# MAP decoding: The BCJR algorithm

- Maximum a posteriori probability (MAP) decoding
  - Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm (1974)
  - Baum-Welch algorithm (1963?)*
- Decoder inputs:
  - Received sequence **r** (soft or hard)
  - A priori $L$-values $L_a(u_l) = \ln(P(u_l = 1)/P(u_l = -1))$
- Decoder outputs:
  - A posteriori probability (APP) $L$-values $L(u_l) = \ln(P(u_l = 1|\mathbf{r})/P(u_l = -1|\mathbf{r}))$
    - $> 0$: $u_l$ is most likely to be 1
    - $< 0$: $u_l$ is most likely to be -1

# BCJR (cont.)

$$L(u_l) \equiv \ln\left[\frac{P(u_l = +1|\mathbf{r})}{P(u_l = -1|\mathbf{r})}\right]$$

$$P(u_l = +1|\mathbf{r}) = \frac{p(u_l = +1, \mathbf{r})}{P(\mathbf{r})} = \frac{\sum_{\mathbf{u}\in\mathbf{U}_l^+} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}{\sum_{\mathbf{u}} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}$$

$$L(u_l) = \ln\left[\frac{\sum_{\mathbf{u}\in\mathbf{U}_l^+} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}{\sum_{\mathbf{u}\in\mathbf{U}_l^-} p(\mathbf{r}|\mathbf{v})P(\mathbf{u})}\right],$$

$$P(u_l = +1|\mathbf{r}) = \frac{p(u_l = +1, \mathbf{r})}{P(\mathbf{r})} = \frac{\sum_{(s',s)\in\Sigma_l^+} p(s_l = s', s_{l+1} = s, \mathbf{r})}{P(\mathbf{r})}$$

# BCJR (cont.)

$$L(u_l) = \ln \left\{ \frac{\sum_{(s',s)\in\Sigma_l^+} p(s_l = s', s_{l+1} = s, \mathbf{r})}{\sum_{(s',s)\in\Sigma_l^-} p(s_l = s', s_{l+1} = s, \mathbf{r})} \right\}$$

$$p(s', s, \mathbf{r}) = p(s', s, \mathbf{r}_{t<l}, \mathbf{r}_l, \mathbf{r}_{t>l}),$$

$$\begin{aligned} p(s', s, \mathbf{r}) &= p(\mathbf{r}_{t>l} | s', s, \mathbf{r}_{t<l}, \mathbf{r}_l) p(s', s, \mathbf{r}_{t<l}, \mathbf{r}_l) \\ &= p(\mathbf{r}_{t>l} | s', s, \mathbf{r}_{t<l}, \mathbf{r}_l) p(s, \mathbf{r}_l | s', \mathbf{r}_{t<l}) p(s', \mathbf{r}_{t<l}) \\ &= p(\mathbf{r}_{t>l} | s) p(s, \mathbf{r}_l | s') p(s', \mathbf{r}_{t<l}), \end{aligned}$$

# BCJR (cont.)

$$\alpha_l(s') \equiv p(s', \mathbf{r}_{t<l})$$

$$\gamma_l(s', s) \equiv p(s, \mathbf{r}_l | s')$$

$$\beta_{l+1}(s) \equiv p(\mathbf{r}_{t>l} | s),$$

$$p(s', s, \mathbf{r}) = \beta_{l+1}(s)\gamma_l(s', s)\alpha_l(s').$$

$$\begin{aligned}
\alpha_{l+1}(s) = p(s, \mathbf{r}_{t<l+1}) &= \sum_{s' \in \sigma_l} p(s', s, \mathbf{r}_{t<l+1}) \\
&= \sum_{s' \in \sigma_l} p(s, \mathbf{r}_l | s', \mathbf{r}_{t<l}) p(s', \mathbf{r}_{t<l}) \\
&= \sum_{s' \in \sigma_l} p(s, \mathbf{r}_l | s') p(s', \mathbf{r}_{t<l}) \\
&= \sum_{s' \in \sigma_l} \gamma_l(s', s) \alpha_l(s'),
\end{aligned}$$

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

# BCJR (cont.)

$$\beta_l(s') = \sum_{s \in \sigma_{l+1}} \gamma_l(s', s)\beta_{l+1}(s)$$

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

$$\begin{aligned} \gamma_l(s', s) = p(s, \mathbf{r}_l | s') &= \frac{p(s', s, \mathbf{r}_l)}{P(s')} \\ &= \left[\frac{P(s', s)}{P(s')}\right]\left[\frac{p(s', s, \mathbf{r}_l)}{P(s', s)}\right] \\ &= P(s|s')p(\mathbf{r}_l|s', s) = P(u_l)p(\mathbf{r}_l|\mathbf{v}_l) \end{aligned}$$

AWGN

$$\gamma_l(s', s) = P(u_l)p(\mathbf{r}_l|\mathbf{v}_l) = P(u_l)\left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n e^{-\frac{E_s}{N_0}||\mathbf{r}_l - \mathbf{v}_l||^2}$$

# MAP algorithm

- Initialize forward and backward recursions $\alpha_0(s)$ and $\beta_N(s)$
- Compute branch metrics $\{\gamma_l(s', s)\}$
- Carry out forward recursion $\{\alpha_{l+1}(s)\}$ based on $\{\alpha_l(s)\}$
- Carry out backward recursion $\{\beta_{l-1}(s)\}$ based on $\{\beta_l(s)\}$
- Compute APP *L*-values
- Complexity: Approximately 3xViterbi
- Requires detailed knowledge of SNR
  - Viterbi just maximizes $\mathbf{r}\cdot\mathbf{v}$, and does not require exact knowledge of SNR

# BCJR (cont.)

$$\gamma_l(s', s) = P(u_l)e^{-E_s/N_0||\mathbf{r}_l - \mathbf{v}_l||^2}$$

$$\begin{aligned} P(u_l = \pm 1) &= \frac{[P(u_l = +1)/P(u_l = -1)]^{\pm 1}}{\{1 + [P(u_l = +1)/P(u_l = -1)]^{\pm 1}\}} \\ &= \frac{e^{\pm L_a(u_l)}}{\{1 + e^{\pm L_a(u_l)}\}} \\ &= \frac{e^{-L_a(u_l)/2}}{\{1 + e^{-L_a(u_l)}\}} e^{u_l L_a(u_l)/2} \\ &= A_l e^{u_l L_a(u_l)/2}, \end{aligned}$$

$$\begin{aligned} \gamma_l(s', s) &= A_l e^{u_l L_a(u_l)/2} e^{-(E_s/N_0)||\mathbf{r}_l - \mathbf{v}_l||^2}, \\ &= A_l e^{u_l L_a(u_l)/2} e^{(2E_s/N_0)(\mathbf{r}_l \cdot \mathbf{v}_l) - ||\mathbf{r}_l||^2 - ||\mathbf{v}_l||^2} \\ &= A_l e^{-(||\mathbf{r}_l||^2 + n)} e^{u_l L_a(u_l)/2} e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)} \\ &= A_l B_l e^{u_l L_a(u_l)/2} e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)}, \quad l = 0, 1, \cdots, h-1, \end{aligned}$$

Information bits

Termination bits

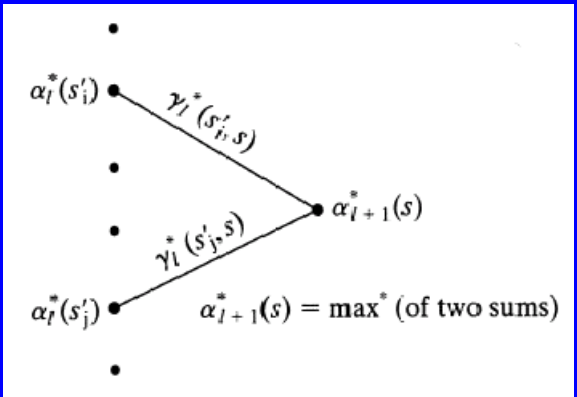
$$\begin{aligned} \gamma_l(s', s) &= P(u_l)e^{-(E_s/N_0)||\mathbf{r}_l - \mathbf{v}_l||^2} \\ &= e^{-(E_s/N_0)||\mathbf{r}_l - \mathbf{v}_l||^2}, \\ &= B_l e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)}, \quad l = h, h+1, \cdots, K-1, \end{aligned}$$

# BCJR (cont.)

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

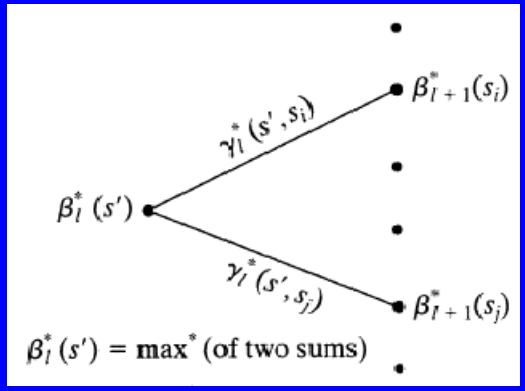
$$\gamma_l^*(s', s) \equiv \ln\ \gamma_l(s', s) = \begin{cases} \frac{u_l L_a(u_l)}{2} + \frac{L_c}{2}\mathbf{r}_l \cdot \mathbf{v}_l, & l = 0, 1, \cdots, h-1, \\ \frac{L_c}{2}\mathbf{r}_l \cdot \mathbf{v}_l, & l = h, h+1, \cdots, K-1, \end{cases}$$

$$\begin{aligned} \alpha_{l+1}^*(s) \equiv \ln\ \alpha_{l+1}(s) &= \ln \sum_{s' \in \sigma_l} \gamma_l(s', s)\alpha_l(s') \\ &= \ln \sum_{s' \in \sigma_l} e^{[\gamma_l^*(s', s) + \alpha_l^*(s')]} \\ &= \max^*_{s' \in \sigma_l}[\gamma_l^*(s', s) + \alpha_l^*(s')], \quad l = 0, 1, \cdots, K-1 \end{aligned}$$




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

$$\begin{aligned} \beta_l^*(s') \equiv \ln\ \beta_l(s') &= \ln \sum_{s \in \sigma_{l+1}} \gamma_l(s', s)\beta_{l+1}(s) \\ &= \ln \sum_{s \in \sigma_{l+1}} e^{[\gamma_l^*(s', s) + \beta_{l+1}^*(s)]} \\ &= \max^*_{s \in \sigma_{l+1}}[\gamma_l^*(s', s) + \beta_{l+1}^*(s)], \quad l = K-1, K-2, \end{aligned}$$




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

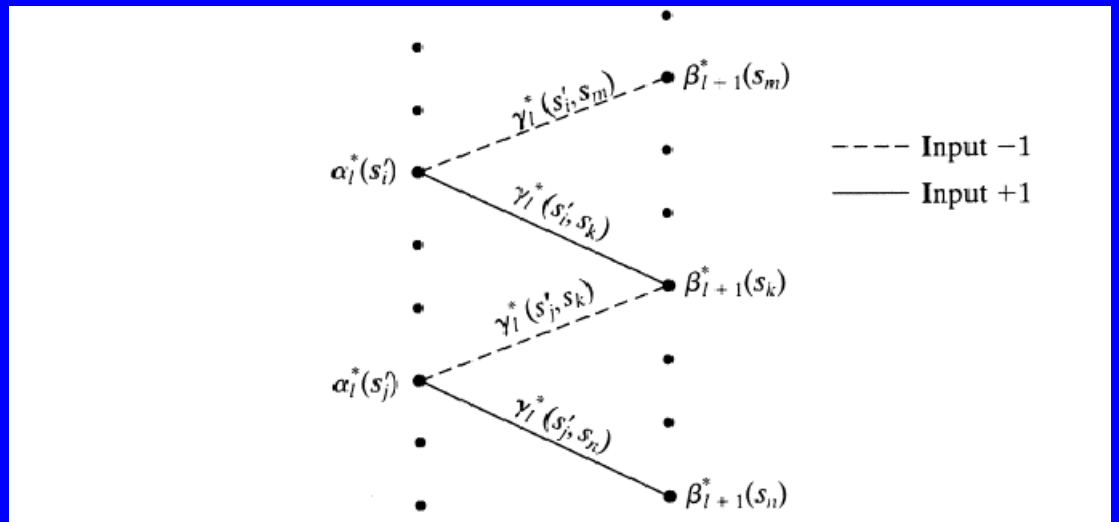
# BCJR (cont.)

$$p(s', s, \mathbf{r}) = e^{\beta^*_{l+1}(s)+\gamma^*_l(s',s)+\alpha^*_l(s')}$$

$$L(u_l) = \ln\left\{\sum_{(s',s)\in\Sigma_l^+} e^{\beta^*_{l+1}(s)+\gamma^*_l(s',s)+\alpha^*_l(s')}\right\} - \ln\left\{\sum_{(s',s)\in\Sigma_l^-} e^{\beta^*_{l+1}(s)+\gamma^*_l(s',s)+\alpha^*_l(s')}\right\}$$

$$\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z]$$

$$L(u_l) = \max^*_{(s',s)\in\Sigma_l^+}[\beta^*_{l+1}(s) + \gamma^*_l(s', s) + \alpha^*_l(s')] - \max^*_{(s',s)\in\Sigma_l^-}[\beta^*_{l+1}(s) + \gamma^*_l(s', s) + \alpha^*_l(s')]$$




$L(u_l) = \max^*(\beta^*_{l+1} + \gamma^*_l + \alpha^*_l$ for solid lines$) - \max^*(\beta^*_{l+1} + \gamma^*_l + \alpha^*_l$ for dashed lines$)$

# Log-MAP algorithm

- Initialize forward and backward recursions $\alpha_0^*(s)$ and $\beta_N^*(s)$
- Compute branch metrics $\{\gamma_l^*(s', s)\}$
- Carry out forward recursion $\{\alpha_{l+1}^*(s)\}$ based on $\{\alpha_l^*(s)\}$
- Carry out backward recursion $\{\beta_{l-1}^*(s)\}$ based on $\{\beta_l^*(s)\}$
- Compute APP *L*-values
- Advantages over MAP algorithm:
  - Easier to implement
  - Numerically more stable

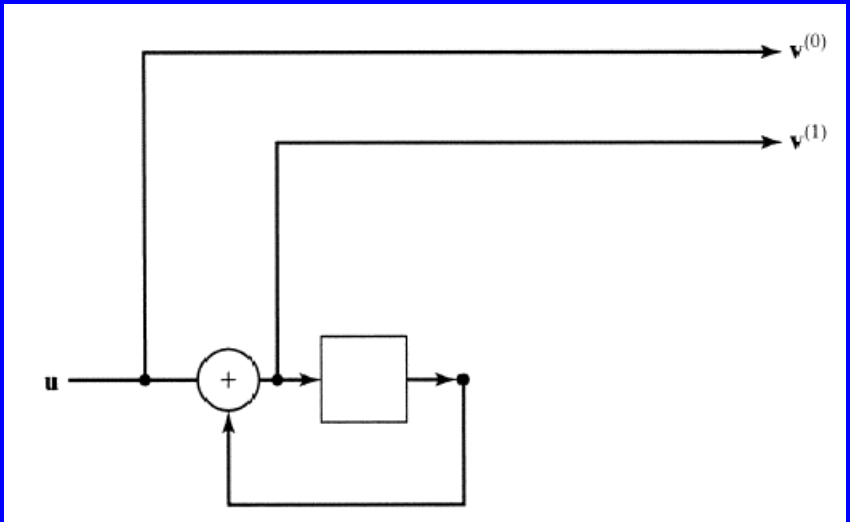
# Max-log-MAP algorithm

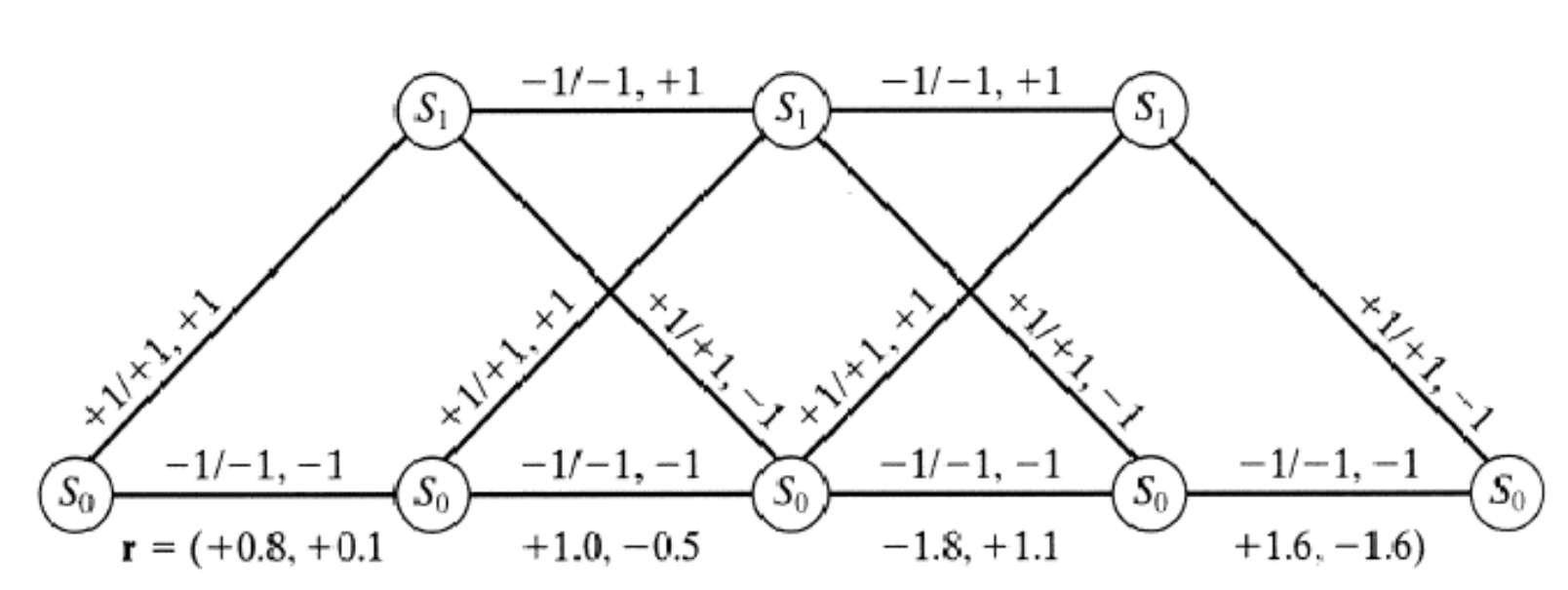
$$\max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})$$

- Replace max* by max, i.e., remove table look-up correction term
- Advantage: Simpler and much faster
- Forward and backward passes are equivalent to a Viterbi decoder
- Disadvantage: Less accurate, but the correction term is limited in size by ln(2)
- Can improve accuracy by scaling with an SNR-(in)dependent scaling factor*

# Example: log-MAP

$$\mathbf{G}(D) = [1 \;\; 1/(1+D)]$$

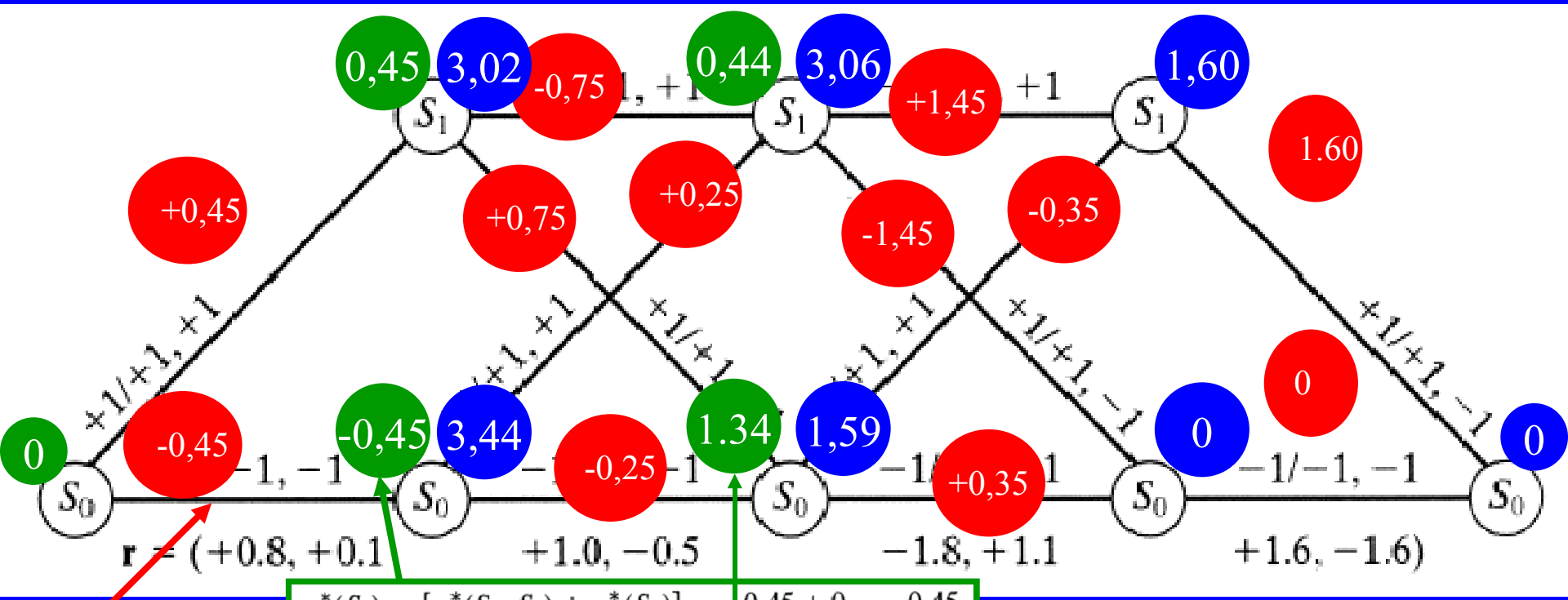


# Example: log-MAP

γ α β

- Assume $E_s/N_0 = 1/4 = -6.02$ dB
- $R = 3/8$, so $E_b/N_0 = 2/3 = -1.76$dB




$$\alpha_1^*(S_0) = [\gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] = -0.45 + 0 = -0.45$$

$$\gamma_0^*(S_0, S_0) = \frac{-1}{2} L_a(u_0) + \frac{1}{2}\mathbf{r}_0 \cdot \mathbf{v}_0$$
$$= \frac{1}{2}(-0.8 - 0.1) = -0.45$$

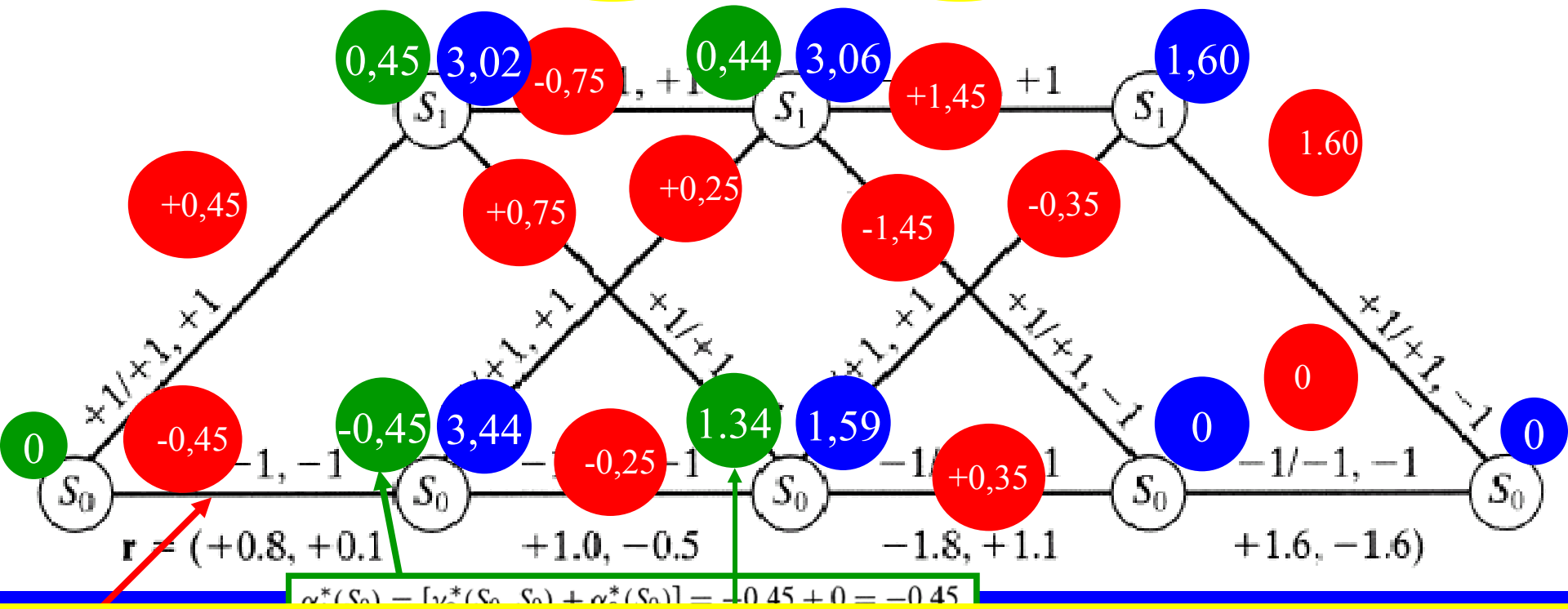
$$\alpha_2^*(S_0) = \max^*\{[\gamma_1^*(S_0, S_0) + \alpha_1^*(S_0)], [\gamma_1^*(S_1, S_0) + \alpha_1^*(S_1)]\}$$
$$= \max^*\{[(-0.25) + (-0.45)], [(0.75) + (0.45)]\}$$
$$= \max^*(-0.70, +1.20) = 1.20 + \ln(1 + e^{-|-1.9|}) = 1.34$$

# Ex[illegible] · l[illegible] P

- [illegible] $N_0 = 1$[illegible]
- [illegible] $E_b/N_0 =$ [illegible] dB

+0.48 +0.62 -1,02

γ α β




$$L(u_0) = [\beta_1^*(S_1) + \gamma_0^*(S_0, S_1) + \alpha_0^*(S_0)] - [\beta_1^*(S_0) + \gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)]$$

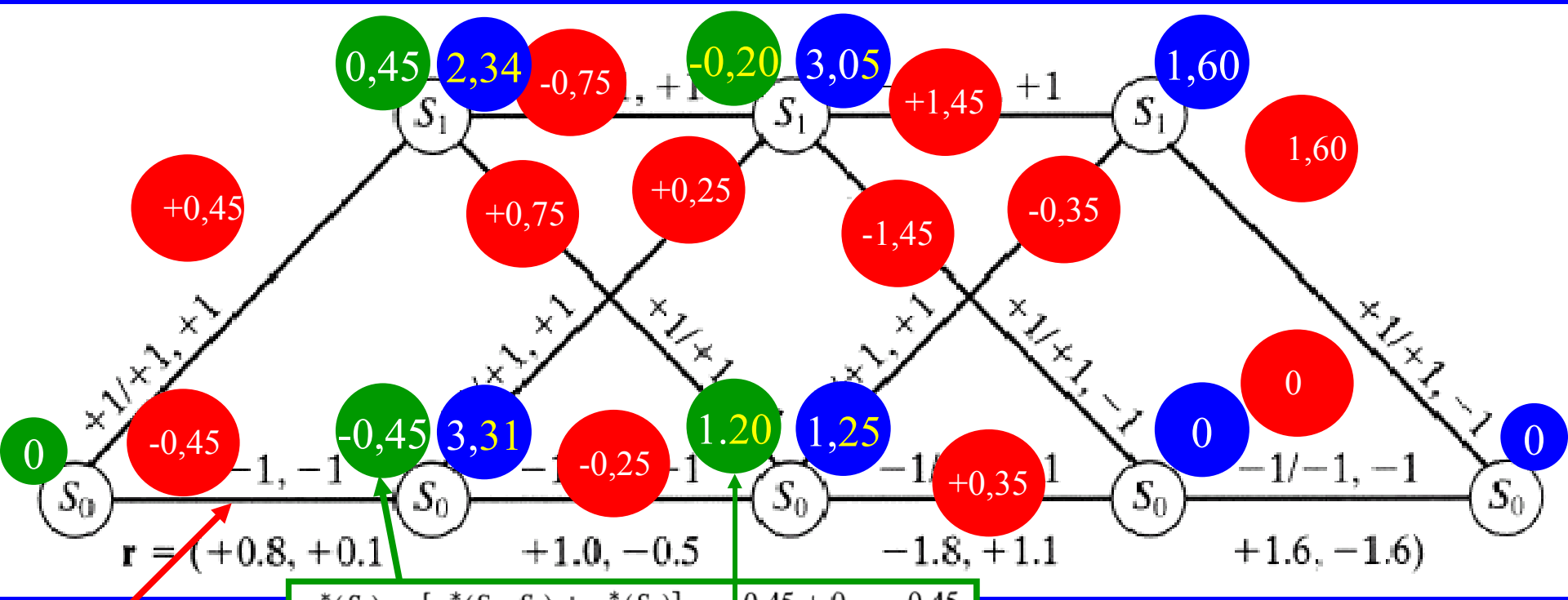
$$= (3.47) - (2.99) = +0.48$$

$$= \max^*(-0.70, +1.20) = 1.20 + \ln(1 + e^{-|-1.9|}) = 1.34$$

# Example: Max-log-MAP

- Assume $E_s/N_0 = 1/4 = -6.02$ dB
- $R = 3/8$, so $E_b/N_0 = 2/3 = -1.76$dB

γ α β




$$\alpha_1^*(S_0) = [\gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] = -0.45 + 0 = -0.45$$

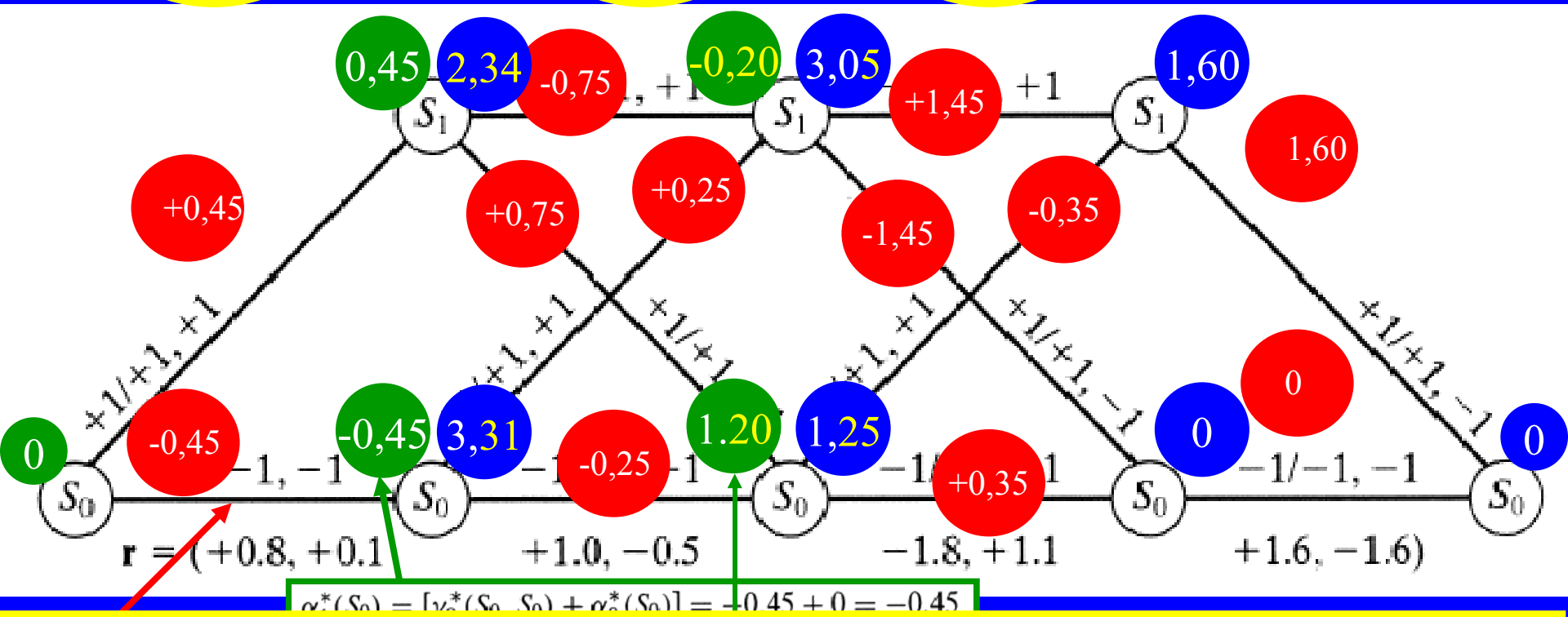
$$\gamma_0^*(S_0, S_0) = \frac{-1}{2} L_a(u_0) + \frac{1}{2}\mathbf{r}_0 \cdot \mathbf{v}_0 = \frac{1}{2}(-0.8 - 0.1) = -0.45$$

$$\alpha_2^*(S_0) = \max(-0.70, +1.20) = 1.20$$

# Exam Max MAP

- $_0 = 1/4$
- $E_b/N_0 = 2$ B

-0,07 +0,10 -0,40 γ α β




$\alpha_0^*(S_0) = [\gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] = +0.45 + 0 = -0.45$

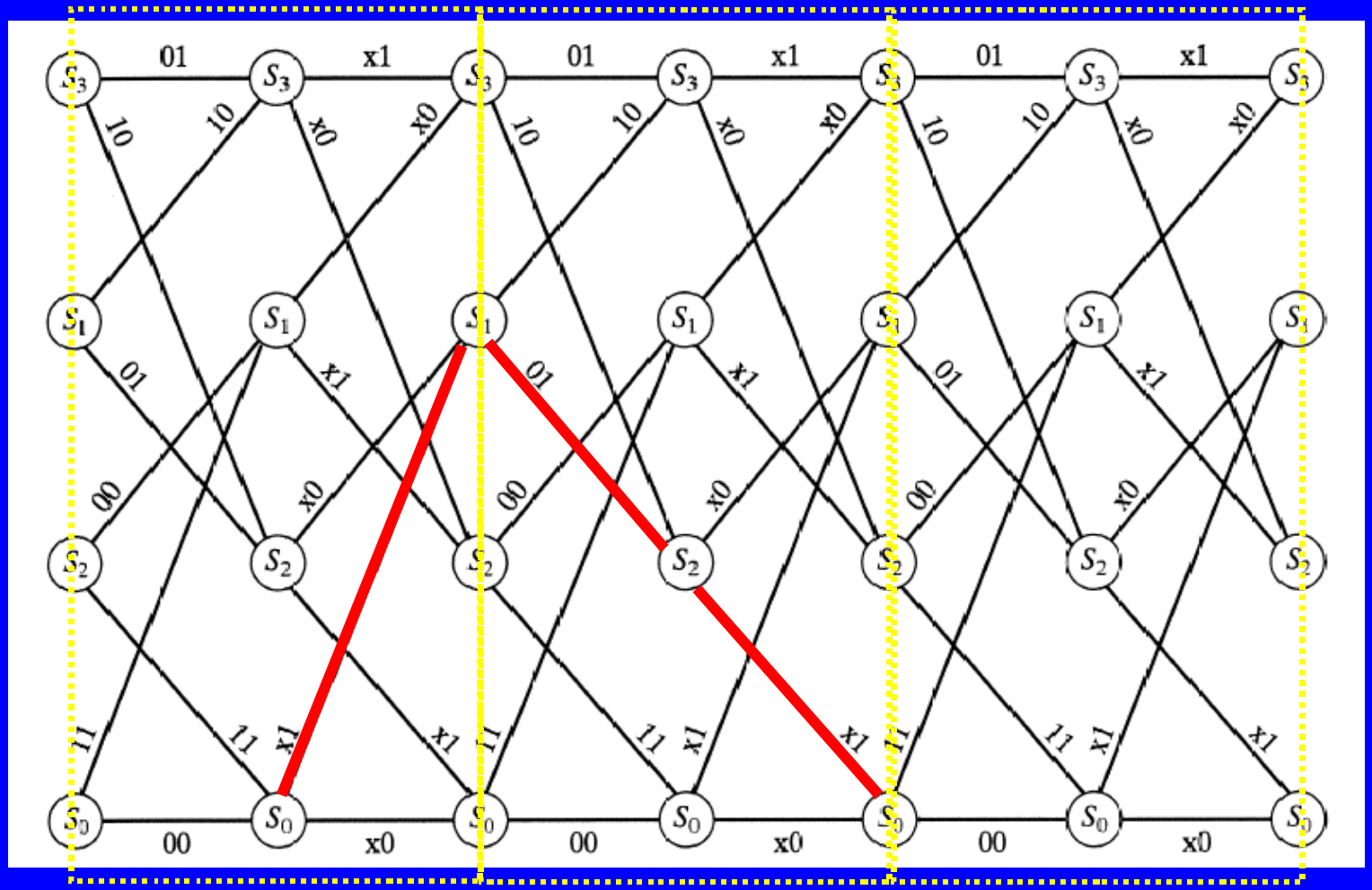
$$L(u_0) = [\beta_1^*(S_1) + \gamma_0^*(S_0, S_1) + \alpha_0^*(S_0)] - [\beta_1^*(S_0) + \gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)]$$

$$= (2.79) - (2.86) = -0.07$$

# Punctured convolutional codes

- Recall that an $(n,k)$ convolutional code has a decoder trellis with $2^k$ branches going into each state
- More complex decoding
- Solutions:
  - Bit-level encoders
  - Syndrome trellis decoding (Riedel)*
  - Punctured codes
    - Start with low-rate convolutional mother code (rate $1/n$?)
    - Puncture (delete) some code bits according to a predetermined pattern
    - Punctured bits are not transmitted. Hence, the code rate is increased, but the free distance of the code could be reduced
    - Decoder inserts dummy bits with neutral metrics contribution

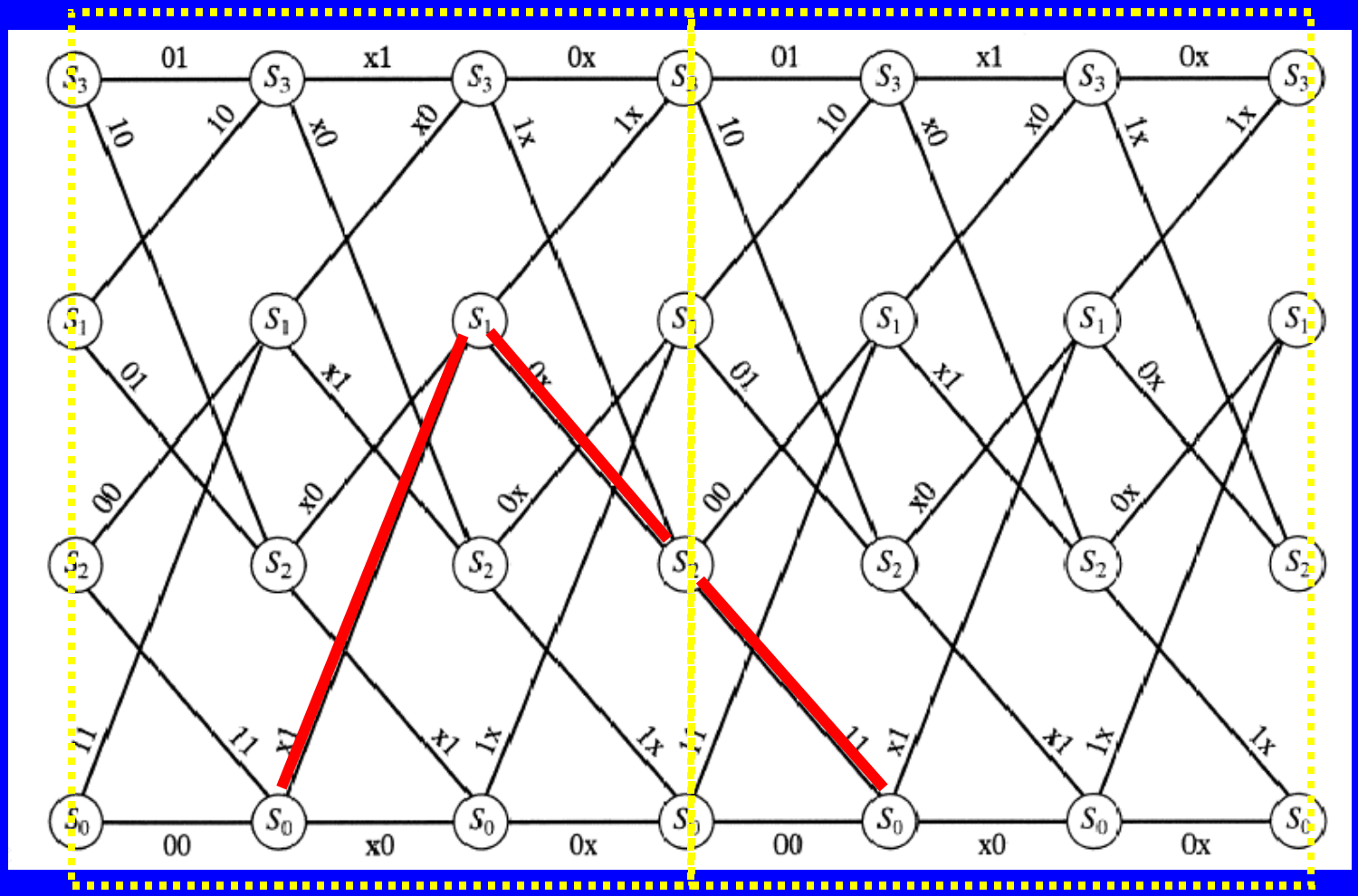
# Example: Rate 2/3 punctured from rate 1/2




The punctured code is also a convolutional code

$d_{\text{free}} = 3$

# Example: Rate 3/4 punctured from rate 1/2




$d_{\text{free}} = 3$

# More on punctured convolutional codes

- Rate-compatible punctured convolutional (RCPC) codes:
  - Used for applications that need to support several code rates, e.g., adaptive coding or hybrid ARQ
  - Sequence of codes is obtained by repeated puncturing
  - Advantage: One decoder can decode all codes in the family
  - Disadvantage: Resulting codes may be sub-optimum
- Puncturing patterns:
  - Usually periodic puncturing patterns
  - Found by computer search
  - Care must be exercised to avoid catastrophic encoders

# Best punctured codes

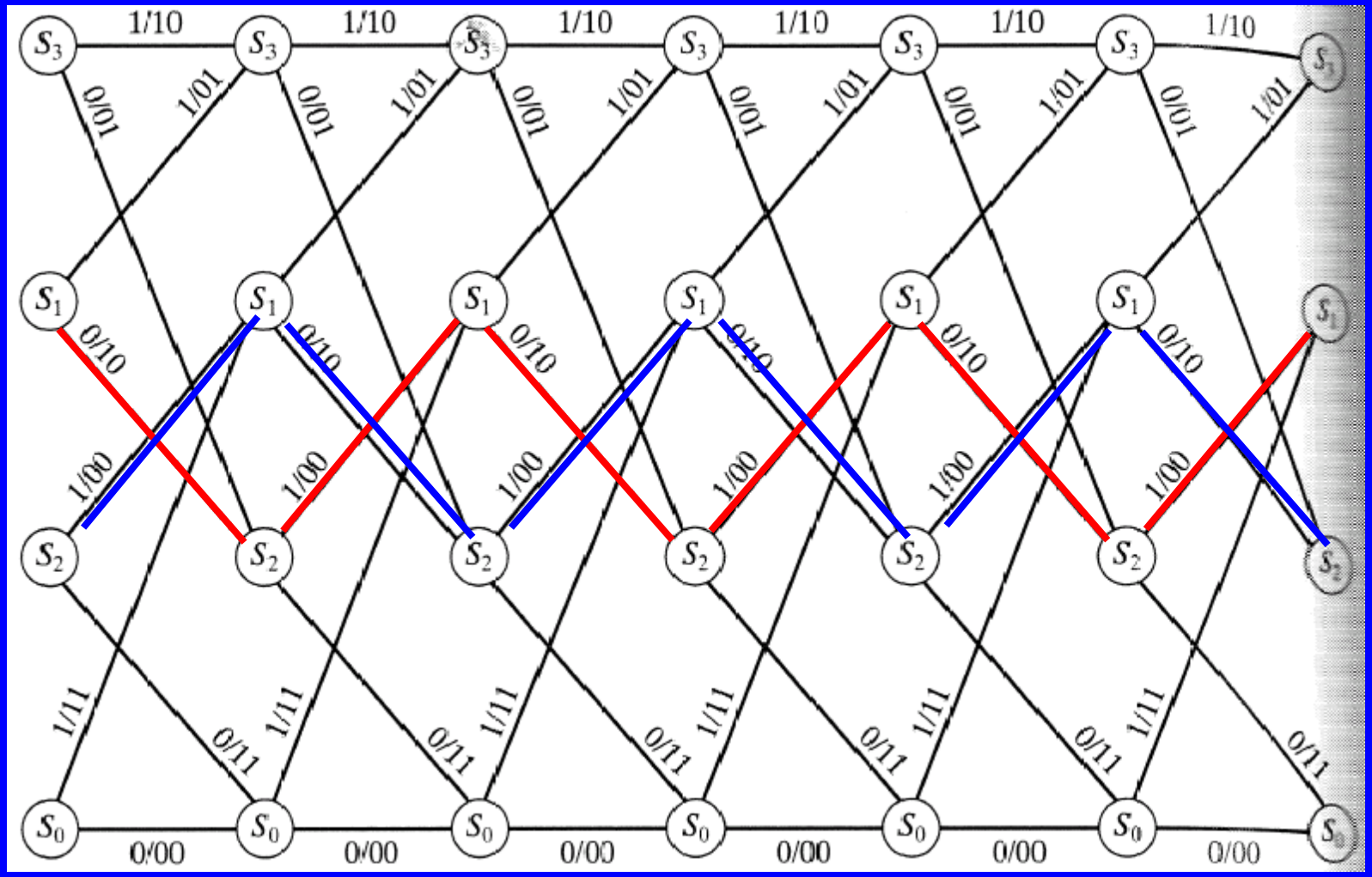
| Mother Code | | | Punctured Code | | | |
|---|---|---|---|---|---|---|
| $\nu$ | $g^{(0)}$ | $g^{(1)}$ | P | $d_{free}$ | $A_{d_{free}}$ | |
| 2 | 5 | 7 | 1 0<br>1 1 | 3 | 1 | |
| 3 | 13 | 17 | 1 1<br>0 1 | 4 | 3 | 1 |
| 4 | 31 | 27 | 1 1<br>0 1 | 4 | 1 | 3 |
| 5 | 65 | 57 | 1 0<br>1 1 | 6 | 19 | 7 |
| 6 | 155 | 117 | 1 1<br>1 0 | 6 | 1 | 17 |

| Mother Code | | | Punctured Code | | | |
|---|---|---|---|---|---|---|
| $\nu$ | $g^{(0)}$ | $g^{(1)}$ | P | $d_{free}$ | $A_{d_{free}}$ | |
| 2 | 5 | 7 | 1 0 1<br>1 1 0 | 3 | 6 | |
| 3 | 13 | 17 | 1 1 0<br>1 0 1 | 4 | 29 | 10 |
| 4 | 31 | 27 | 1 0 1<br>1 1 0 | 3 4 | 1 | 2 |
| 5 | 65 | 57 | 1 0 0<br>1 1 1 | 4 5 | 1 | 7 |
| 6 | 155 | 117 | 1 1 0<br>1 0 1 | 5 6 | 8 | 27 |

# Tailbiting convolutional codes

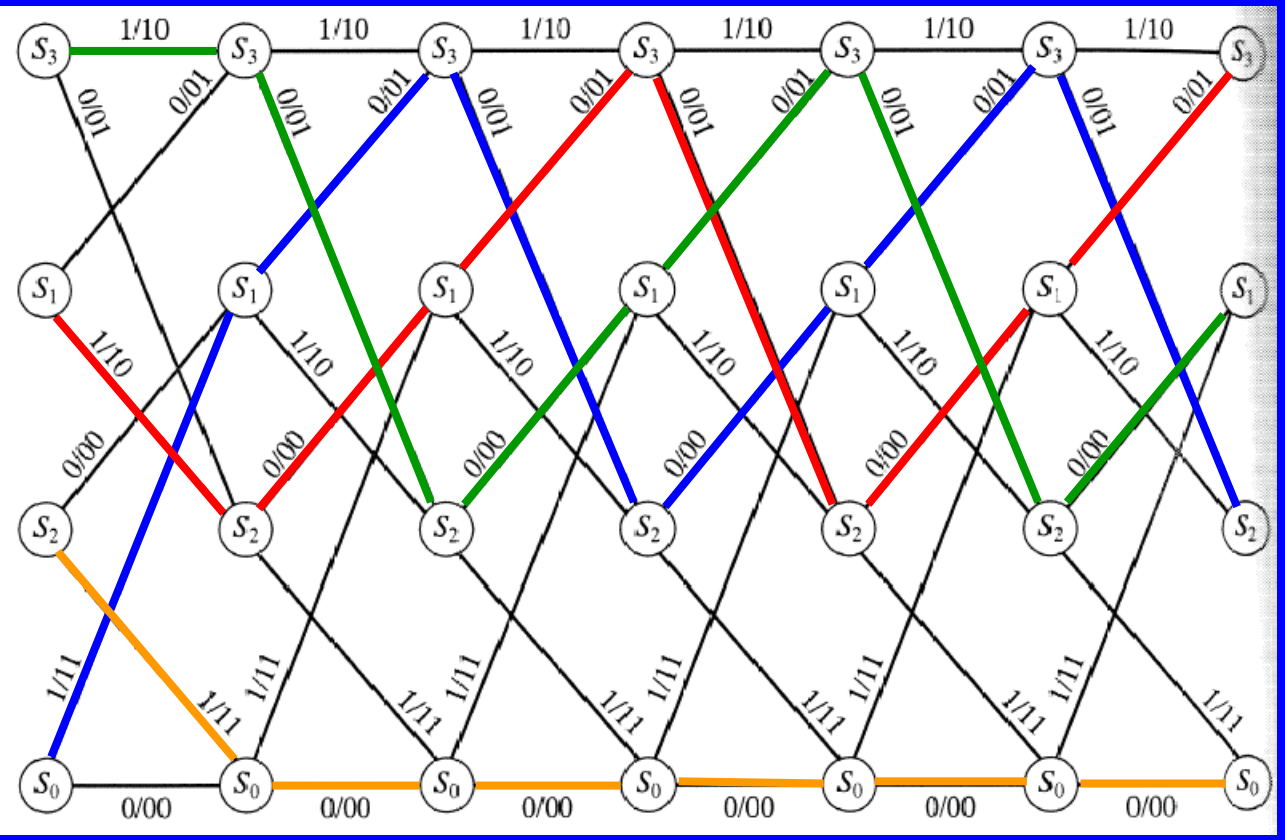
- Purpose: Avoid the terminating tail (rate loss) and maintain a uniform level of protection
  - Note: Cannot avoid distance loss completely unless the length is not too short. When the length gets larger, the minimum distance approaches the free distance of the convolutional code
- Codewords can start in any state
  - This gives $2^{\nu}$ as many codewords
  - However, each codeword must end in the same state that it started from. This gives $2^{-\nu}$ as many codewords
  - Thus, the code rate is equal to the encoder rate
- Tailbiting codes are increasingly popular for moderate length purposes
- Some of the best known linear block codes are tailbiting codes
- Tables of optimum tailbiting codes are given in the book
- DVB: Turbo codes with tailbiting component codes

# Example: Feedforward encoder




Feedforward encoder: Always possible to find an information vector that ends in the proper state (inspect the last *m* *k*-bit input tuples)

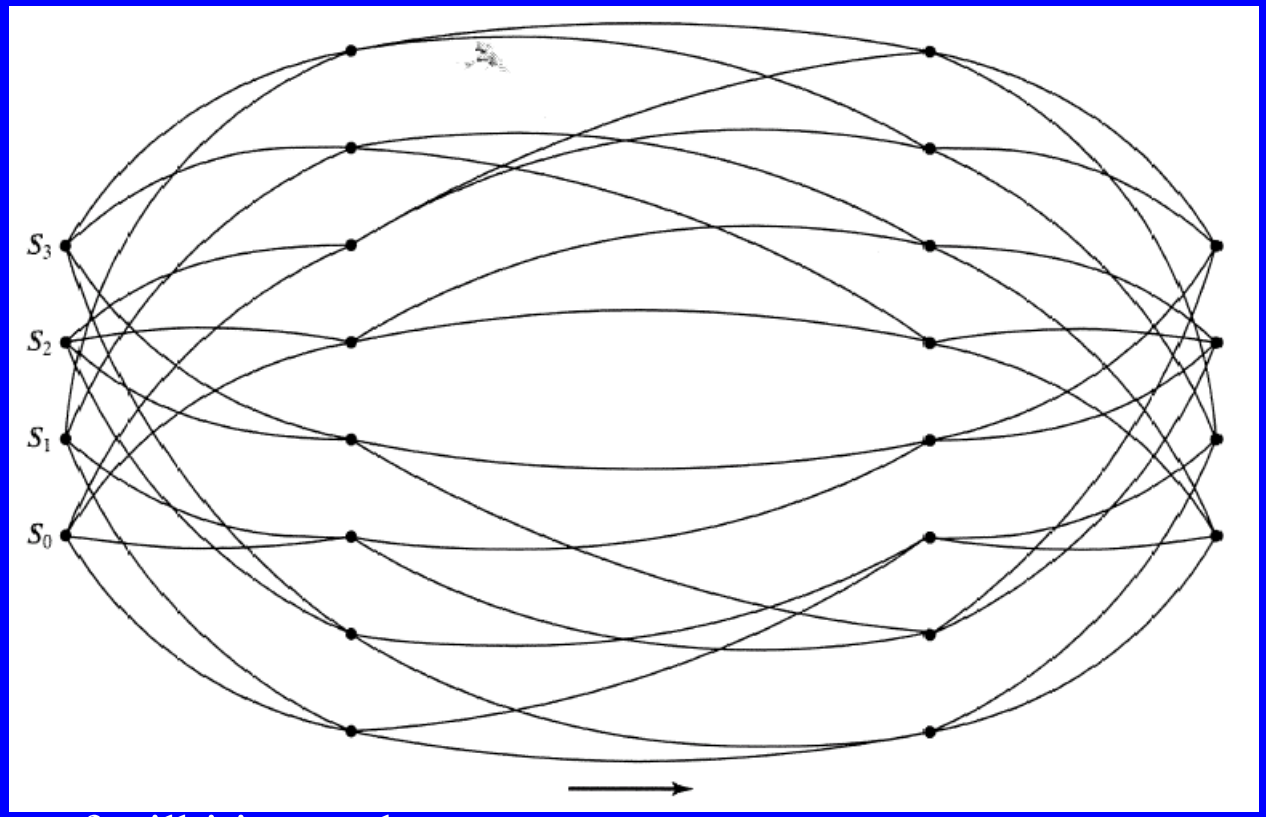
# Example: Feedback encoder




| $S_0$ | $S_1$ | $S_2$ | $S_3$ |
|---|---|---|---|
| 0000000000 | **1000010100** | **0010000101** | **0100100001** |
| 1101011100 | **0101001000** | **0001010010** | 1010010001 |
| 0011010111 | 0110100100 | 1100110101 | 1001000110 |
| 1110110000 | 0101110011 | 0001101001 | 0111001101 |
| 1101100111 | 1011000011 | 1111011001 | 0100011010 |
| 0011101100 | 0110011111 | 1100001110 | 1001111101 |
| 0000111011 | 1011111000 | 1111100010 | 0111110110 |
| 1110001011 | 1000101111 | 0010111110 | 1010101010 |

- Feedback encoder: Not always possible, for every length, to construct a tailbiting code
- For each **u**: Must find unique starting state
- $L^* = 6$ not OK
- $L^* = 5$ OK
- In general, $L^*$ should not have the length of a zero input-weight cycle as a divisor

# ”Circular” trellis




- Decoding of tailbiting codes:
  - Try all possible starting states (multiplies complexity by $2^\nu$), i.e., run the Viterbi algorithm for each of the $2^\nu$ subcodes and compare the best paths from each subcode
  - Suboptimum Viterbi: Initialize an arbitrary state at time 0 with zero metric and find the best ending state. Continue ”one round” from there with the best subcode
  - MAP: Similar