Weight enumerators

a) Weight enumerating function (WEF) $A(X) = \sum_d A_d X^d$

b) Input-output weight enumerating function (IOWEF)

- $A(W,X,L) = \sum_{w,d,l} A_{w,d,l} W^w X^d L^l$
- Gives the most complete information about weight structure
- From the IOWEF we can derive other enumerator functions:
  - WEF (set W=L=1)
  - Conditional WEF (CWEF): Consider a given input weight
  - Bit CWEF / Bit IOWEF / Bit WEF
  - Input-redundancy WEFs (IRWEFs)
  - WEFs of truncated / terminated codes

Conditional WEF (CWEF)

a) $A_w(X) = \sum_d A_{w,d} X^d$, where $A_{w,d}$ is the number of codewords of information weight w and codeword weight d

b) An encoder property

c) Useful for analyzing turbo codes with convolutional codes as component codes

Truncated / terminated encoders

a) Output length limited to $\lambda = h + m$ blocks
   - h is the number of input blocks
   - m is the number of terminating output blocks (the tail) necessary to bring the encoder back to the initial state

b) For a terminated code, apply the following procedure:
   - Write the IOWEF $A(W,X,L)$ in increasing order of L
   - Delete the terms of L-degree larger than λ

Do we count all codewords?

a) No
   - Only those that start at time 0
     - Why?
     - Each time instant is similar (for a time-invariant code)
     - The Viterbi decoding algorithm (ML on trellis) makes decisions on *k* input bits at a time. Thus, any error pattern will start at some time, and the error pattern will be structurally similar to an error pattern starting at time 0
   - Only first event paths
     - Why?
     - Same as above

b) Thus, the FER/BER calculation depends only on the first event errors that start at time 0

BER calculation

a) Bit CWEF $B_w(X) = \sum_d B_{w,d} X^d$, where $B_{w,d} = (w/k) A_{w,d}$ is the total number of non-zero information bits associated with all codewords of weight d and produced by information sequences of weight w, divided by k

b) Bit IOWEF $B(W,X,L) = \sum_{w,d,l} B_{w,d,l} W^w X^d L^l$

c) Bit WEF $B(X) = \sum_d B_d X^d = \sum_{w,d} B_{w,d} W^w X^d |_{W=1}$

$$= \sum_{w,d} (w/k) A_{w,d} W^w X^d |_{W=1}$$

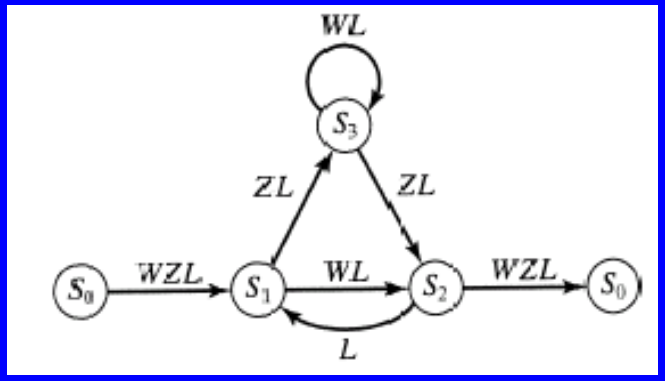
$$= 1/k \ \partial(\sum_{w,d} A_{w,d} W^w X^d)/ \partial W |_{W=1}$$

IRWEF

a) Systematic encoders: Codeword weight $d = w + z$, where z is the parity weight

b) Instead of the IOWEF $A(W,X,L) = \sum_{w,d,l} A_{w,d,l}\, W^w X^d L^l$
we may (and in some cases it is more convenient to) consider the input-redundancy WEF $A(W,Z,L) = \sum_{w,z,l} A_{w,z,l}\, W^w Z^z L^l$

Alternative to Mason's formula



a) Introduce state variables Σ_i giving the weights of all paths from S_0 to state S_i

- $\Sigma_1 = WZL + L \cdot \Sigma_2$
- $\Sigma_2 = WL \cdot \Sigma_1 + ZL \cdot \Sigma_3$
- $\Sigma_3 = ZL \cdot \Sigma_1 + WL \cdot \Sigma_3$
- $A(W,Z,L) = WZL \cdot \Sigma_2$

b) Solve this set of linear equations

Distance properties

a) The decoding method determines what is actually the most important distance property
   - ML decoding: The free distance of the code
   - Sequential decoding: The column distance function (CDF)
   - Majority logic decoding: The minimum distance of the code

Free distance

a) $d_{\text{free}} = \min_{\mathbf{u},\mathbf{u}'}\{d(\mathbf{v},\mathbf{v}'): \mathbf{u} \neq \mathbf{u}'\}$
$= \min_{\mathbf{u},\mathbf{u}'}\{w(\mathbf{v}+\mathbf{v}'): \mathbf{u} \neq \mathbf{u}'\}$
$= \min_{\mathbf{u}}\{w(\mathbf{v}): \mathbf{u} \neq \mathbf{0}\}$

b) It is assumed that **v** and **v'** have finite length and start and end in the all-zero state

c) Lowest power of X in the WEFs

d) Minimum weight of any path that diverge from the all-zero state and remerges later

- Note: We implicitly assume a non-catastrophic encoder here
- Catastrophic encoders: May have paths of smaller weight than d_{free} that do not remerge with the all-zero state

Column distance function

a) Let $[\mathbf{G}]_l$ denote the binary matrix consisting of the first $n(l+1)$ columns and $k(l+1)$ rows of $\mathbf{G}$

b) Column distance function (CDF) is denoted by d_l

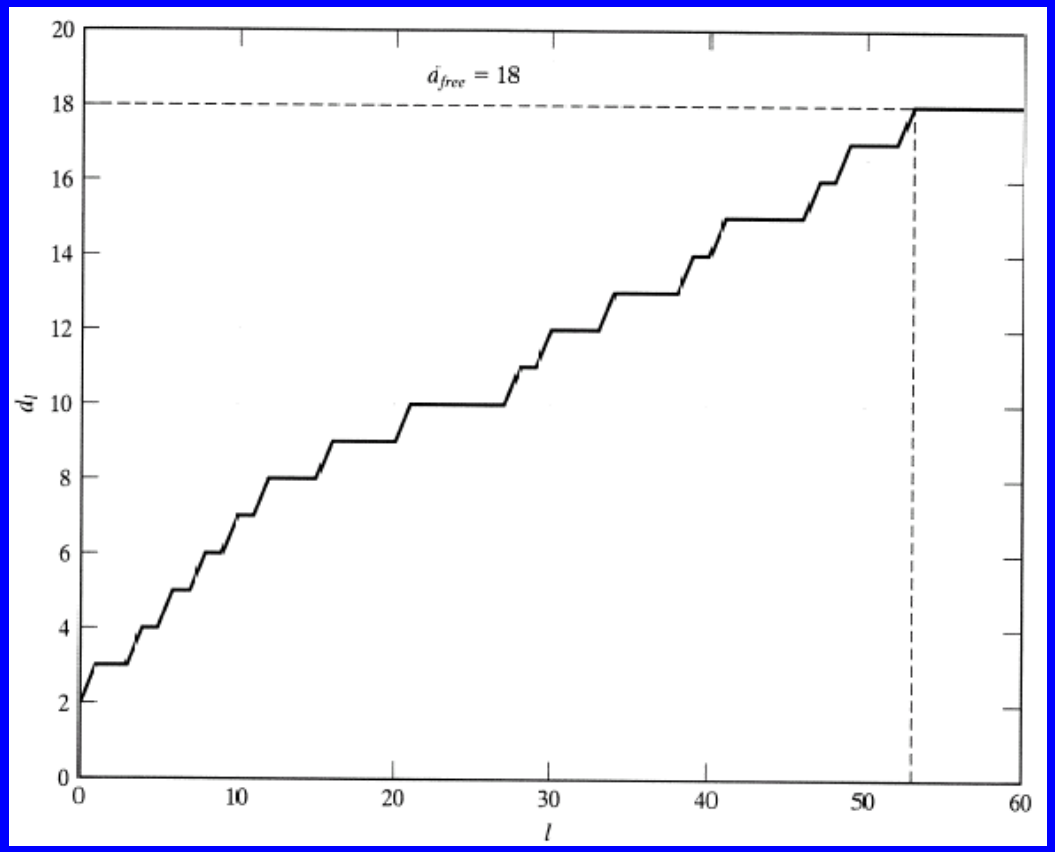
c) d_l is the minimum distance of the block code defined by $[\mathbf{G}]_l$

d) Important for sequential decoding

$$[\mathbf{G}]_l = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \cdots & \mathbf{G}_{m-1} & \mathbf{G}_m & & & & \\ & \mathbf{G}_0 & \cdots & \cdots & \mathbf{G}_{m-2} & \mathbf{G}_{m-1} & \mathbf{G}_m & & & \\ & & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & & \\ & & & \ddots & \mathbf{G}_1 & \mathbf{G}_2 & \mathbf{G}_3 & \cdots & \mathbf{G}_m & \\ & & & & \mathbf{G}_0 & \mathbf{G}_1 & \mathbf{G}_2 & \cdots & \mathbf{G}_{m-1} & \mathbf{G}_m \\ & & & & & \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{m-2} & \mathbf{G}_{m-1} \\ & & & & & & \mathbf{G}_0 & \vdots & \vdots & \vdots \\ & & & & & & & \ddots & \mathbf{G}_1 & \mathbf{G}_2 \\ & & & & & & & & \mathbf{G}_0 & \mathbf{G}_1 \\ & & & & & & & & & \mathbf{G}_0 \end{bmatrix}$$

Special cases of column distance

- If $l = m$, then d_l = minimum distance (important for majority logic decoding)
- $l \to \infty$, $d_l \to d_{\text{free}}$ for non-catastrophic encoders



Optimum decoding of CCs (CH 12)

a) A trellis offers an ”economic” representation of all codewords
b) Maximum likelihood decoding: The Viterbi algorithm
   - Decode to the nearest codeword
c) Maximum a posteriori (MAP) decoding: The BCJR algorithm
   - Minimize information bit error probability
   - Turbo decoding applications

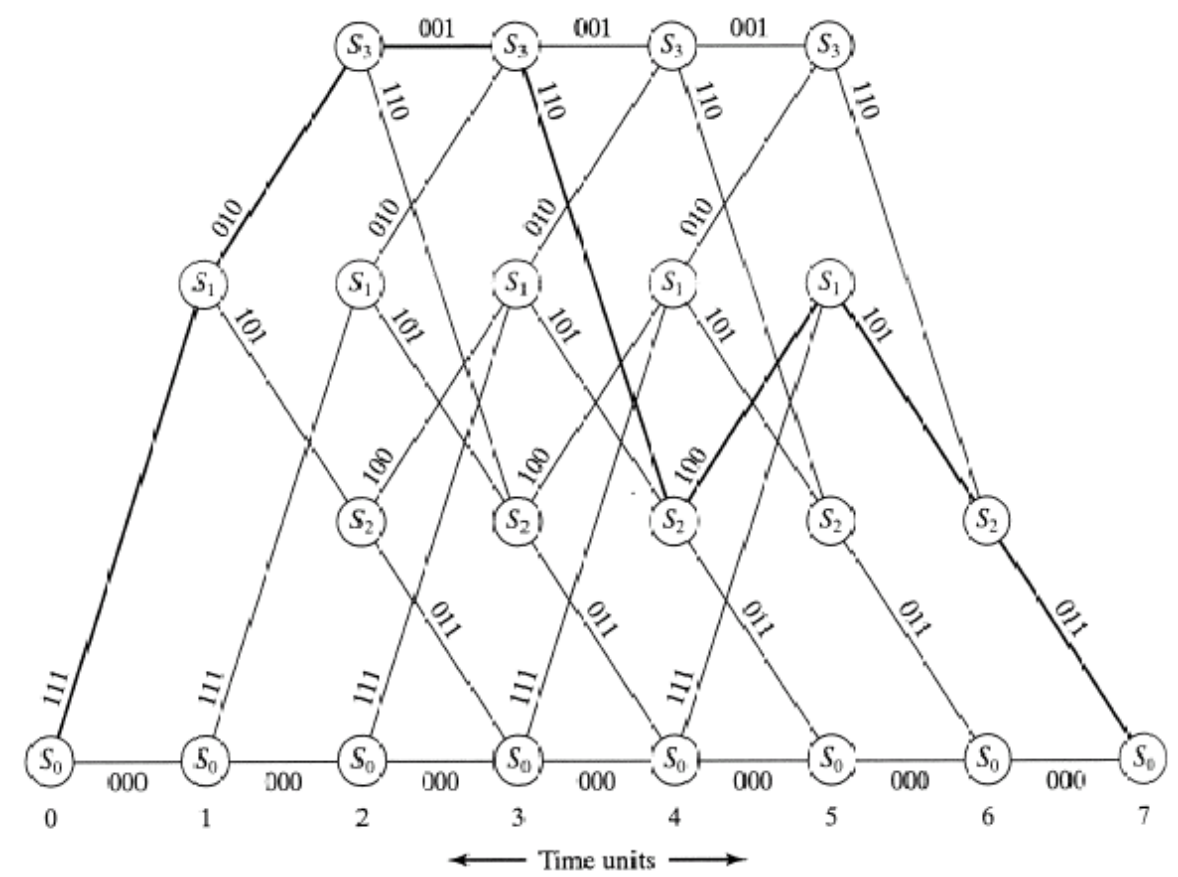
Trellises for convolutional codes

a) How to obtain the trellis from the state diagram?

- Make one copy of the set of states of a state diagram for each time instant
- Let branches from states at time instant i go to states at time instant $i+1$

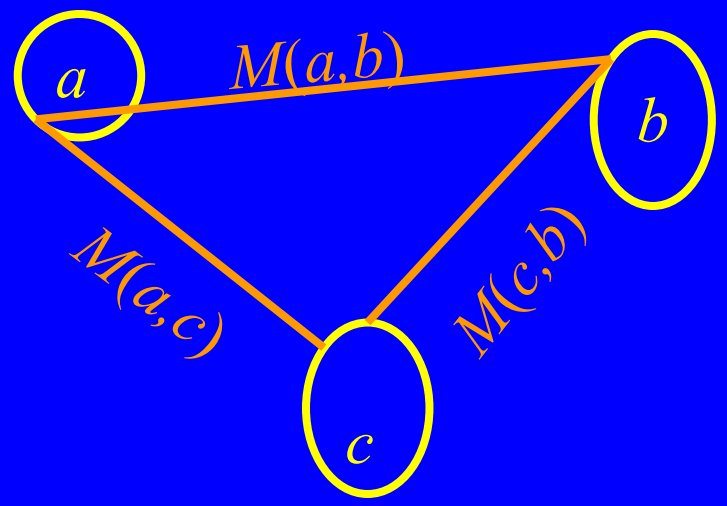
Example

$\mathbf{G}(D) = [1+D,\ 1+D^2,\ 1+D+D^2]$



Metrics

A metric is a measure of (abstract) distance between (abstract) points



that obeys the triangle inequality $M(a,b) \leq M(a,c) + M(c,b)$

Metrics for a DMC

a) Information $\mathbf{u} = (\mathbf{u}_0, \ldots, \mathbf{u}_{h-1}) = (u_0, \ldots, u_{K-1})$ $\quad K = kh$

b) Codeword $\mathbf{v} = (\mathbf{v}_0, \ldots, \mathbf{v}_{h-1}) = (v_0, \ldots, v_{N-1})$ $\quad N = n(h+m)$

c) Received $\mathbf{r} = (\mathbf{r}_0, \ldots, \mathbf{r}_{h-1}) = (r_0, \ldots, r_{N-1})$

d) Recall:

- $P(\mathbf{r}|\mathbf{v}) = \Pi_{l=0..h+m-1} P(\mathbf{r}_l|\mathbf{v}_l) = \Pi_{j=0..N-1} P(r_j|v_j)$
- ML decoder: Choose $\mathbf{v}$ to maximize this expression,
- or to maximize

$$\log P(\mathbf{r}|\mathbf{v}) = \Sigma_{l=0..h+m-1} \log P(\mathbf{r}_l|\mathbf{v}_l) = \Sigma_{j=0..N-1} \log P(r_j|v_j)$$

Path metrics: $M(\mathbf{r}|\mathbf{v}) = \log P(\mathbf{r}|\mathbf{v})$

Branch metrics: $M(\mathbf{r}_l|\mathbf{v}_l) = \log P(\mathbf{r}_l|\mathbf{v}_l)$

Bit metrics: $M(r_j|v_j) = \log P(r_j|v_j)$

Partial path metrics

a) Path metric for the first t branches of a path

b) $M([\mathbf{r}|\mathbf{v}]_t) = \sum_{l=0..t-1} M(\mathbf{r}_l|\mathbf{v}_l) = \sum_{l=0..t-1} \log P(\mathbf{r}_l|\mathbf{v}_l)$

$= \sum_{j=0..nt-1} \log P(r_j|v_j)$

The Viterbi algorithm

a) Recursive algorithm to grow the partial path metric of the best paths going through each state

b) Initialize $t = 1$. The loop of the algorithm looks like this:

1. (Add, compare, and select)
   - Add: Compute the partial path metrics for each path entering each state at time t based on the partial path metrics at time $t-1$ and the branch metrics
   - Compare: Compare all such incoming paths
   - Select: Select the (information block associated with the) best path, record its partial path metric, and put a pointer to where it came from
2. Set $t = t+1$. If $t < h+m$, repeat from 1
3. Backtracing: At time $h+m$, trace back through the pointers to obtain a winning path

Proof of ML decoding

a) **Theorem:** The final survivor $\mathbf{w}$ in the Viterbi algorithm is an ML path, that is, $M(\mathbf{r}|\mathbf{w}) \geq M(\mathbf{r}|\mathbf{v})$ for all $\mathbf{v} \in C$

b) **Proof:**

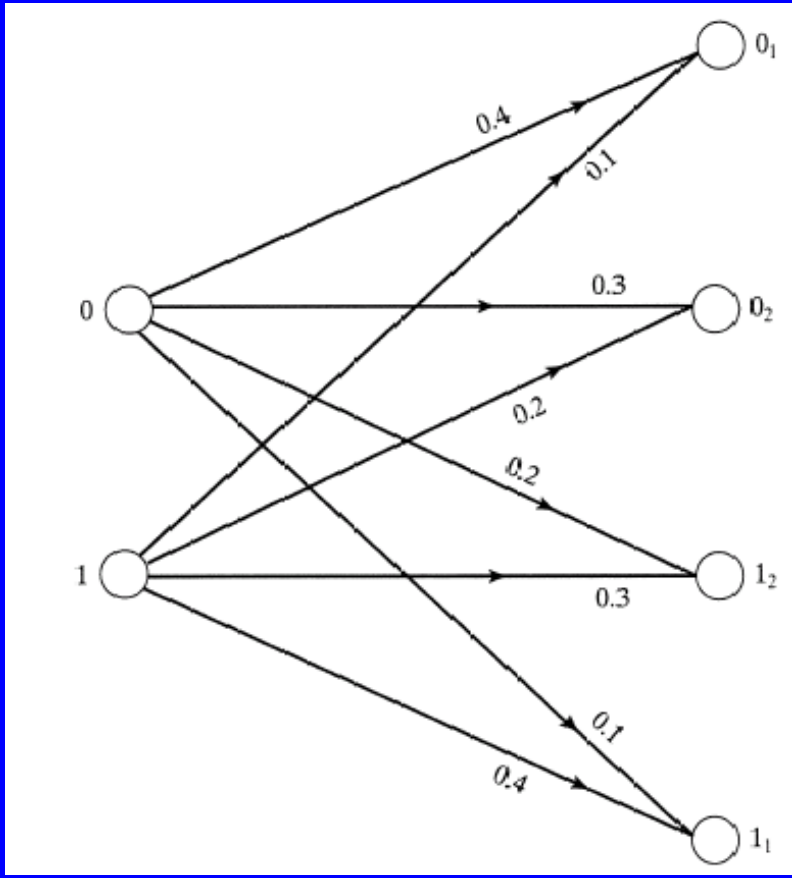
- Assume that the ML path is eliminated by the algorithm at time t
- Thus, the partial path metric of the survivor path exceeds that of the ML path at time t
- Append the remaining portion of the ML path onto the survivor path at time t
- Then, the total path metric of the survivor path will exceed the total path metric of the ML path, and we have contradiction, since the ML path (per definition) has the largest path metric

Note on implementation

- In hardware: Implementations of the Viterbi algorithm often use simple processors that either cannot process floating point numbers, or where such processing is slow
- For a DMC, the bit metrics can be represented by a finite size table
- The bit metric $M(r_j|v_j) = \log P(r_j|v_j)$ is usually a real number, but

  since the algorithm only determines the path of maximum metric, the result is not affected by scaling or by adding constants
- Thus, $M(r_j|v_j) = \log P(r_j|v_j)$ can be replaced by $c_2[\log P(r_j|v_j) + c_1]$
- Select the constants c_1 and c_2 such that all bit metrics are closely approximated by integers

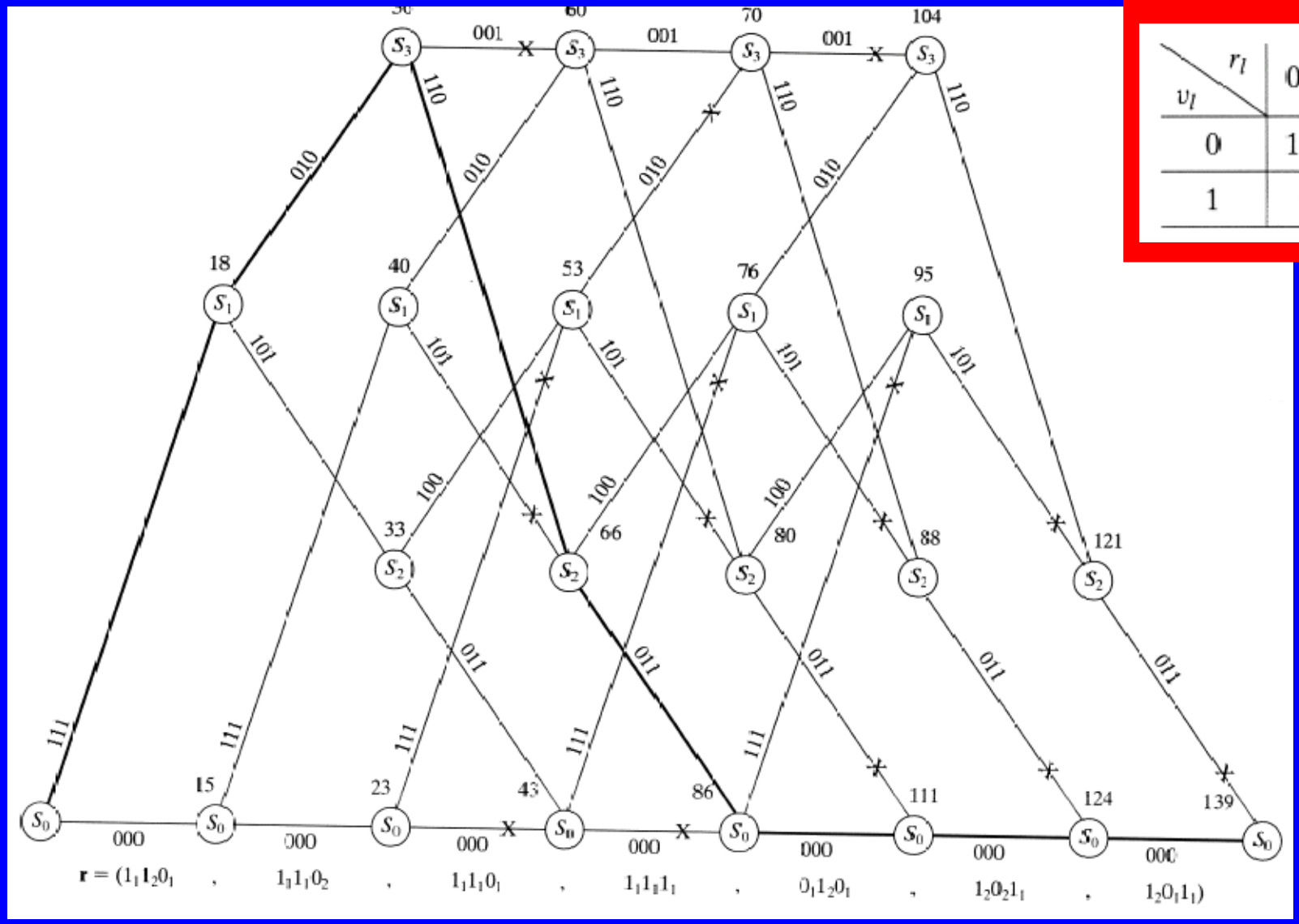
Example: 2-input, 4-output DMC



v_l \ r_l	0_1	0_2	1_2	1_1
0	−0.4	−0.52	−0.7	−1.0
1	−1.0	−0.7	−0.52	−0.4

v_l \ r_l	0_1	0_2	1_2	1_1
0	10	8	5	0
1	0	5	8	10

Example



v_l \ r_l	0_1	0_2	1_2	1_1
0	10	8	5	0
1	0	5	8	10