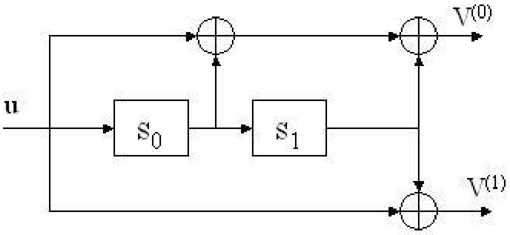
Binary Convolutional Codes

A convolutional code has memory over a short block length. This memory results in encoded output symbols that depend not only on the present input, but also on past inputs.
An (n,k,m) convolutional code is implemented using k-input, n-output linear sequential system with a shift-register having m stages. In practice k and n are small and m is large to achieve low error probabilities. In the particular case when k =1, the information sequence is not divided into blocks and can be processed continuously.
In practice, the state of the convolutional code is periodically forced to a defined state. for synchronization.

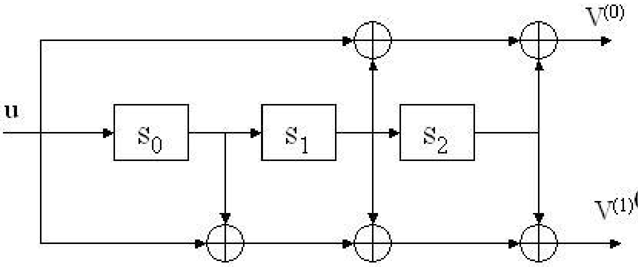
Figure 1 shows a (2,1,2) code. Figure 2 shows a (2, 1, 3) code and Figure 3 shows a (3, 2, 1) binary code. In each case a state sequence diagram can be built.
There can also be convolutional codes where the delay sequence is different for different outputs, Figure 4.
The constraint length, K, is defined as the maximum length of output sequence that can be affected by an input. In general this is given by n(m+1), where m is the maximum delay path through the system, (if the delays are not all the same for different output paths). For the codes in Figures, 1 – 4, the constraint lengths are 6, 8, 6, 9, respectively.



A (2, 1, 2) encoder

$G_o(D) = 1+D+D^2$
$G_1(D) = 1+D^2$

Figure 1

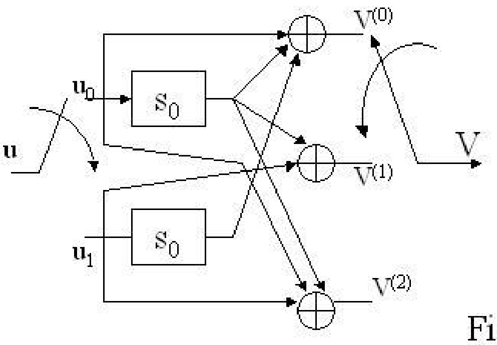


A (2, 1, 3) encoder

$G_o(D) = 1+D^2+D^3$

$G_1(D) = 1+D+D^2+D^3$

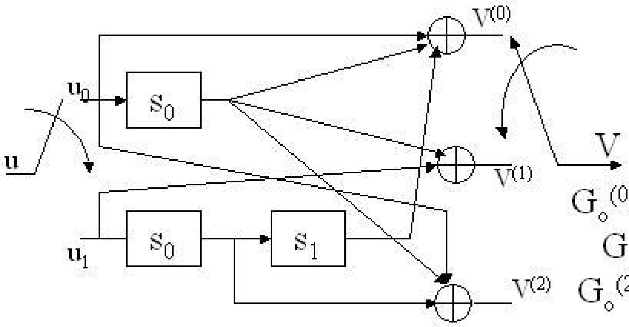
Figure 2



A (3, 2, 1) encoder

$G_o^{(0)}(D) = 1+D;\ G_1^{(0)}(D) = D$
$G_o^{(1)}(D) = D;\ G_1^{(1)}(D) = 1$
$G_o^{(2)}(D) = 1+D;\ G_1^{(2)}(D) = 1$

Figure 3



A (3, 2, 2) encoder

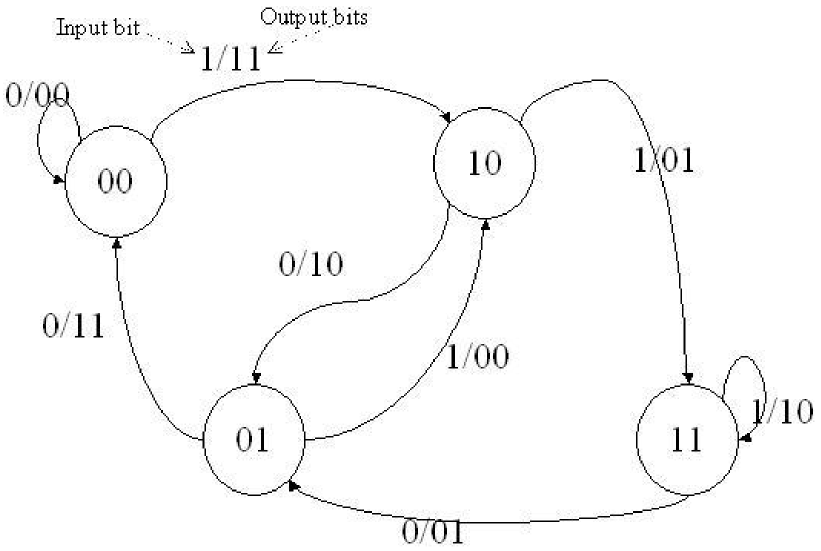
$G_o^{(0)}(D) = 1+D;\ G_1^{(0)}(D) = D^2$
$G_o^{(1)}(D) = D;\ G_1^{(1)}(D) = 1$
$G_o^{(2)}(D) = 1+D;\ G_1^{(2)}(D) = D$

Figure 4

Since a convolutional encoder generates n encoded bits for each k information bits, R = k/n is the code rate. For an information sequence that is short, k.L, the corresponding total encoded bits are n(L+m). The last n.m non zero outputs are the delay bits within the system, as they come out at the end. Therefore, if this is taken into account, the code rate becomes R = kL/n(L+m).. If L>>m, then L/(L+m)≈ 1 and R =k/n. But if this is not the case R is reduced by a fractional amount given by m/(L+m), also called the fractional rate loss.

State diagrams

Figure 5 shows the state diagram for the code of Figure 1. Table 1 gives the information of Figure 5 in tabular form. For a shift register sequence, m, there are 2^m states in the diagram.



State Diagram of a (2,1,2) convolutional encoder

Figure 5

Initial State $s_0[i]\ s_1[i]$	Information $u[i]$	Final State $s_0[i+1]\ s_1[i+1]$	Outputs $v^{(0)}[i]\ v^{(1)}[i]$
00	0	00	00
00	1	10	11
01	0	00	11
01	1	10	00
10	0	01	10
10	1	11	01
11	0	01	01
11	1	11	10

Table 1

Given a vector [u] = [101001], the resultant code for the 2,1,2) code above is
11, 10, 00, 10, 11, 11

The encoding process can also make use of a matrix G, in terms of the G_i . For tbe code of Figure 3, this is given as a semi infinite matrix, where each row is made up of

$$G = \begin{bmatrix} 11 & 01 & 11 & 11 & \cdots & & & \\ & 11 & 01 & 11 & 11 & \cdots & & \\ & & 11 & 01 & 11 & 11 & \cdots & \\ & & & 11 & 01 & 11 & 11 & \cdots \end{bmatrix}$$

the G_0 = 1+D^2+ D^3 and G_1 = 1+D+D^2+ D^3 are the elements in a row of the matrix, (not each is read from right to left). Each successive row is shifted two places, equal to the two output bits. Every empty position in the matrix is a zero. When performing the encoding [v] = [u]G, the vector is started from row0 column0 . For a [u] = [10111] and the code (2,1,3) the output is
1 1, 0 1, 0 0, 0 1, 0 1, 0 1, 0 0, 1 1.
Note that there are 8 output pairs due to the delay elements, which as explained, gives a fractional rate loss for this case of 3/8. (i.e a rate of 5/16 instead of a ½).

Weight Distribution Sequence WDS

The free distance d_f of a convolutional code is the smallest distance between any two distinct code sequences. The free distance of a convolutional code can be obtained from its weight enumerator polynomial. Initially the weight distribution of a 1/n linear block code is derived. Let Ω(x) be a 2^m X 2^m matrix, known as a state transition matrix, where $\Omega_{ij}(x) = \delta_{ij}\, x^{h_{ij}}$ and δ_{ij} =1 if there is a transition from state i to state j, and h_{ij} is the Hamming Weight of the corresponding output vector of length n.
For the convolutional encoder with a state diagram of Figure 5, the state transition matrix is

$$\Omega(x) = \begin{bmatrix} 1 & 0 & x^2 & 0 \\ x^2 & 0 & 1 & 0 \\ 0 & x & 0 & x \\ 0 & x & 0 & x \end{bmatrix}$$

Consider a (2,1,2) code, using message vectors of 3 bits, and allowing the encoder to pass out the delay elements, by inserting two zeroes between each 3-bit vector. This results in a d_f=5. and a code rate of kL/n(L+m) = 3/10 = 0.3 instead of 3/6 = 0.5. Using Table 1, the weight distribution sequence, WDS, is found, from Table 2, to be

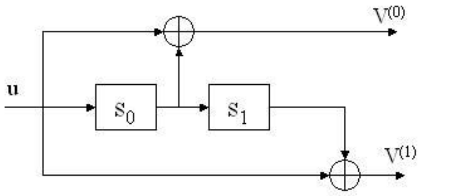
k' = (L+m)	n(L+m)	Weight
000 00	00 00 00 00 00	0
001 00	00 00 11 10 11	5
010 00	00 11 10 11 00	5
011 00	00 11 01 01 11	6
100 00	11 10 11 00 00	5
101 00	11 10 00 10 11	5
110 00	11 01 01 11 00	6
111 00	11 01 10 01 11	7

Table 2

WDS ≡ A(x) = 1 + $3x^5$ + $3x^6$ + x^7 . Note that the minimum distance is 5.

Catastrophic Code

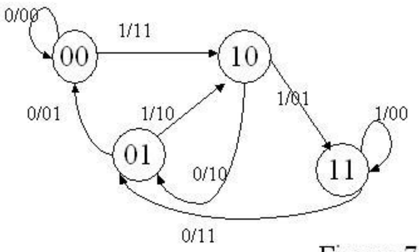
A convolutional encoder is said to be catastrophic if a finite number of channel errors produce an infinite number of errors after decoding. It is characterized by having, in the state diagram, a self loop of zero weight, other than that around the state S_{00}. This is illustrated, by modifying the (2,1,2) code of Figure 1, to Figure 6, and resultant state diagram Figure 7.



A (2, 1, 2) encoder

$G_o(D) = 1+D$
$G_1(D) = 1+D^2$

Figure 6



State diagram
for Figure 6

Figure 7

Maximum Likelihood Decoding

The likelihood of a received sequence [R] over a noisy memory less channel, given that [v] was sent using a BSC with an error bit probability p, is given by

$$p(r\,|\,v) = \prod_{i=0}^{n-1}(1-p)\left(\frac{p}{1-p}\right)^{d_H(r_i,v_i)}$$ with $d_H(r_i,v_i) = 1$ if $r_i, \neq v_i$, and $d_H(r_i,v_i) = 0$ if $r_i, = v_i$.

If there are no errors, the above results in $p(v) = (1-p)^n$, which is the probability of receiving every bit correctly. Using logs, the log likelihood function becomes

$$\log P(r\,|\,v) = d([r]\,|\,[v])\log\frac{p}{1-p} + N\log(1-p)$$ where d([r]||[v]) is the Hamming distance between received word [r] and decoded codeword [v]. Since log[p/(1-p)]<0 and Nlog(1-p) is a constant for all [v], the MLD for a BSC is the codeword that minimizes d([r],[v]).

The most widely used decoding algorithm is the Viterbi algorithm based on a trellis decoder. This algorithm chooses the best path at stage[i] and works back along the trellis the resultant best path for stage [i]. Viterbi in his algorithm shows that for errors to be corrected the trellis depth from the error stage to the current stage, for a memory m, rate ½ convolutional code, should have a received sequence, of length ℓ such that ℓ > 5m. This is the minimum decoding depth for proper operation.

The algorithm uses the following steps

(i) Initialise the trellis to i=0 and each metric $S_0^{(k)} = 0$, where the metric expresses the distance between the received word [r] and the Viterbi generated word S_i at the ith stage and k is the trellis state, $0 < k < 2^m - 1$.

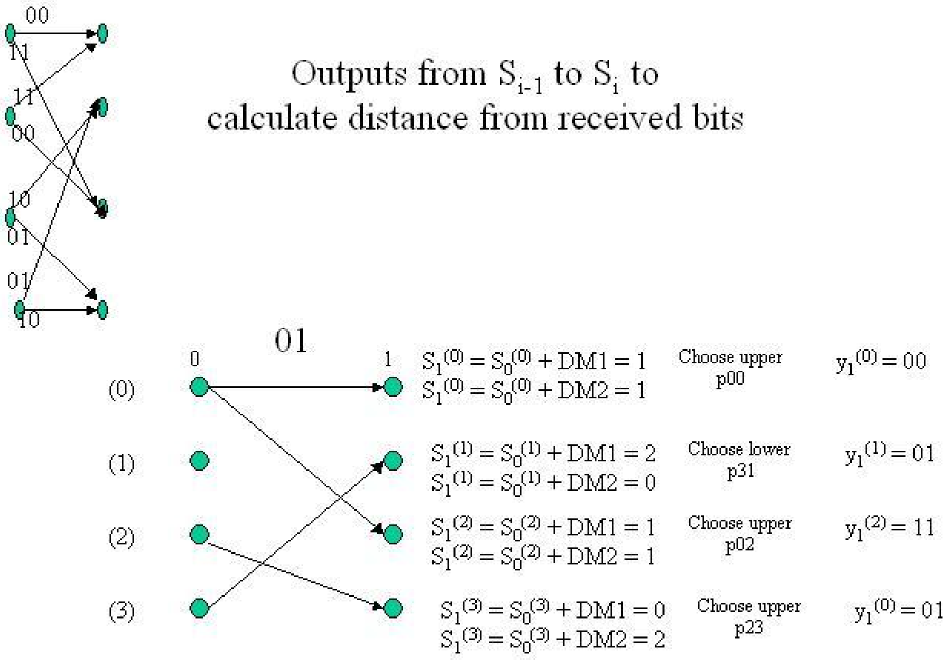
(ii) At a stage i, compute the distance between the received n-tuple, and the value from the trellis precursor states $S_{i-1}^{(k1)}$ and $S_{i-1}^{(k2)}$ to the current trellis node $S_i^{(k)}$

(iii) Choose the minimum of the n inputs from the previous stage. In case of a tie decide randomly on one of the least equal values

(iv) Work out, for each present node $y_i^{(k)}$ the survivor path backwards, by moving backwards along the path indicated as optimal from the previous node backward.

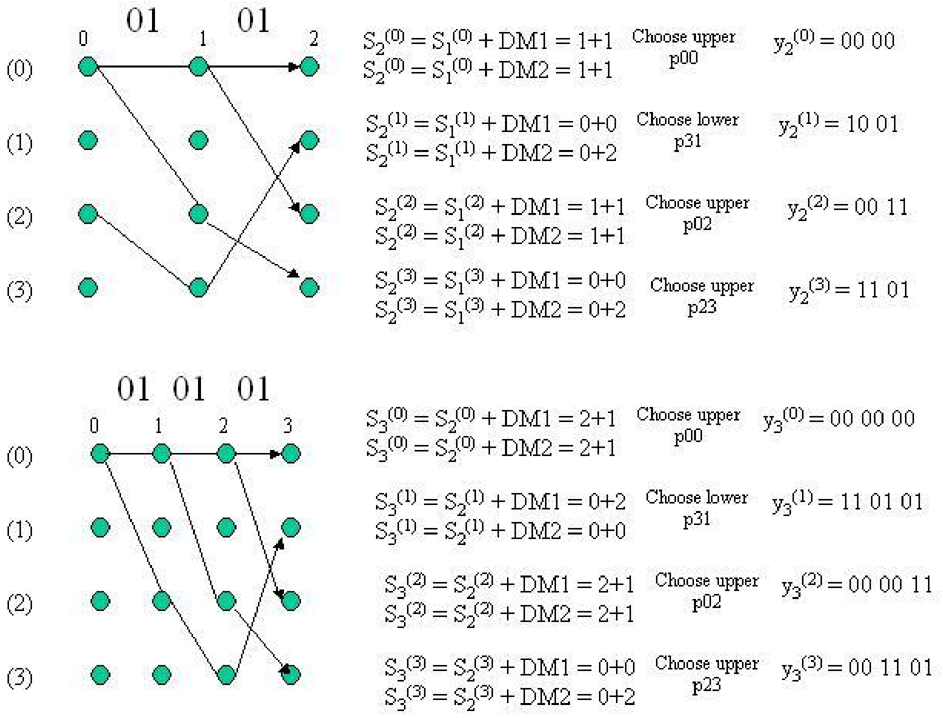
(v) This is repeated until at least $\ell > 5m$

An example is worked to demonstrate the operation. Assume the (2,1,2) code of Figure 1 and Table 1. Assume that the code vector sent is 11 01 01 00 10 11 and the received n-tuple is 01 01 01 00 10 11, having a bit in error in the first pair.



00
11
11
00
10
01
01
10
Outputs from S_{i-1} to S_i to calculate distance from received bits
0
01
1
(0)
(1)
(2)
(3)
$S_1^{(0)} = S_0^{(0)} + DM1 = 1$
$S_1^{(0)} = S_0^{(0)} + DM2 = 1$
Choose upper p00
$y_1^{(0)} = 00$
$S_1^{(1)} = S_0^{(1)} + DM1 = 2$
$S_1^{(1)} = S_0^{(1)} + DM2 = 0$
Choose lower p31
$y_1^{(1)} = 01$
$S_1^{(2)} = S_0^{(2)} + DM1 = 1$
$S_1^{(2)} = S_0^{(2)} + DM2 = 1$
Choose upper p02
$y_1^{(2)} = 11$
$S_1^{(3)} = S_0^{(3)} + DM1 = 0$
$S_1^{(3)} = S_0^{(3)} + DM2 = 2$
Choose upper p23
$y_1^{(0)} = 01$

Figure 8



$S_2^{(0)} = S_1^{(0)} + DM1 = 1+1$ Choose upper $y_2^{(0)} = 00\ 00$
$S_2^{(0)} = S_1^{(0)} + DM2 = 1+1$ p00

$S_2^{(1)} = S_1^{(1)} + DM1 = 0+0$ Choose lower $y_2^{(1)} = 10\ 01$
$S_2^{(1)} = S_1^{(1)} + DM2 = 0+2$ p31

$S_2^{(2)} = S_1^{(2)} + DM1 = 1+1$ Choose upper $y_2^{(2)} = 00\ 11$
$S_2^{(2)} = S_1^{(2)} + DM2 = 1+1$ p02

$S_2^{(3)} = S_1^{(3)} + DM1 = 0+0$ Choose upper $y_2^{(3)} = 11\ 01$
$S_2^{(3)} = S_1^{(3)} + DM2 = 0+2$ p23

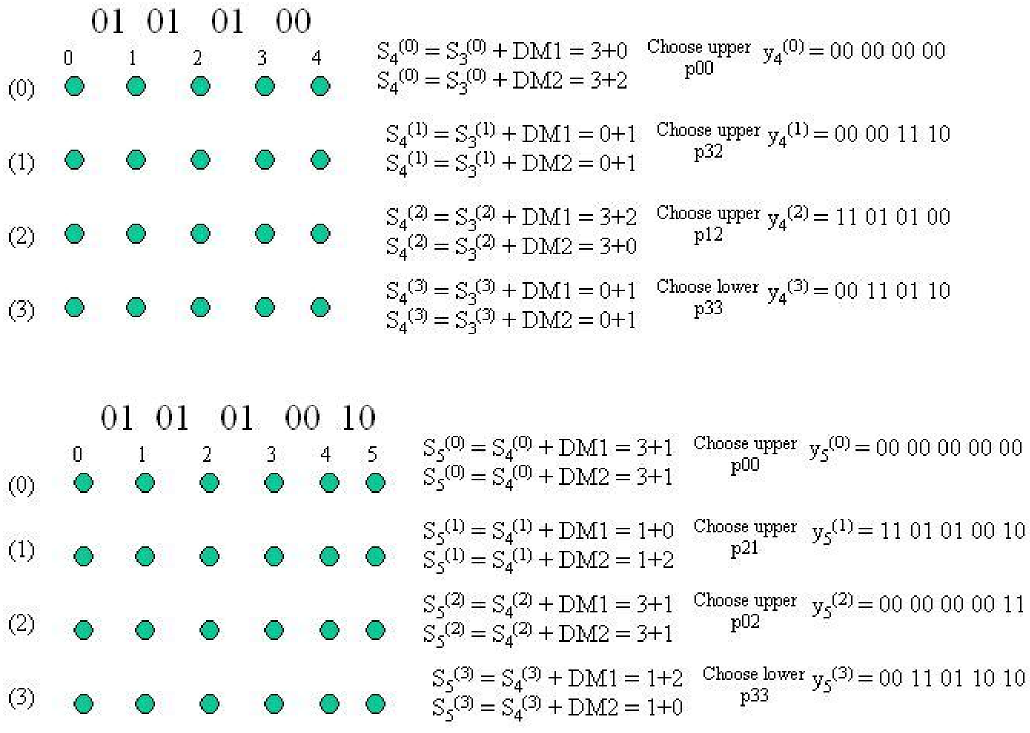
$S_3^{(0)} = S_2^{(0)} + DM1 = 2+1$ Choose upper $y_3^{(0)} = 00\ 00\ 00$
$S_3^{(0)} = S_2^{(0)} + DM2 = 2+1$ p00

$S_3^{(1)} = S_2^{(1)} + DM1 = 0+2$ Choose lower $y_3^{(1)} = 11\ 01\ 01$
$S_3^{(1)} = S_2^{(1)} + DM2 = 0+0$ p31

$S_3^{(2)} = S_2^{(2)} + DM1 = 2+1$ Choose upper $y_3^{(2)} = 00\ 00\ 11$
$S_3^{(2)} = S_2^{(2)} + DM2 = 2+1$ p02

$S_3^{(3)} = S_2^{(3)} + DM1 = 0+0$ Choose upper $y_3^{(3)} = 00\ 11\ 01$
$S_3^{(3)} = S_2^{(3)} + DM2 = 0+2$ p23

Figure 9



$S_4^{(0)} = S_3^{(0)} + DM1 = 3+0$ Choose upper $y_4^{(0)} = 00\ 00\ 00\ 00$
$S_4^{(0)} = S_3^{(0)} + DM2 = 3+2$ p00

$S_4^{(1)} = S_3^{(1)} + DM1 = 0+1$ Choose upper $y_4^{(1)} = 00\ 00\ 11\ 10$
$S_4^{(1)} = S_3^{(1)} + DM2 = 0+1$ p32

$S_4^{(2)} = S_3^{(2)} + DM1 = 3+2$ Choose upper $y_4^{(2)} = 11\ 01\ 01\ 00$
$S_4^{(2)} = S_3^{(2)} + DM2 = 3+0$ p12

$S_4^{(3)} = S_3^{(3)} + DM1 = 0+1$ Choose lower $y_4^{(3)} = 00\ 11\ 01\ 10$
$S_4^{(3)} = S_3^{(3)} + DM2 = 0+1$ p33

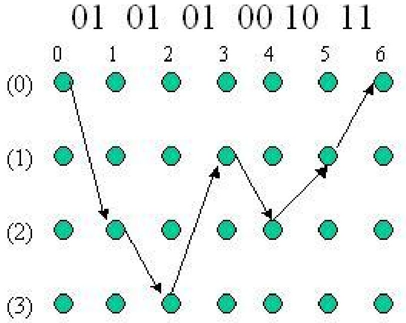
$S_5^{(0)} = S_4^{(0)} + DM1 = 3+1$ Choose upper $y_5^{(0)} = 00\ 00\ 00\ 00\ 00$
$S_5^{(0)} = S_4^{(0)} + DM2 = 3+1$ p00

$S_5^{(1)} = S_4^{(1)} + DM1 = 1+0$ Choose upper $y_5^{(1)} = 11\ 01\ 01\ 00\ 10$
$S_5^{(1)} = S_4^{(1)} + DM2 = 1+2$ p21

$S_5^{(2)} = S_4^{(2)} + DM1 = 3+1$ Choose upper $y_5^{(2)} = 00\ 00\ 00\ 00\ 11$
$S_5^{(2)} = S_4^{(2)} + DM2 = 3+1$ p02

$S_5^{(3)} = S_4^{(3)} + DM1 = 1+2$ Choose lower $y_5^{(3)} = 00\ 11\ 01\ 10\ 10$
$S_5^{(3)} = S_4^{(3)} + DM2 = 1+0$ p33

Figure 10



$S_6^{(0)} = S_5^{(0)} + DM1 = 4+0$ Choose lower p10 $y_6^{(0)} = 11\ 01\ 01\ 00\ 10\ 11$
$S_6^{(0)} = S_5^{(0)} + DM2 = 4+1$

$S_6^{(1)} = S_5^{(1)} + DM1 = 1+1$ Choose upper p21 $y_6^{(1)} = 00\ 00\ 00\ 00\ 11\ 10$
$S_6^{(1)} = S_5^{(1)} + DM2 = 1+1$

$S_6^{(2)} = S_5^{(2)} + DM1 = 4+0$ Choose lower p02 $y_6^{(2)} = 00\ 10\ 00\ 00\ 00\ 11$
$S_6^{(2)} = S_5^{(2)} + DM2 = 4+2$

$S_6^{(3)} = S_5^{(3)} + DM1 = 1+1$ Choose lower p33 $y_6^{(3)} = 00\ 11\ 01\ 10\ 10\ 10$
$S_6^{(3)} = S_5^{(3)} + DM2 = 1+1$

Comparing $y_6^{(0)}, y_6^{(1)}, y_6^{(2)}, y_6^{(3)}$ to [r]
$S_6^{(0)}= 1$; $S_6^{(1)}= 5$; $S_6^{(2)}= 5$; $S_6^{(3)}= 5$; Choose $S_6^{(0)}$

Work back the path $S_6^{(0)} -> S_5^{(1)} -> S_4^{(2)} -> S_3^{(1)} -> S_2^{(3)} -> S_1^{(2)} -> S_0^{(0)}$

Data sent worked out by reversing and starting from $S_0^{(0)}$

Figure 11

In practice various techniques are used to handle the Viterbi decoding on a computer. These include parameter normalization due to the fixed range within a computer, using a threshold as a basis to then subtract a value T, from every metric. There is also the way the path memory is kept, to be able to extract the information bits. A traceback memory using decision values that indicate state transitions are kept to reconstruct the sequence of states in reverse order. This is used when the code is implemented in hardware.

Synchronisation

If the n-tuples fall out of synchronization, the trellis starts giving continuous erroneous results. This can be checked for, using expected statistics of BER and path metric growth. This monitor is external to the decoder. This is achieved using a synchronization stage whose function is to advance the reference sequence [v] in the decoder, by skipping received symbols (a maximum of n-1) one at a time until the synchronization variables indicate normal behaviour.
Alternatively, the data is broken up into fixed length (eg a few thousand bits). Then a known unique word is added to synchronize the receiver and forces the convolutional encoder to return to a known state.

Punctured Convolutional Codes
Puncturing is the process of systematically deleting some of the generated bits by the encoder. Since the trellis is the same, the number of information bits is the same. However this puncturing gives rise to a higher rate encoder than the original.
The basis of puncturing is a matrix, called a Puncturing matrix, which defines the operation.

Example A memory 3/3 convolutional code can be constructed by puncturing the output bits of the (2,1,2) encoder using the puncturing matrix,

$P = \begin{bmatrix} 11 \\ 10 \end{bmatrix}$. In this case if the originally generated bits, see Figure 6, were $V^{(0)}$and $V^{(1)}$.

Let $V^{(0)}$ be [100101..] and $V^{(1)}$ be [111001..], then the resulting output is v=[11 01 01 10 00 11 …]. Interpreting $V_p^{(1)}$ as [1X1X0X…] the transferred bits after puncturing are [11, 0, 01, 1, 00, 1 …]. This gives rise to a rate 2/3 code instead of the original ½.

The process of decoding follows the same lines as the original decoding. In dealing with the unknown bit, when calculating the distance DM, see Figures 9 to 12, only the valid known bit is considered. The rest of the decoding algorithm is the same. However the decoding depth L must be increased as more output bits are punctured.