s}_1\right) \right\}$$

$$\Rightarrow \quad \hat{m} = \arg\max \left\{ \prod_{j=1}^{N} e^{-(x_j - s_{j,0})^2/2\sigma^2}, \prod_{j=1}^{N} e^{-(x_j - s_{j,1})^2/2\sigma^2} \right\}$$

$$\Rightarrow \quad \|\boldsymbol{x} - \boldsymbol{s}_0\|^2 \underset{\boldsymbol{s}_1}{\overset{\boldsymbol{s}_0}{\lessgtr}} \|\boldsymbol{x} - \boldsymbol{s}_1\|^2 \qquad \boldsymbol{x} \begin{cases} \mathcal{N}(\boldsymbol{s}_0, \sigma^2 \mathbb{I}) & \boldsymbol{s}_0 \text{ transmitted} \\ \mathcal{N}(\boldsymbol{s}_1, \sigma^2 \mathbb{I}) & \boldsymbol{s}_1 \text{ transmitted} \end{cases}$$

- (Approximate) Error probability
  - Based on the decision rule $\|\boldsymbol{x} - \boldsymbol{s}_0\|^2 \underset{\boldsymbol{s}_1}{\overset{\boldsymbol{s}_0}{\lessgtr}} \|\boldsymbol{x} - \boldsymbol{s}_1\|^2$

Dominant pairwise error

$$\begin{aligned}
&= P\left(\boldsymbol{s}_0 \text{ transmitted}\right) P\left(\|\boldsymbol{x} - \boldsymbol{s}_0\|^2 > \|\boldsymbol{x} - \boldsymbol{s}_1\|^2 \,|\boldsymbol{s}_0 \text{ transmitted}\right) \\
&\quad + P\left(\boldsymbol{s}_1 \text{ transmitted}\right) P\left(\|\boldsymbol{x} - \boldsymbol{s}_0\|^2 < \|\boldsymbol{x} - \boldsymbol{s}_1\|^2 \,|\boldsymbol{s}_1 \text{ transmitted}\right) \\
&= P\left(\|\boldsymbol{x} - \boldsymbol{s}_0\|^2 > \|\boldsymbol{x} - \boldsymbol{s}_1\|^2 \,|\boldsymbol{s}_0 \text{ transmitted}\right) \\
&= P\left(\|\boldsymbol{w}\|^2 > \|\boldsymbol{w} + \boldsymbol{s}_0 - \boldsymbol{s}_1\|^2\right), \text{ where } \boldsymbol{x} = \boldsymbol{s}_0 + \boldsymbol{w} \\
&= P\left(\langle \boldsymbol{w}, \boldsymbol{s}_0 - \boldsymbol{s}_1 \rangle < -\frac{1}{2}\|\boldsymbol{s}_0 - \boldsymbol{s}_1\|^2\right) \qquad \langle \boldsymbol{w}, \boldsymbol{s}_0 - \boldsymbol{s}_1 \rangle \sim \mathcal{N}(0, \|\boldsymbol{s}_0 - \boldsymbol{s}_1\|^2 \sigma^2) \\
&= \Phi\left(\frac{-\frac{1}{2}\|\boldsymbol{s}_0 - \boldsymbol{s}_1\|^2 - 0}{\|\boldsymbol{s}_0 - \boldsymbol{s}_1\|\sigma}\right) = \Phi\left(-\frac{d_{\text{free}}}{2\sigma}\right) \leq \exp\{-d_{\text{free}}^2/(4N_0)\}
\end{aligned}$$

$$\Phi(-x) = \frac{1}{2}\text{erfc}\left(\frac{x}{\sqrt{2}}\right) \leq \frac{1}{x\sqrt{2\pi}} e^{-x^2/2} \leq e^{-x^2/2} \text{ for } x > 1/\sqrt{2\pi} \qquad \sigma^2 = N_0/2$$

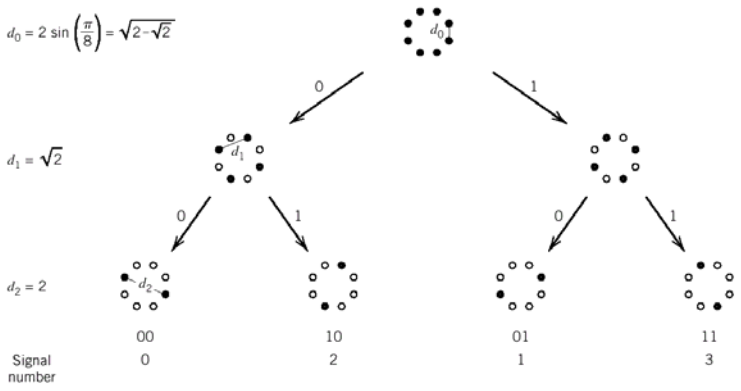
- Asymptotic Coding gain (asymptotic = at high SNR) G_a
  - The performance gain due to coding (i.e., the performance gain of coded system against uncoded system)

$$\text{Uncoded } \exp\left\{-d_{\text{ref}}^2/(4N_0)\right\}$$

$$\text{Coded system } \exp\left\{-d_{\text{free}}^2/(4N_0)\right\}$$

$$G_a = 10\log_{10}\left(\frac{d_{\text{free}}^2}{d_{\text{ref}}^2}\right)$$

    - 4-state Ungerboeck code
      - Its code rate is 2 bits/symbol; hence, it should be compared with uncoded QPSK.



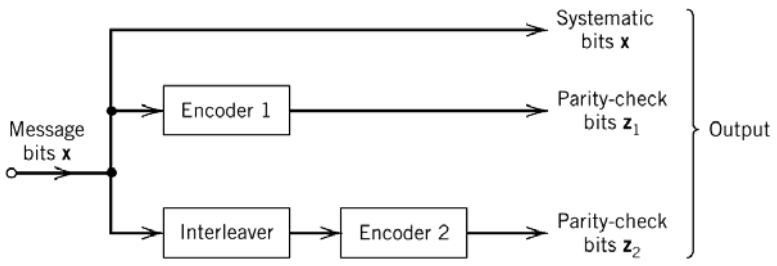
$$d_{\text{free}} = d_2 = 2$$
$$d_{\text{ref}} = d_1 = \sqrt{2}$$
$$\Rightarrow G_a = 10\log_{10}\left(\frac{d_{\text{free}}^2}{d_{\text{ref}}^2}\right) = 3 \text{ dB}$$

Number of states	4	8	16	32	64	128	256	512
Coding gain (dB)	3	3.6	4.1	4.6	4.8	5	5.4	5.7

- Final note
  - Asymptotic coding gain of Ungerboeck codes increases as the number of states grows.
  - To have a 6 dB coding gain, we may need an Ungerboeck code with a large number of states.

10.8 Turbo codes

- Structure of turbo code encoder

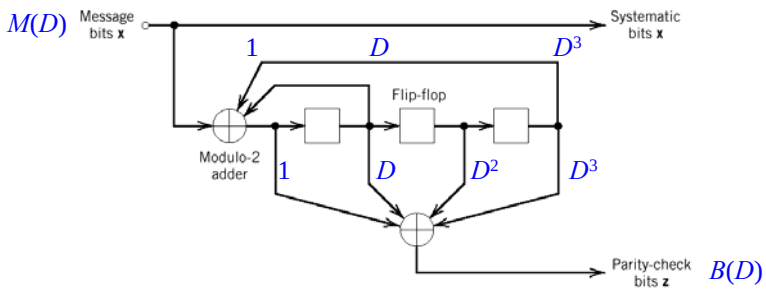


10.8 Turbo codes

- Basic consideration for such design
  - Add an interleaver to tie together distant bits.
  - Use *recursive* systematic convolutional (RSC) codes to make the internal state depend on the past outputs.
  - Use recursive *systematic* convolutional (RSC) codes to make the turbo-like iterative decoding possible.
    - RSC code may suffer *catastrophic error propagation* (one single output error produces an infinite number of parity errors).
  - Use *short constraint-length* RSC codes to reduce to decoding burden in each decoding iteration.

10.8 Turbo codes

- Example 10.8 Eight-state RSC (constituent) encoder



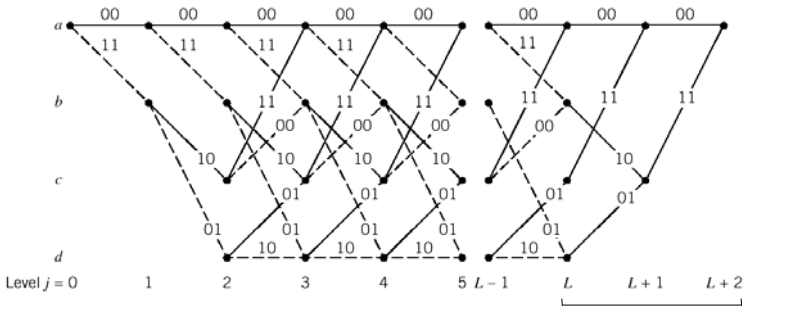
$$g(D) = \left[1, \frac{1 + D + D^2 + D^3}{1 + D + D^3}\right]$$

10.8 Turbo codes

$$\frac{B(D)}{M(D)} = \frac{1 + D + D^2 + D^3}{1 + D + D^3}$$

$$\Rightarrow b_i = m_i + m_{i-1} + m_{i-2} + m_{i-3} - b_{i-1} - b_{i-3}$$

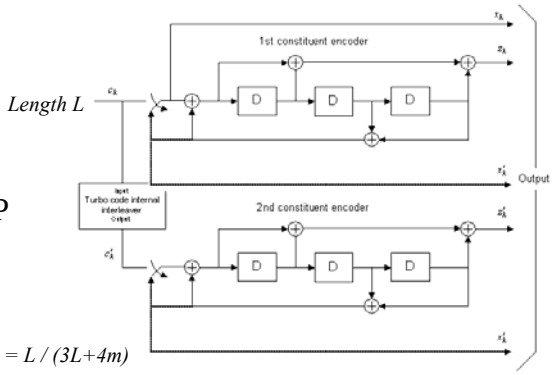
- Remark 1: Zero tail bits
  - With the pseudo-random interleaver, the zero tail bits for the first encoder may not appear to be the tail bits of the second encoder.



Appended two zeros to clear the shift-register contents

- Remark 1: Zero tail bits (continue)
  - With a more careful design, dual clearing of the two encoder register contents can be achieved, which results in considerable performance improvement at medium to high signal-to-noise ratios.

3GPP



Effective code rate = L / (3L+4m)

- Remark 2: Punctured convolution code.
  - Let L be the number of information bits.
  - Each (2, 1) constituent encoder will generate approximately L parity-check bits.
  - With two constituent encoders, the system transmits L information bits and $2L$ parity-check bits, which reduces the code rate to 1/3.

- Remark 2: Punctured convolution code (continue)
  - To improve the code rate, one can "puncture" half of the parity-check bits generated by each constituent encoder.
  - With two constituent encoders, the system transmits L information bits and $L = (L/2) * 2$ parity-check bits, which reduces the code rate to 1/2.
  - At the decoder side, since we exactly know that "no transmission" is performed in those punctured positions, we can directly "nullify" (i.e., make them zero) the corresponding received scalars.
  - For example,

    x_1 and x_2 are information bits.

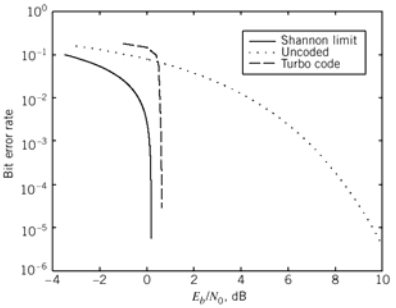
    $$[r_1\ r_2\ r_3\ r_5] = [x_1\ x_2\ x_3\ x_5] + [w_1\ w_2\ w_3\ w_5]$$

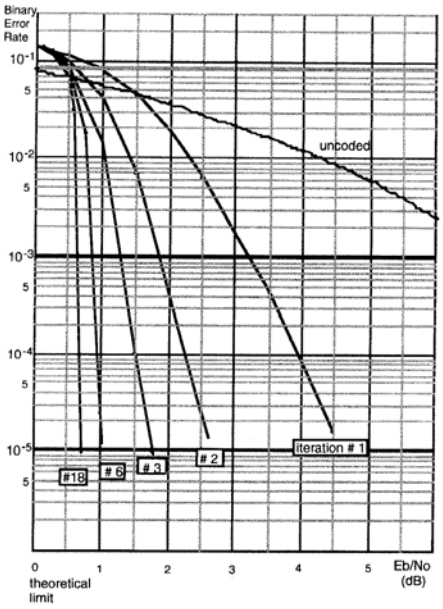
    Parity-check bits x_4 and x_6 are punctured.

    Decoder decodes x_1 and x_2 based on $[r_1\ r_2\ r_3\ 0\ r_5\ 0]$.

10.8 Turbo codes

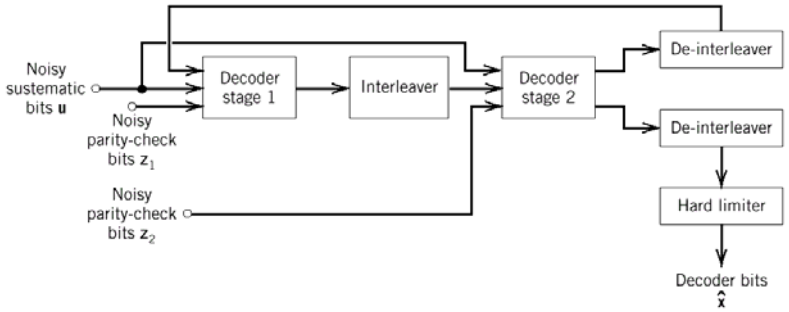
- Performance of Turbo codes





10.8 Turbo codes

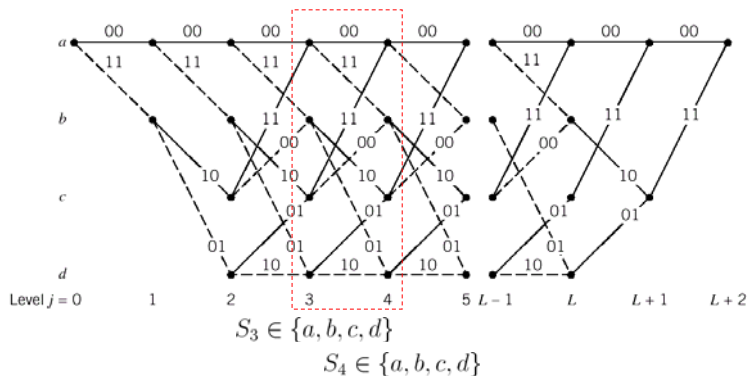
- Turbo decoder



- Turbo component decoder (BCJR algorithm or log-MAP algorithm)
  - Only apply to decode a code whose present state and present output are a function of the past state and current input bit.

Set of transition corresponding to symbol 0 : $\mathcal{B}_{3,4}(0) = \{(a,a),(b,c),(c,a),(d,c)\}$

Set of transition corresponding to symbol 1 : $\mathcal{B}_{3,4}(1) = \{(a,b),(b,d),(c,b),(d,d)\}$



$S_3 \in \{a,b,c,d\}$

$S_4 \in \{a,b,c,d\}$

- It minimizes the bit error directly rather than word error.

$$P(m_4 = 0|\boldsymbol{r}) = P((S_3, S_4) \in \mathcal{B}_{3,4}(0)|\boldsymbol{r})$$

Left-hand side = The probability of the 4th message bit given that the receiver receives ***r***.

Right-hand side = The probability of the encoder going through state S_3 and state S_4 in $B_{3,4}(0)$ given that the receiver receives ***r***.

$$\Rightarrow l(4) = \log \frac{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(1)} P(S_3, S_4|\boldsymbol{r})}{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(0)} P(S_3, S_4|\boldsymbol{r})} = \log \frac{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(1)} P(S_3, S_4, \boldsymbol{r})}{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(0)} P(S_3, S_4, \boldsymbol{r})}$$

My derivation is different from the text, which is based on the original work.

$$\begin{aligned} P(S_3, S_4, \boldsymbol{r}) &= P(S_3, S_4, r_1^6, r_7^8, r_9^N) \\ &= P(r_9^N|S_3, S_4, r_1^6, r_7^8)P(S_3, S_4, r_1^6, r_7^8) \\ &= \underbrace{P(r_9^N|S_4)}_{\beta(S_4)} P(S_3, S_4, r_1^6, r_7^8) \end{aligned}$$

$(S_3, r_1^6, r_7^8) \to S_4 \to r_9^N$ forms a Markov chain

$$\begin{aligned} P(S_3, S_4, r_1^6, r_7^8) &= P(S_3, r_1^6)P(S_4, r_7^8|S_3, r_1^6) \\ &= \underbrace{P(S_3, r_1^6)}_{\alpha(S_3)} \underbrace{P(S_4, r_7^8|S_3)}_{\gamma(S_3,S_4)} \end{aligned}$$

$$\Rightarrow l(4) = \log \frac{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(1)} \alpha(S_3)\beta(S_4)\gamma(S_3, S_4)}{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(0)} \alpha(S_3)\beta(S_4)\gamma(S_3, S_4)}$$

$$\begin{aligned}
\alpha(S_3) &= P(S_3, r_1^6) \\
&= \sum_{S_2 \in \{a,b,c,d\}} P(S_2, S_3, r_1^4, r_5^6) \\
&= \sum_{S_2 \in \{a,b,c,d\}} P(S_2, r_1^4) P(S_3, r_5^6 | S_2, r_1^4) \\
&= \sum_{S_2 \in \{a,b,c,d\}} P(S_2, r_1^4) P(S_3, r_5^6 | S_2) \\
&= \sum_{S_2 \in \{a,b,c,d\}} \alpha(S_2)\gamma(S_2, S_3)
\end{aligned}$$

$$\text{Initial value } \alpha(S_0) = P(S_0, r_1^0) = P(S_0) = \begin{cases} 1, & S_0 = a; \\ 0, & S_0 = b; \\ 0, & S_0 = c; \\ 0, & S_0 = d \end{cases}$$

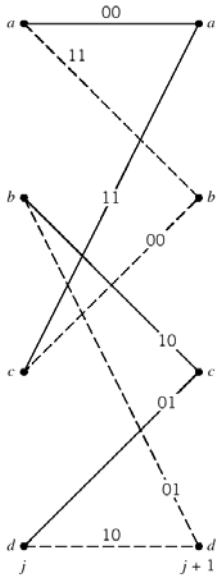
$$\begin{aligned}
\beta(S_4) &= P(r_9^N | S_4) \\
&= \sum_{S_5 \in \{a,b,c,d\}} P(S_5, r_9^{10}, r_{10}^N | S_4) \\
&= \sum_{S_5 \in \{a,b,c,d\}} P(r_{10}^N | S_4, S_5, r_9^{10}) P(S_5, r_9^{10} | S_4) \\
&= \sum_{S_5 \in \{a,b,c,d\}} P(r_{10}^N | S_5) P(S_5, r_9^{10} | S_4) \\
&= \sum_{S_5 \in \{a,b,c,d\}} \beta(S_5)\gamma(S_4, S_5)
\end{aligned}$$

$$\text{Initial value } \beta(S_{L+m}) = \begin{cases} 1, & S_{L+m} = a \\ 0, & S_{L+m} = b \\ 0, & S_{L+m} = c \\ 0, & S_{L+m} = d \end{cases}$$

Note that the turbo codes use *systematic* code; hence, one of the code bit should be the same as the message bit. The example in the next slide is simply taken from the textbook for convenience (so that I don't have to draw a figure myself.)

$$\begin{aligned}
\gamma(S_3, S_4) &= P(S_4, r_7^8 | S_3) \\
&= P(r_7^8 | S_3, S_4) P(S_4 | S_3) \\
&= \underbrace{P(r_7^8 | x_7^8(S_3, S_4))}_{\text{channel}} \underbrace{P(S_4 | S_3)}_{\text{prior}} = \underbrace{P(r_7^8 | m_4, x_8(S_3, S_4))}_{\text{channel}} \underbrace{P(S_4 | S_3)}_{\text{prior}} \\
&= \underbrace{P(r_7 | m_4)}_{\text{systematic}} \underbrace{P(r_8 | x_8(S_3, S_4))}_{\text{parity}} \underbrace{P(S_4 | S_3)}_{\text{prior}}
\end{aligned}$$

$$P(S_4|S_3) = \begin{cases} P(a|a) = P(m_4=0) \\ P(b|a) = P(m_4=1) \\ P(c|a) = 0 \\ P(d|a) = 0 \\ P(a|b) = 0 \\ P(b|b) = 0 \\ P(c|b) = P(m_4=0) \\ P(d|b) = P(m_4=1) \\ P(a|c) = P(m_4=0) \\ P(b|c) = P(m_4-1) \\ P(c|c) = 0 \\ P(d|c) = 0 \\ P(a|d) = 0 \\ P(b|d) = 0 \\ P(c|d) = P(m_4=0) \\ P(d|d) = P(m_4=1) \end{cases}$$



Let $\tilde{\gamma}(S_3,S_4) = \underbrace{P(r_8|x_8(S_3,S_4))}_{\text{parity}}$

$$\Rightarrow l(4) = \log \frac{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(1)} \alpha(S_3)\beta(S_4)\tilde{\gamma}(S_3,S_4)P(S_4|S_3)}{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(0)} \alpha(S_3)\beta(S_4)\tilde{\gamma}(S_3,S_4)P(S_4|S_3)}$$

$$= \log \frac{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(1)} \alpha(S_3)\beta(S_4)\tilde{\gamma}(S_3,S_4)P(m_4=1)P(r_7|1)}{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(0)} \alpha(S_3)\beta(S_4)\tilde{\gamma}(S_3,S_4)P(m_4=0)P(r_7|0)}$$

$$= \underbrace{\log \frac{P(m_4=1)}{P(m_4=0)}}_{\substack{a\ priori\ l_{\text{in}} \\ \text{(intrinsic)}}} + \underbrace{\log \frac{P(r_7|1)}{P(r_7|0)}}_{\text{systematic}} + \underbrace{\log \frac{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(1)} \alpha(S_3)\beta(S_4)\tilde{\gamma}(S_3,S_4)}{\sum_{(S_3,S_4)\in\mathcal{B}_{3,4}(0)} \alpha(S_3)\beta(S_4)\tilde{\gamma}(S_3,S_4)}}_{\text{extrinsic } l_{\text{ex}}}$$

Step 1: With $l_{\text{in}}(m_j)$ known, we can compute

$$P(m_j = 0) = \frac{1}{1 + \exp\{l_{\text{in}}(m_j)\}} \text{ and } P(m_j = 1) = \frac{\exp\{l_{\text{in}}(m_j)\}}{1 + \exp\{l_{\text{in}}(m_j)\}}$$

for each $1 \le j \le L$. We can in turn compute $\gamma(S_j, S_{j+1})$ for all $(S_j, S_{j+1}) \in \{a, b, c, d\}^2$ and for each $0 \le j \le j + m$.

Step 2: With γ available, we can recursively compute α and β in the forward and backward fashion, respectively.

Step 3: With α, β and γ ready, we can compute

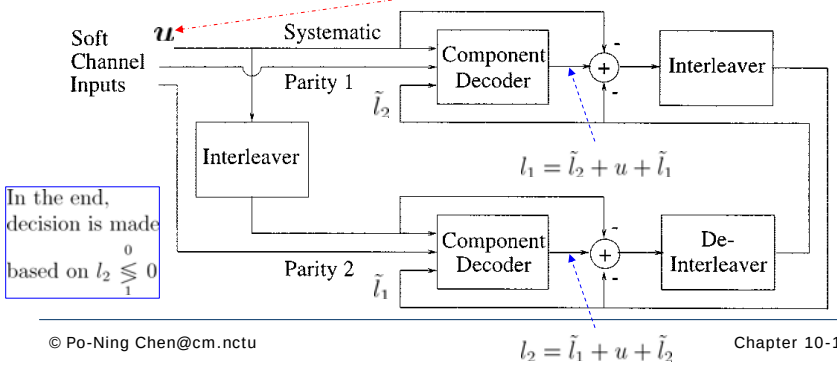
$$l(j) = \log \frac{P(m_j = 1|\boldsymbol{r})}{P(m_j = 0|\boldsymbol{r})} = \log \frac{\sum_{(S_{j-1}, S_j) \in \mathcal{B}_{j-1,j}(1)} \alpha(S_{j-1})\beta(S_j)\gamma(S_{j-1}, S_j)}{\sum_{(S_{j-1}, S_j) \in \mathcal{B}_{j-1,j}(0)} \alpha(S_{j-1})\beta(S_j)\gamma(S_{j-1}, S_j)}$$

Step 4: Calculate $l_{\text{ex}}(j) = l(j) - \log \frac{P(r_{2j-1}|1)}{P(r_{2j-1}|0)} - l_{\text{in}}(j)$.

Observe $l(j) = l_{\text{in}}(j) + \log \frac{P(r_{2j-1}|1)}{P(r_{2j-1}|0)} + l_{\text{ex}}(j)$.

Intuition: Recursion between the two. (The a priori information should come from an "independent source" and not to reuse the same information more than once.)

$$\begin{aligned} u_j &= \log \frac{P(r_{2j-1}|1)}{P(r_{2j-1}|0)} \\ &= \log \frac{\exp\left\{-\frac{(r_{2j-1}-(+1))^2}{2\sigma^2}\right\}}{\exp\left\{-\frac{(r_{2j-1}-(-1))^2}{2\sigma^2}\right\}} \\ &= \frac{2}{\sigma^2} r_{2j-1} \end{aligned}$$



In the end, decision is made based on $l_2 \underset{1}{\overset{0}{\lessgtr}} 0$

- Turbo coding, although quite impressive in performance, is designed based on "good" intuition.
  - For example, Berrou, Glaviexu and Thitimajshima's wrote in their 1993 ICC paper (the first paper for turbo coding technique) that
    - *... for very low SNRs, the BER can sometimes increase during the iterative decoding process. In order to overcome this effect, the extrinsic information $\tilde{l}_1$ (resp. $\tilde{l}_2$) has been divided by* $(1+\theta|l_1|)$ *(resp.* $(1+\theta|l_2|)$*). θ acts as a stability factor and its value of* 0.15 *was adopted after several simulation tests at* $E_b/N_0 = 0.7$ *dB....* [1, pp. 1270]

[1] Claude Berrou and Alain Glavieux, "Near optimal error correcting coding and decoding: Turbo-codes," IEEE Transactions on Communications, vol. 44, no. 10, pp. 1261-1271, October 1996.

10.9 Computer experiment: Turbo coding

- Read by yourself.

10.10 Low-density parity-check codes

- LDPC codes (also known as Gallager codes) are also iteratively decodable.
- Its advantage over turbo coding technique is
  - Absence of low-weight code words.
    - With careless interleaver design, the turbo code may have low weight codewords, which is the main cause for error floor.
  - Iterative decoding of lower complexity.

10.10 Low-density parity-check codes

- In notations, a LDPC code is usually denoted by three tuple (n, t_c, t_r).
  - n = block length
  - t_c = number of 1s in each column (of $(n–k)$ bits)
  - t_r = number of 1s in each row (of n bits)
  - It is not necessary to specify k since

$$(\#\text{ of 1s}) = nt_c = (n-k)t_r \Rightarrow \frac{t_c}{t_r} = 1 - \frac{k}{n}$$

- In terminologies,

$$\mathbf{H}_{(n-k)\times n}\mathbf{G}^T_{n\times k} = \mathbf{0}_{(n-k)\times k}$$

$$\boldsymbol{c}_{1\times n}\mathbf{H}^T_{n\times(n-k)} = \boldsymbol{m}_{1\times k}\mathbf{G}_{k\times n}\mathbf{H}^T_{n\times(n-k)} = \mathbf{0}_{1\times(n-k)}$$

$$\boldsymbol{c}_{1\times n} = \boldsymbol{m}_{1\times k}\mathbf{G}_{k\times n}$$

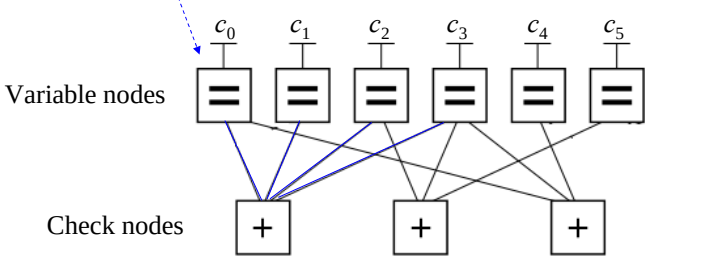
  - Example. $(n, t_c, t_r) = (10, 3, 5)$. $\frac{k}{n} = 1 - \frac{t_c}{t_r} = 1 - \frac{3}{5} = \frac{2}{5}$

$$[c_0, c_1, \ldots, c_9]\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & \ddots & & & & \\ 1 & \ddots & & & & \\ \vdots & & & & & \end{bmatrix}_{10\times(10-4)} = \mathbf{0}_{1\times(10-4)}$$

- Forney's factor graph (Bipartite graph)

$$[c_0, c_1, \ldots, c_5]\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}_{6\times 3} = \mathbf{0}_{1\times 3}$$

Need to solve 6 variables $[c_0, c_1, \ldots, c_5]$



Have $(n-k) = (6-3)$ parity-check equations

10.10 Low-density parity-check codes

- How to find the generator matrix for a given parity-check matrix?
  - Presumption: Systematic LDPC code

$$\boldsymbol{c} = [\boldsymbol{b} : \boldsymbol{m}]$$

$$\text{where} \begin{cases} \boldsymbol{b} \text{ are parity-check bits} \\ \boldsymbol{m} \text{ are message bits} \end{cases}$$

$$\mathbf{H}^T_{(n-k)\times n} = \begin{bmatrix} \mathbf{H}_1 \\ \cdots \\ \mathbf{H}_2 \end{bmatrix} \Rightarrow [\boldsymbol{b} : \boldsymbol{m}] \begin{bmatrix} \mathbf{H}_1 \\ \cdots \\ \mathbf{H}_2 \end{bmatrix} \Rightarrow \boldsymbol{b}\mathbf{H}_1 + \boldsymbol{m}\mathbf{H}_2 = \mathbf{0}$$

The generator matrix of a systematic code must of the shape

$$\mathbf{G} = \left[\mathbf{P}_{k\times(n-k)} \ \vdots \ \mathbf{I}_k\right] \Rightarrow \boldsymbol{m}\mathbf{P} = \boldsymbol{b}$$

This concludes to:

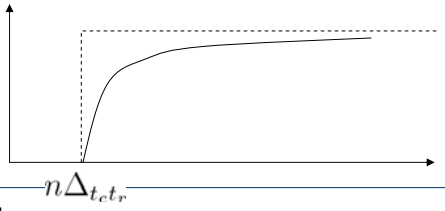
$$\boldsymbol{m}\mathbf{P}\mathbf{H}_1 + \boldsymbol{m}\mathbf{H}_2 = \mathbf{0} \Rightarrow \mathbf{P} = \mathbf{H}_2\mathbf{H}_1^{-1} \Rightarrow \mathbf{G} = \left[\mathbf{H}_2^{-1}\mathbf{H}_1 \ \vdots \ \mathbf{I}_k\right]$$

10.10 Low-density parity-check codes

- Remarks
  - Low-density parity-check code gets its name since the number of 1s in each row and column is small (low-density).
  - If the number of 1s in each row and also in each column is fixed, the LDPC code is said to be *regular*.
  - Under regularity, the inverse matrix of $\mathbf{H}_1$ may be difficult to make to exist.
  - Hence, some "manipulation" or even allowing some "irregularity" is sometimes necessary.

10.10 Low-density parity-check codes

- Minimum distance of LDPC codes
  - By uniformly selecting code word pairs, the pairwise distance becomes a random variable, for which the cumulative distribution function (cdf) can be empirically plotted.
    - It is shown that this cdf can be overbounded by a unit step function as shown below.

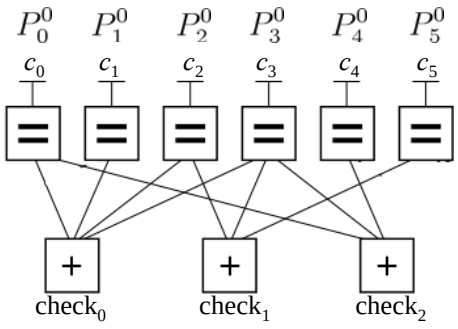


10.10 Low-density parity-check codes

t_c	t_r	Code rate k/n	$\Delta_{t_c t_r}$
5	6	0.167	0.255
4	5	0.2	0.210
3	4	0.25	0.122
4	6	0.333	0.129
3	5	0.4	0.044
3	6	0.5	0.023

10.10 Low-density parity-check codes

- Probabilistic decoding of LDPC codes



P_j^0 pseudo-posterior probability that jth bit $= 0$

p_j^0 prior probability that jth bit $= 0$

Based on

$$\begin{aligned} p_j^0 &= P(c_j = 0|r_j) \\ &= \frac{P(c_j = 0)P(r_j|c_j = 0)}{P(c_j = 0)P(r_j|c_j = 0) + P(c_j = 1)P(r_j|c_j = 1)} \end{aligned}$$

to find the most probable $\boldsymbol{c}$ such that $\boldsymbol{c}\mathbf{H}^T = \mathbf{0}$ through recursion.

$$\begin{bmatrix} Q_{0,0}^0 & Q_{1,0}^0 & Q_{2,0}^0 \\ Q_{0,1}^0 & Q_{1,1}^0 & Q_{2,1}^0 \\ Q_{0,2}^0 & Q_{1,2}^0 & Q_{2,2}^0 \\ Q_{0,3}^0 & Q_{1,3}^0 & Q_{2,3}^0 \\ Q_{0,4}^0 & Q_{1,4}^0 & Q_{2,4}^0 \\ Q_{0,5}^0 & Q_{1,5}^0 & Q_{1,5}^0 \end{bmatrix} \Leftrightarrow \begin{bmatrix} P_{0,0}^0 & P_{1,0}^0 & P_{2,0}^0 \\ P_{0,1}^0 & P_{1,1}^0 & P_{2,1}^0 \\ P_{0,2}^0 & P_{1,2}^0 & P_{2,2}^0 \\ P_{0,3}^0 & P_{1,3}^0 & P_{2,3}^0 \\ P_{0,4}^0 & P_{1,4}^0 & P_{2,4}^0 \\ P_{0,5}^0 & P_{1,5}^0 & P_{1,5}^0 \end{bmatrix}$$

$$[c_0, c_1, \ldots, c_5] \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}_{6\times 3} = \mathbf{0}_{1\times 3}$$

Initialization:

$$P_{i,j}^0 = p_j^0$$

$$\begin{bmatrix} Q_{0,0}^0 & Q_{1,0}^0 & Q_{2,0}^0 \\ Q_{0,1}^0 & Q_{1,1}^0 & Q_{2,1}^0 \\ Q_{0,2}^0 & Q_{1,2}^0 & Q_{2,2}^0 \\ Q_{0,3}^0 & Q_{1,3}^0 & Q_{2,3}^0 \\ Q_{0,4}^0 & Q_{1,4}^0 & Q_{2,4}^0 \\ Q_{0,5}^0 & Q_{1,5}^0 & Q_{1,5}^0 \end{bmatrix} \Leftrightarrow \begin{bmatrix} P_{0,0}^0 & P_{1,0}^0 & P_{2,0}^0 \\ P_{0,1}^0 & P_{1,1}^0 & P_{2,1}^0 \\ P_{0,2}^0 & P_{1,2}^0 & P_{2,2}^0 \\ P_{0,3}^0 & P_{1,3}^0 & P_{2,3}^0 \\ P_{0,4}^0 & P_{1,4}^0 & P_{2,4}^0 \\ P_{0,5}^0 & P_{1,5}^0 & P_{1,5}^0 \end{bmatrix}$$

Horizontal step:

$$Q_{i,j}^0 = \frac{1}{2}\left(1 + \prod_{k\in \text{Check}(i)-\{j\}} (2P_{i,k}^0 - 1)\right)$$

Vertical step:

$$P_{i,j}^0 = \frac{p_j^0 \prod_{k\in \text{Bit}(j)-\{i\}} Q_{k,j}^0}{(1-p_j^0)\prod_{k\in \text{Bit}(j)-\{i\}} (1-Q_{k,j}^0) + p_j^0 \prod_{k\in \text{Bit}(j)-\{i\}} Q_{k,j}^0}$$

Decision step:

$$P_j^0 = \frac{p_j^0 \prod_{k\in \text{Bit}(j)} Q_{k,j}^0}{(1-p_j^0)\prod_{k\in \text{Bit}(j)} (1-Q_{k,j}^0) + p_j^0 \prod_{k\in \text{Bit}(j)} Q_{k,j}^0} \underset{0}{\overset{1}{\lessgtr}} 1 - P_j^0$$

Termination:

If $\hat{\boldsymbol{c}}\mathbf{H}^T = \mathbf{0}$, the algorithm stops;
else go to Horizontal step.

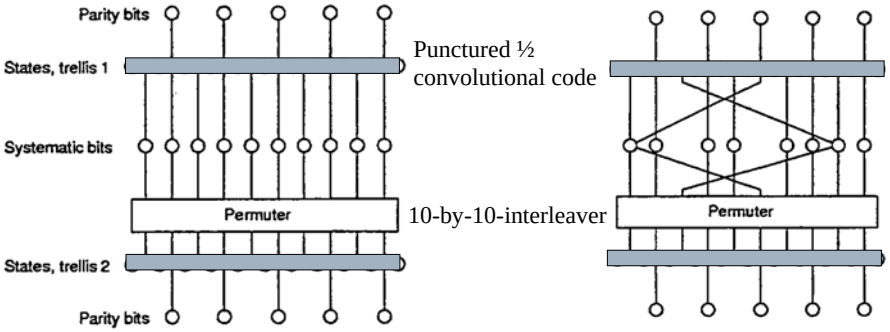
- Final remark
  - Regular LDPC codes do not appear to come as close to Shannon's limit as do their turbo code counterparts.
  - Hence, irregular LDPC codes are most popular due to this reason.

10.11 Irregular codes

- The performance of turbo codes and LDPC codes can be further improved by "irregularity".
  - By "irregularity", we mean that each systematic bit is not used the same number of times.
  - For example, regular turbo code indicates that each systematic bit is used twice in encoding process.

Regular (20, 10) turbo code

Irregular (18, 8) turbo code
(Bits 0 and 6 are used four times, while bits 1, 2, 3, 4, 5, 7 are used only twice.)



10.11 Irregular codes

- Why irregularity gives better performance?
  - The code word is "bit-wisely dependent".

000000
000111
111000
111111

  - If we give a much better estimation on certain positions, e.g., bit 0 and bit 4 in the above example, then the transmitted codeword may be more easily (or earlier-in-iterative-decoding-viewpoint) identified.

10.11 Irregular codes