Code design: Computer search

- Low rate codes
  - Represent the code by its generator matrix
    - Find one representative for each equivalence class of codes
      - Permutation equivalences?
      - Do NOT try several generator matrices for the same code?
    - Avoid non-minimal matrices (and of course catastrophic ones)
- In general, more distance is obtainable with nonsystematic feedforward encoders than with systematic feedforward encoders

Code design: Computer search*

- High rate codes
  - Represent the code by its parity check matrix
    - Find one representative for each equivalence class of codes
    - Avoid non-minimal matrices
- Problem: Branch complexity of rate k/n codes is large
  - One solution is to use punctured codes (to be described later)
  - Another solution is to use a bit-level trellis
  - Can limit complexity of bit-level trellis to $\nu+\beta$, where $0 \leq \beta \leq \min(n-k, k)$
  - Reference: E. Rosnes and Ø. Ytrehus, "Maximum length convolutional codes under a trellis complexity constraint," *J. of Complexity*, vol. 20, pp. 372-403, Apr.-Jun. 2004

Code design / Bounds on codes*

- Heller bound: $d_{\text{free}} \leq$ best minimum distance in any block code of the same parameters as any terminated code
  - McEliece: For rate $(n\text{-}1)/n$ codes with $d_{\text{free}} = 5$, $n \leq 2^{(\nu+1)/2}$ asymptotically as ν approaches infinity (Heller bound in combination with the Hamming bound)
- Sphere packing bound:
  - The bound applies to codes of odd d_{free}
  - Similar to the Hamming bound for block codes
  - Corollary: For rate $(n\text{-}1)/n$ codes of $d_{\text{free}} = 5$, $n \leq 2^{\nu/2}$ asymptotically as ν approaches infinity (Heller bound in combination with sphere packing bound)
  - Reference: E. Rosnes and Ø. Ytrehus, "Sphere packing bounds for convolutional codes," *IEEE Trans. on Information Theory*, vol. 50, pp. 2801-2809, Nov. 2004

Quick-look-in (QLI) encoders

- Subclass of rate ½ nonsystematic feedforward encoders with the property that $\mathbf{g}^{(0)}(D) + \mathbf{g}^{(1)}(D) = D$ and $g_0^{(0)} = g_0^{(1)} = g_m^{(0)} = g_m^{(1)} = 1$
- Non-catastrophic encoder
- Feed-forward inverse: $\mathbf{G}^{-1}(D) = [1, 1]^T$ since $\mathbf{G}(D)\mathbf{G}^{-1}(D) = D$
- Thus, an error in u_l is caused by an error in $v_{l+1}^{(0)}$ or $v_{l+1}^{(1)}$ (but not in both)
- The probability of such an error is $\cong 2p$, where p is the error probability of the channel (assuming a BSC)
- In general with $\mathbf{G}^{-1}(D) = [\mathbf{g}_0^{-1}(D), \mathbf{g}_1^{-1}(D)]^T$, the probability of an error in u_l is $\cong Ap$, where $A = w(\mathbf{g}_0^{-1}(D)) + w(\mathbf{g}_1^{-1}(D))$ (A is the *error probability amplification factor*)
- $A = 1$ for systematic encoders (feedforward or feedback)
- QLI codes are almost as good as the best codes and give better free distance than codes with systematic feedforward encoders

Code design: Construction from block codes

- Massey, Costello, and Justesen (uses $d_{\min}$ of a cyclic block code (and its dual) to provide a lower bound on d_{free})
  - Constr. 12.1 …skip
  - Constr. 12.2 …skip
- Other attempts
  - The Wyner code construction of $d_{\text{free}} = 3$ codes
    - Generalizations for $d_{\text{free}} = 3$ and 4 exist that are optimum
    - Reference: E. Rosnes and Ø. Ytrehus, "Maximum length convolutional codes under a trellis complexity constraint," *J. of Complexity*, vol. 20, pp. 372-403, Apr.-Jun. 2004
  - Other algebraic ("BCH-like") constructions (not in book) also exist
    - Warning: These construction are messy

Implementation issues

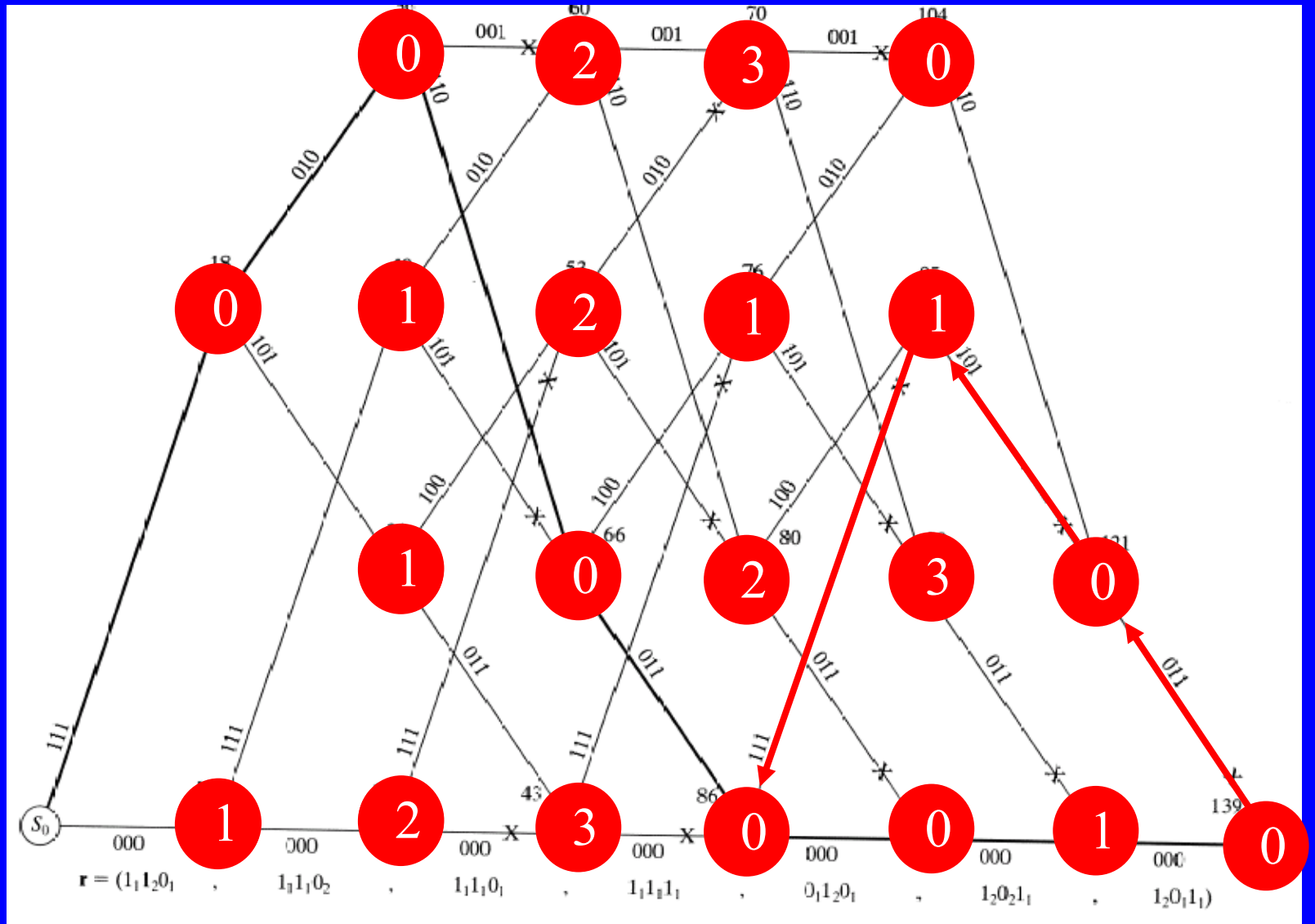
- Decoder memory
- Metrics dynamic range (normalization)*
- Path memory
- Decoder synchronization
- Receiver quantization

Decoder memory

- There is a limit to how large ν can be
- In practice, ν is seldom more than 4 (16 states)
- Note that there exists a Viterbi decoder implementation for $\nu = 14$, the so-called big Viterbi decoder (BVD)
- The practical soft-decision coding gains are in most cases limited to about 7 dB with convolutional codes

Dynamic range of metrics

- Normalize metric values after each time instant!



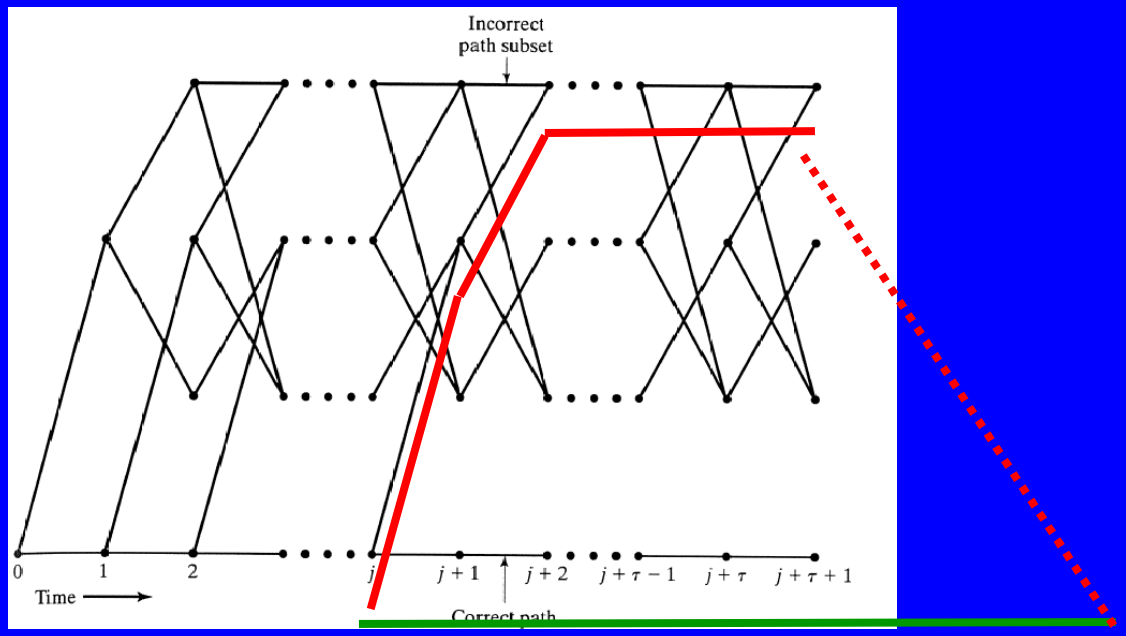
Path memory

- For large information length:
  - Impractical to store complete path before backtracing
    - Requires a large amount of memory
    - Imposes delay
  - Practical solution:
    - Select integer τ
    - After τ blocks: Start backtracing by either
      - Starting from the survivor in an arbitrary state, or the survivor in the state with the best metric
      - Alternatively, try backtracing from all states, and select the *k*-bit block that occurs most frequently in these backtracings
    - This determines the first *k*-bit information block. Subsequent information blocks are decoded successively in the same way

Truncation of the decoder

- Not maximum likelihood
- The truncated decoder can make two types of error
  - E_{ML} = the kind of error that an ML decoder would also make
    - Associated with low-weight paths (paths of weight d_{free}, $d_{free}+1, \ldots$)
    - $P(E_{ML}) \approx A_{d_{free}} \cdot (D_0)^{d_{free}}$, where $(D_0)^d$ depends on the channel and is decreasing rapidly as the argument d increases
  - E_T = the kind of error that is due to truncation, and which an ML decoder would not make
    - Why would an ML decoder not make those errors?
    - Because if the decoding decision is allowed to ”mature”, the erroneous codewords will not be preferred over the correct one. However, the truncated decoder enforces an early decision, and thus makes mistakes

Truncation length



- Assume the all-zero word is the correct word. Then

$$P(E) < \left[A(W, X, L) + \sum_{i=1}^{2^{\nu}-1} A_i^{\tau}(W, X, L) \right]_{X = D_0, W = 1, L = 1}$$

- where the last summation is the codeword WEF for the subset of incorrect paths that can cause decoding errors due to truncation
- Each term is the codeword WEF for the set of all unmerged paths of length more than τ branches that connect the all-zero state with the *i*-th state

Truncation length (cont.)

- Furthermore,

$$P(E) \cong A_{d_{\text{free}}} \left(D_0\right)^{d_{\text{free}}} + A_{d(\tau)} \left(D_0\right)^{d(\tau)}$$

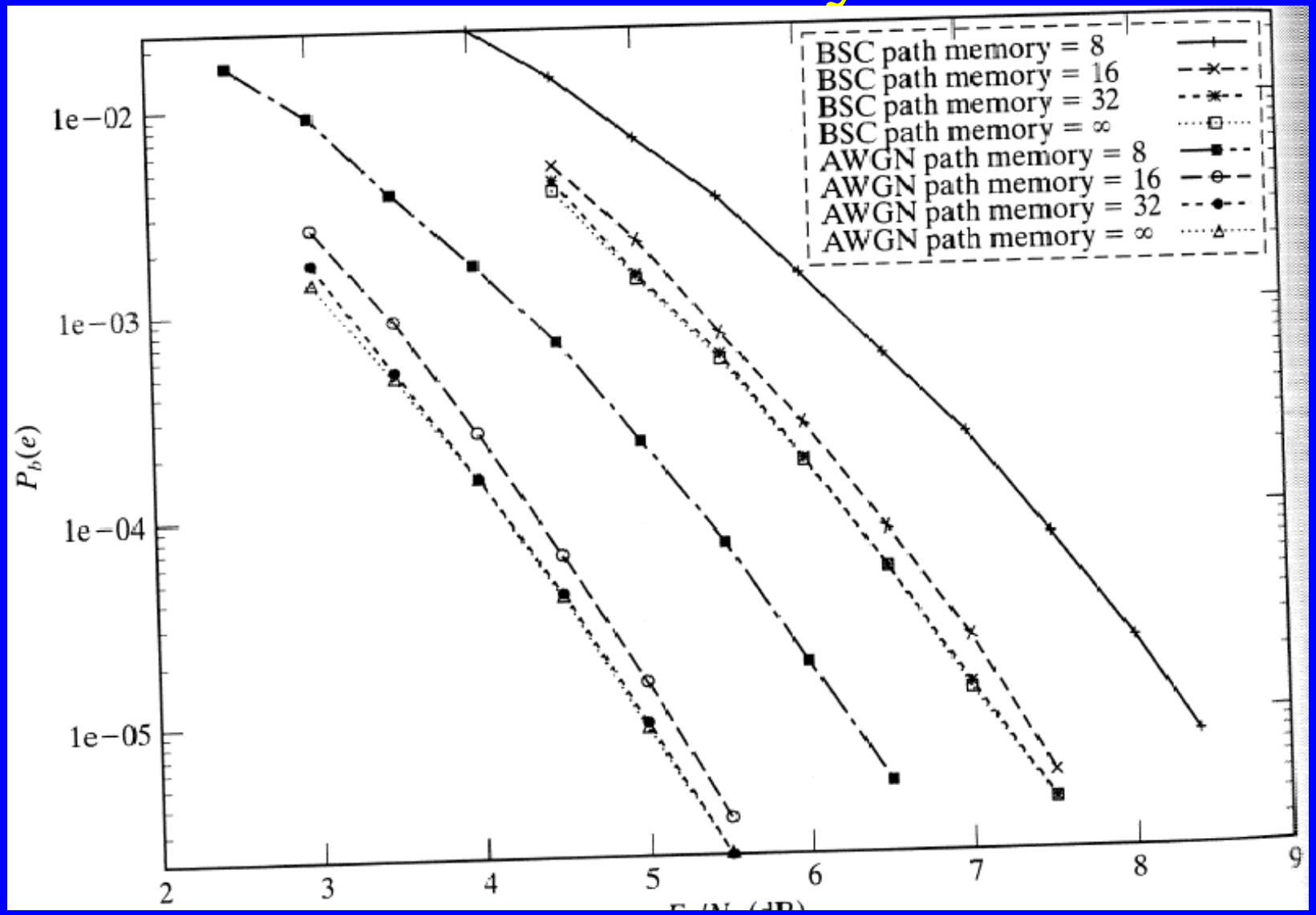
- under good channel conditions, where $d(\tau)$ is the minimum weight of a path that leaves the all-zero state at time j or earlier, and that is unmerged at time $j+\tau+1$
- We want to select τ such that $P(E_{\text{ML}}) >> P(E_{\text{T}})$ (i.e. the first term is much larger than the second term)
- The minimum length τ such that $d(\tau) > d_{\text{free}}$ is the minimum truncation length τ_{min}. The minimum truncation length is often 4 to 5 times the memory m
- Note: τ_{min} can be determined by a modified Viterbi algorithm
- Note the difference between the CDF d_l and $d(\tau)$. In particular, $d(\tau)$ increases without bound as τ increases (non-catastrophic encoders)

Example

TABLE 12.3: Minimum truncation lengths for rate $R = 1/2$ optimum free distance codes.

m	d_{free}	τ_{min}	$d(\tau_{min})$
2	5	7	6
3	6	9	7
4	7	14	8
5	8	18	9
6	10	26	11
7	10	27	11

R=1/2, ν=4 code with varying path memory



Synchronization

- Symbol synchronization: Which n bits belong together as a branch label?
  - Metrics evaluation: If metrics are consistently bad, this indicates loss of synchronization
    - May be enhanced by code construction*
  - May embed special synchronization patterns in the transmitted sequence
    - What if the noise is bad?
- Decoder: Do we know that the encoder starts in the all-zero state?
  - Not always
  - Solution: Discard the first $5m$ branches

Quantization

Skip this

Computational complexity issues

- Branch complexity (of high rate encoders)
  - Punctured codes
  - Bit-level encoders (equivalence to punctured codes)*
  - Syndrome trellis decoding (Riedel)*
- Speed-up:
  - Parallel processing
    - One processor per node
    - Differential Viterbi decoding (CSA instead of ACS). Gives a speed-up by a factor of about 1/3 for a large class of nonsystematic feedforward encoders

SISO decoding algorithms

- Two new elements
  - Soft-in, soft-out (SISO)
  - Allows non-uniform input probabilities
    - The a priori information bit probabilities $P(u_l)$ may be different
- Application: Turbo decoding
- Two most used algorithms:
  - The soft output Viterbi algorithm (SOVA)
  - The BCJR algorithm or the maximum a posteriori probability (MAP) algorithm

SOVA

$$
\begin{aligned}
M([\mathbf{r}|\mathbf{v}]_t) &= \ln\left\{\left[\prod_{l=0}^{t-1} p(\mathbf{r}_l|\mathbf{v}_l)P(u_l)\right] p(\mathbf{r}_t|\mathbf{v}_t)P(u_t)\right\} \\
&= \ln\left\{\prod_{l=0}^{t-1} p(\mathbf{r}_l|\mathbf{v}_l)P(u_l)\right\} + \ln\left\{\left[\prod_{j=0}^{n-1} p(r_t^{(j)}|v_t^{(j)})\right] P(u_t)\right\} \\
&= \ln\left\{\prod_{l=0}^{t-1} p(\mathbf{r}_l|\mathbf{v}_l)P(u_l)\right\} + \left\{\sum_{j=0}^{n-1} \ln\left[p(r_t^{(j)}|v_t^{(j)})\right] + \ln\left[P(u_t)\right]\right\}
\end{aligned}
$$

$$
\begin{aligned}
M^*([\mathbf{r}|\mathbf{v}]_t) &= M^*([\mathbf{r}|\mathbf{v}]_{t-1}) + \sum_{j=0}^{n-1}\left\{2\ln\left[p(r_t^{(j)}|v_t^{(j)})\right] - C_r^{(j)}\right\} + \left\{2\ln\left[P(u_t)\right] - C_u\right\} \\
&= M^*([\mathbf{r}|\mathbf{v}]_{t-1}) + \sum_{j=0}^{n-1} v_t^{(j)} \ln\left[\frac{p(r_t^{(j)}|v_t^{(j)} = +1)}{p(r_t^{(j)}|v_t^{(j)} = -1)}\right] + u_t \ln\left[\frac{p(u_t = +1)}{p(u_t = -1)}\right].
\end{aligned}
$$

SOVA (cont.)

$$L(r) = \ln\left[\frac{p(r|v=+1)}{p(r|v=-1)}\right].$$

$$L(u) = \ln\left[\frac{p(u=+1)}{p(u=-1)}\right].$$

$$L(r) = \ln\left[\frac{p(r|v=+1)}{p(r|v=-1)}\right] = \ln\left[\frac{p(v=+1|r)}{p(v=-1|r)}\right].$$

$$M^*([\mathbf{r}|\mathbf{v}]_t) = M^*([\mathbf{r}|\mathbf{v}]_{t-1}) + \sum_{j=0}^{n-1} L_c v_t^{(j)} r_t^{(j)} + u_t L(u_t),$$

$$\Delta_{t-1}(S_i) = \frac{1}{2}\{M^*([\mathbf{r}|\mathbf{v}]_t) - M^*([\mathbf{r}|\mathbf{v}']_t)\}.$$

$$P(C) = \frac{P([\mathbf{v}|\mathbf{r}]_t)}{\{P([\mathbf{v}|\mathbf{r}]_t) + P([\mathbf{v}'|\mathbf{r}]_t)\}}.$$

SOVA (cont.)

$$P([\mathbf{v}|\mathbf{r}]_t) = \frac{p([\mathbf{r}|\mathbf{v}]_t)P([\mathbf{v}]_t)}{p(\mathbf{r})} = \frac{e^{M([\mathbf{r}|\mathbf{v}]_t)}}{p(\mathbf{r})}$$

$$M^*([\mathbf{r}|\mathbf{v}]_t) = 2M([\mathbf{r}|\mathbf{v}]_t) - c$$

$$\begin{aligned} P(C) &= \frac{[e^{\{M^*([\mathbf{r}|\mathbf{v}]_t)/2\}+c}/p(\mathbf{r})]}{[e^{\{M^*([\mathbf{r}|\mathbf{v}]_t)/2\}+c}/p(\mathbf{r})] + [e^{\{M^*([\mathbf{r}|\mathbf{v}']_t)/2\}+c}/p(\mathbf{r})]} \\ &= \frac{e^{M^*([\mathbf{r}|\mathbf{v}]_t)/2}}{e^{M^*([\mathbf{r}|\mathbf{v}]_t)/2} + e^{M^*([\mathbf{r}|\mathbf{v}']_t)/2}} \\ &= \frac{e^{\Delta_{t-1}(S_i)}}{1 + e^{\Delta_{t-1}(S_i)}}. \end{aligned}$$

$$\ln\left\{\frac{P(C)}{[1 - P(C)]}\right\} = \Delta_{t-1}(S_i).$$

SOVA (cont.)

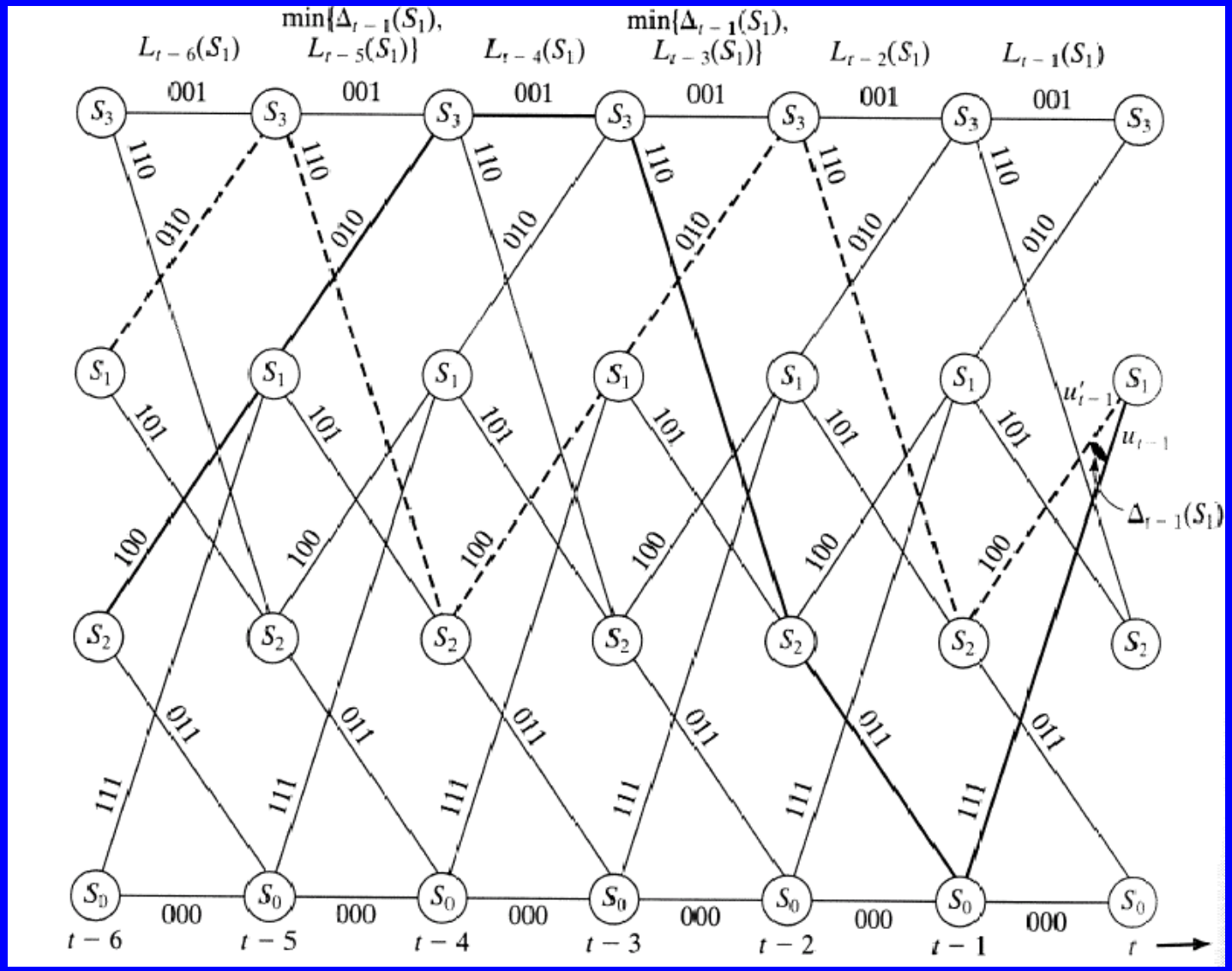
$$\mathbf{L}_{m+1}(S_i) = [L_0(S_i), L_1(S_i), \cdots, L_m(S_i)],$$

$$L_l(S_i) \equiv \begin{cases} \Delta_m(S_i) & \text{if } u_l \neq u_l' \\ \infty & \text{if } u_l = u_l' \end{cases}, \quad l = 0, 1, \cdots, m.$$

$$\mathbf{L}_t(S_i) = [L_0(S_i), L_1(S_i), \cdots, L_{t-1}(S_i)],$$

$$L_l(S_i) \rightarrow \begin{cases} \min[\Delta_{t-1}(S_i), \ L_l(S_i)] & \text{if } u_l \neq u_l' \\ L_l(S_i) & \text{if } u_l = u_l' \end{cases}, \quad l = 0, 1, \cdots, t-1.$$

SOVA update


$L_{t-6}(S_1)$
$\min\{\Delta_{t-1}(S_1), L_{t-5}(S_1)\}$
$L_{t-4}(S_1)$
$\min\{\Delta_{t-1}(S_1), L_{t-3}(S_1)\}$
$L_{t-2}(S_1)$
$L_{t-1}(S_1)$
001
110
010
101
100
011
111
000
S_3
S_1
S_2
S_0
u'_{t-1}
u_{t-1}
$\Delta_{t-1}(S_1)$
$t-6$
$t-5$
$t-4$
$t-3$
$t-2$
$t-1$
t

SOVA

- Finite path memory: Straightforward
- The complexity is just slightly increased compared with the complexity of the Viterbi algorithm
  - Storage complexity: Need to store reliability vectors in addition to metrics and survivor pointers
  - Computational complexity: Need to update the reliability vector
- Provides ML decoding (minimizes WER) and reliability measure
- It is NOT MAP decoding and does not minimize the BER