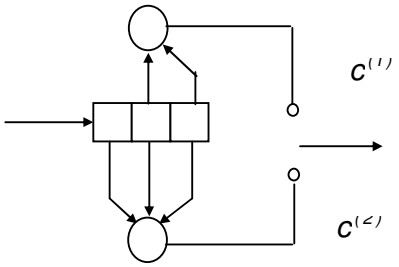
Concatenated codes:

7.1) The output codeword of a block code $C_b(6,3)$ generated by the generator matrix $\boldsymbol{G}$ is then input to a convolutional encoder like that seen in the following figure, operating in pseudo-block form. This means that after inputting the 6 bits of the codeword of the block code, then two additional zeros (tailing bits) are also input to clear the registers of the convolutional encoder.

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$



The transpose of the parity check matrix of the block code is obtained as follows:

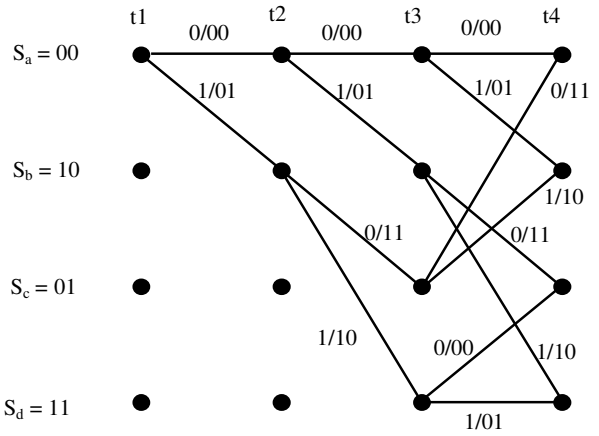
$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{P} & \boldsymbol{I}_k \end{bmatrix},\ \boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_{n-k} & \boldsymbol{P}^T \end{bmatrix},\ \boldsymbol{H}^T = \begin{bmatrix} \boldsymbol{I}_{n-k} \\ \boldsymbol{P} \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix},\ \boldsymbol{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

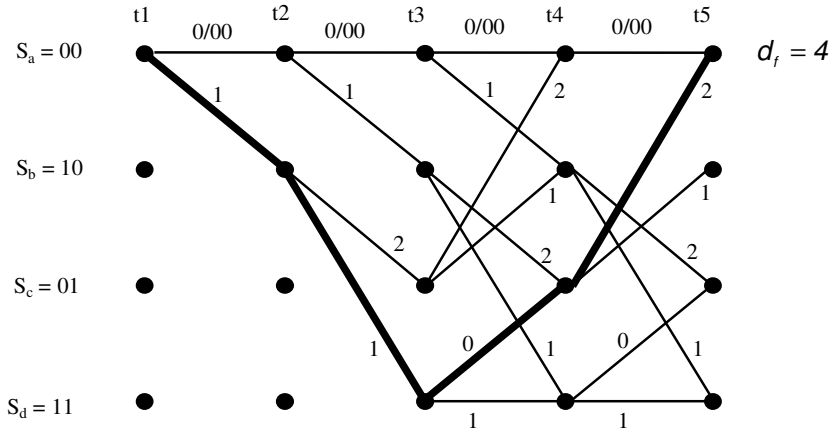
The syndrome-error pattern table is:

e							S		
1	0	0	0	0	0		1	0	0
0	1	0	0	0	0		0	1	0
0	0	1	0	0	0		0	0	1
0	0	0	1	0	0		1	1	0
0	0	0	0	1	0		0	1	1
0	0	0	0	0	1		1	0	1

The trellis of the convolutional encoder seen in the above figure is:



And the minimum Hamming free distance of the convolutional code is equal to $d_f = 4$:

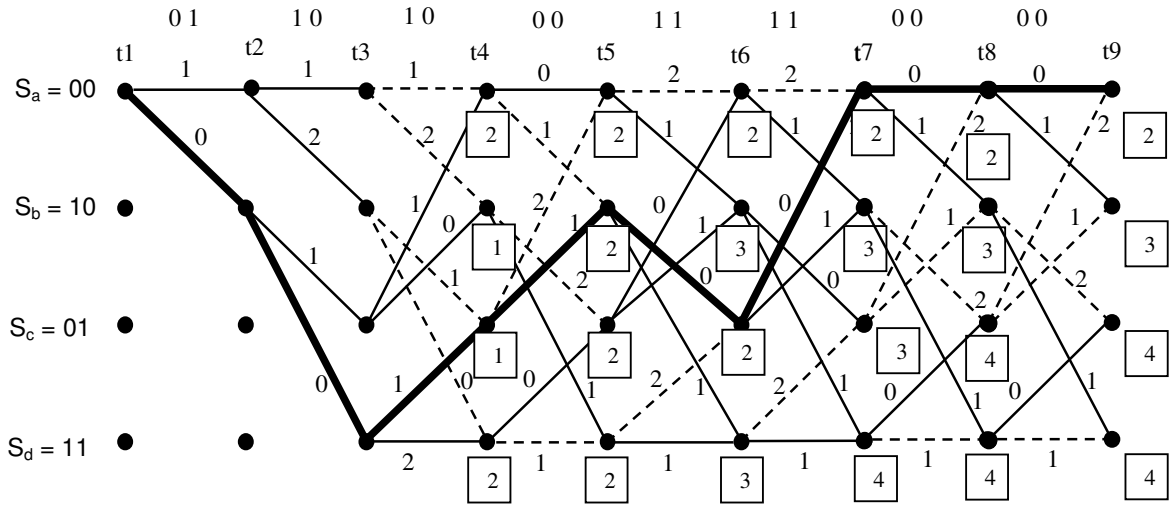


The following is the code table of the concatenated code:

m	c	w										w
000	000000	0		00	00	00	00	00	00	00	00	0
001	101001	3		01	11	10	11	11	01	11	11	13
010	011010	3		00	01	10	00	10	11	11	00	7
011	110011	4		01	10	00	11	01	10	00	11	8
100	110100	3		01	10	00	10	11	11	00	00	7
101	011101	4		00	01	10	01	00	10	11	11	8
110	101110	4		01	11	10	10	01	00	11	00	8
111	000111	3		00	00	00	01	10	01	00	11	5

The minimum Hamming distance of the concatenated code is $d_{min} = 5$.

The received sequence $r = (01\,10\,10\,00\,11\,11\,00\,00)$ is first decoded using the convolutional code with the classic Viterbi algorithm. Minimum accumulated Hamming distance values at each state are inside the squares.



Even before the last two steps in the Viterbi decoding, the decoded sequence is seen to be $d = (0110\ 00\ 10\ 1111\ 00\ 00)$, which corresponds to the message sequence $m' = (110100)$. The tailing zeros are discarded.

This decoded sequence is a codeword of the block code, so that the syndrome calculation in this second step of the concatenated code is equal to zero, and the decoded message is directly determined by truncating the decoded vector. Thus, $m = (100)$.

Note that the minimum distance of the block code on its own is 3. This would also be the minimum distance of the concatenated code without the extra tailing process, which adds at least 2 to the weights of the codewords of the block code. This indicates the importance of properly tailing off when using a convolutional code.

7.2)

The cyclic code $C_{cyc}(3,1)$ generated by the polynomial $g(X) = 1 + X + X^2$ is a repetition code. Its code table is the following:

m	c(X)	c	w
1	$1+X+X^2$	111	3
0	0	000	-

Its minimum Hamming distance is $d_{min\_(3,1)} = 3$.

The cyclic code $C_{cyc}(7,3)$, generated by the polynomial $g(X) = 1 + X + X^2 + X^4$ has the following code table:

m	c	w
000	0 0 0 0 0 0 0	-
001	1 1 0 1 0 0 1	4
010	0 1 1 1 0 1 0	4
011	1 0 1 0 0 1 1	4
100	1 1 1 0 1 0 0	4
101	0 0 1 1 1 0 1	4
110	1 0 0 1 1 1 0	4
111	0 1 0 0 1 1 1	4

Its minimum Hamming distance is $d_{min\_(7,3)} = 4$.

Since $g(X) = 1 + X + X^2 + X^4$ and $n - k = 7 - 3 = 4$, $X^{n-k} = X^4$. As an example, we calculate one of the code polynomials as follows:

$m(X) = X^2$;

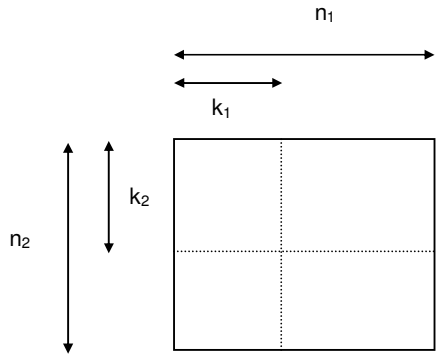
$$
\begin{array}{ll}
X^6 & / X^4 + X^2 + X + 1 \\
X^6 + X^4 + X^3 + X^2 & X^2 + 1 \\
\text{---------- } & \\
X^4 + X^3 + X^2 & \\
X^4 + X^2 + X + 1 & \\
\text{---------- } & \\
X^3 + X + 1 = p(X) &
\end{array}
$$

Then:

$c(X) = X^4 m(X) + p(X) = X^6 + X^3 + X + 1 \Rightarrow c = (1101001)$

The remaining code polynomials can be determined in the same way.

The array code is constructed as indicated in the following figure:



where $k_1 = 1$, $n_1 = 3$, $k_2 = 3$ and $n_2 = 7$.

The non-zero codewords of this concatenated code are now determined. Message bits are in bold:

1	1	1		**0**	0	0		**1**	1	1		**0**	0	0		**1**	1	1		**0**	0	0		**1**	1	1
0	0	0		**1**	1	1		**1**	1	1		**0**	0	0		**0**	0	0		**1**	1	1		**1**	1	1
0	0	0		**0**	0	0		**0**	0	0		**1**	1	1		**1**	1	1		**1**	1	1		**1**	1	1
1	1	1		1	1	1		0	0	0		0	0	0		1	1	1		1	1	1		0	0	0
0	0	0		1	1	1		1	1	1		1	1	1		1	1	1		0	0	0		0	0	0
1	1	1		1	1	1		0	0	0		1	1	1		0	0	0		0	0	0		1	1	1
1	1	1		0	0	0		1	1	1		1	1	1		0	0	0		1	1	1		0	0	0

The minimum weight in this code table is $w_{min} = d_{min} = 12$, so the array code can correct up to 5 errors and detect up to 6 errors. This minimum Hamming distance is such that:

$$d_{conc\_min} = d_{(3,1)\_min} x d_{(7,3)\_min} = 3x4 = 12$$

which confirms the product code nature of this type of array code.

Since the 3 message bits generate 21 coded bits, then the rate of the concatenated code is $R_c = 1/7$, which can also be determined by multiplying the rates of the two component codes: (1/3)x(3/7) = 3/21 = 1/7.

If we concatenate these two codes by applying first the cyclic code $C_{cyc}(7,3)$ and then the cyclic code $C_{cyc}(3,1)$, then in the above figure $k_1 = 3$, $n_1 = 7$, $k_2 = 1$ and $n_2 = 3$. The resulting array code is described in the following table, where m is the message bits, followed by the row constituent codeword and then the column constituent codewords corresponding to each of the row bits, and the last column in the table is the weight of each array codeword:

m	$C_{cycl}(7,3)$	$C_{cycl}(3,1)$								w
000	0000000		000	000	000	000	000	000	000	12
001	1101001		111	111	000	111	000	000	111	12
010	0111010		000	111	111	111	000	111	000	12
011	1010011		111	000	111	000	000	111	111	12
100	1110100		111	111	111	000	111	000	000	12
101	0011101		000	000	111	111	111	000	111	12
110	1001110		111	000	000	111	111	111	000	12
111	0100111		000	111	000	000	111	111	111	12

The minimum Hamming distance of the concatenated code is again 12, and as before is the product of the minimum Hamming distances of the constituent codes $d_{conc\_min} = d_{(3,1)\_min} x d_{(7,3)\_min} = 3x4 = 12$. Also the rate is again 1/7, the product of the constituent code rates. This confirms that the order of concatenation is unimportant in the case of this type of array or product code.

Turbo codes:

7.3)

A simple binary array code (or punctured product code) has codewords with block length $n = 8$ and $k = 4$ information bits, in the format seen in the following figure:

1	*2*	*5*
3	*4*	*6*
7	*8*	

There are 2^4 = 16 codewords in this code.

The 15 non-zero codewords are determined as follows. In each sub-table, the message bits are in bold:

0	**0**	0		**0**	**0**	0		**0**	**0**	0		**0**	**1**	1		**0**	**1**	1		**0**	**1**	1	
0	**1**	1		**1**	**0**	1		**1**	**1**	0		**0**	**0**	0		**0**	**1**	1		**1**	**0**	1	
0	1			1	0			1	1			0	1			0	0			1	1		

0	**1**	1		**1**	**0**	1		**1**	**0**	1		**1**	**0**	1		**1**	**0**	1		**1**	**1**	0	
1	**1**	0		**0**	**0**	0		**0**	**1**	1		**1**	**0**	1		**1**	**1**	0		**0**	**0**	0	
1	0			1	0			1	1			0	0			0	1			1	1		

1	**1**	0		**1**	**1**	0		**1**	**1**	0	
0	**1**	1		**1**	**0**	1		**1**	**1**	0	
1	0			0	1			0	0		

The code has rate $R_c = 4/8 = 1/2$.

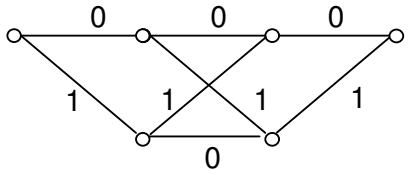
The minimum Hamming distance is the minimum value of the Hamming weight calculated over all the non-zero codewords, so this array code has a minimum Hamming distance $d_{min} = 3$. Note that the 3-bit row and column single-parity-check sub-codes generate independent parity checks. This property can be used to simplify the encoding and decoding trellises of the array code, as indicated below.

If the missing 9th bit (the check on checks) had been present, then the Hamming distance of the code would have been 2x2 = 4, but deleting one bit from a code with even Hamming distance always reduces the minimum distance by one, in this case to 3, thus confirming the above result.

The table below lists the codewords in the code:

1	***2***	***3***	***4***		***5***	***6***		***7***	***8***
0	0	0	0		0	0		0	0
0	0	0	1		0	1		0	1
0	0	1	0		0	1		1	0
0	0	1	1		0	0		1	1
0	1	0	0		1	0		0	1
0	1	0	1		1	1		0	0
0	1	1	0		1	1		1	1
0	1	1	1		1	0		1	0
1	0	0	0		1	0		1	0
1	0	0	1		1	1		1	1
1	0	1	0		1	1		0	0
1	0	1	1		1	0		0	1
1	1	0	0		0	0		1	1
1	1	0	1		0	1		1	0
1	1	1	0		0	1		0	1
1	1	1	1		0	0		0	0

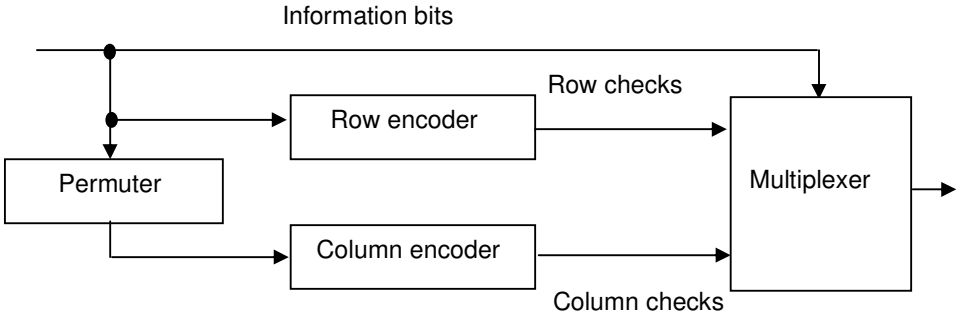
By re-ordering the code bits of the row sub-codes as 125, 346, then their trellis has the following form:



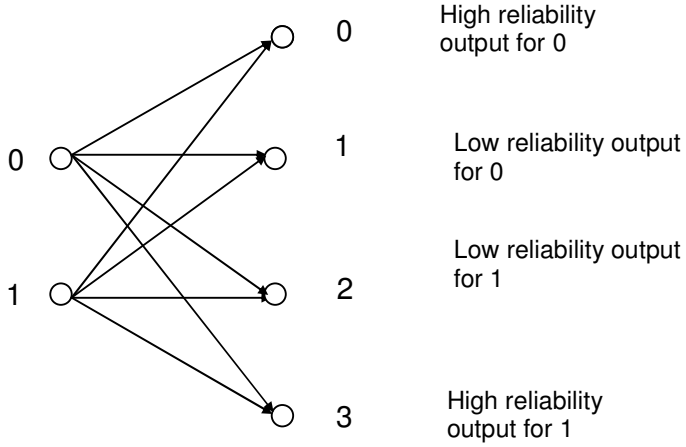
This trellis corresponds to the simple parity check code, whose table is seen below for the case of the row sub-code 125:

1	**2**	**5**
0	0	0
0	1	1
1	0	1
1	1	0

This array code can be regarded as a simple form of turbo code. In terms of the turbo code structure shown below in the figure, the parallel concatenated component encoders calculate the row and column parity checks of the array code, and the permuter alters the order in which the information bits enter the column encoder from {1,2,3,4} to {1,3,2,4}. The multiplexer then collects the information and parity bits to form a complete codeword.



A codeword from the code is modulated, transmitted over a soft-decision discrete symmetric memoryless channel like that seen in the following figure, with the conditional probabilities described in the following table, and received as the vector $\boldsymbol{r} = (10300000)$. Using the turbo (iterative) MAP decoding algorithm, determine the information bits that were transmitted.



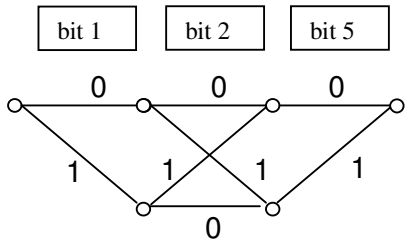
$P(y/x)$				
x , y	0	1	2	3
0	0.4	0.3	0.2	0.1
1	0.1	0.2	0.3	0.4

We assume that the transmitted code vector is $\boldsymbol{c} = (00000000)$. The received vector is $\boldsymbol{r} = (10300000)$. Note that elements of the transmitted vector (which we need to determine) are inputs of the channel of the above figure, and elements of the received vector are outputs of that channel. The following table shows the transition probabilities for the input elements '1' and '0' for each received element:

j	1	2	3	4	5	6	7	8
$(P(y_j/0), P(y_j/1))$	(0.3,0.2)	(0.4,0.1)	(0.1,0.4)	(0.4,0.1)	(0.4,0.1)	(0.4,0.1)	(0.4,0.1)	(0.4,0.1)

In the row code, bits 1, 2 and 5 are related as in the trellis seen in the figure. The same happens to bits 3, 4 and 6, which constitute a parity check code that is independent

from the code of bits 1, 2 and 5. We can calculate the values $\gamma_i(u',u)$ for each of these two codes. Coefficient for the row code of bits 1, 2 and 5 will be denoted with a superindex $\gamma_i^{RA}(u',u)$, whereas coefficients of the row code of bits 3, 4 and 6 will be denoted as $\gamma_i^{RB}(u',u)$. The trellis for the sub-code 125 is then:



i=1:

$$\gamma_1^{RA}(0,0) = \sum_{x \in A_x} P(S_1 = 0 / S_0 = 0).P(X_1 = x / \{S_0 = 0, S_1 = 0\}).P(Y_1 / X_1 = x)$$
$$= P(S_1 = 0 / S_0 = 0).P(X_1 = 0 / \{S_0 = 0, S_1 = 0\}).P(Y_1 / X_1 = 0)$$
$$+ P(S_1 = 0 / S_0 = 0).P(X_1 = 1 / \{S_0 = 0, S_1 = 0\}).P(Y_1 / X_1 = 1) = 0.5x1x0.3 + 0.5x0x0.2$$
$$= 0.15$$

$$\gamma_1^{RA}(0,1) = P(S_1 = 1 / S_0 = 0).P(X_1 = 0 / \{S_0 = 0, S_1 = 1\}).P(Y_1 / X_1 = 0)$$
$$+ P(S_1 = 1 / S_0 = 0).P(X_1 = 1 / \{S_0 = 0, S_1 = 1\}).P(Y_1 / X_1 = 1) = 0.5x0x0.3 + 0.5x1x0.2$$
$$= 0.1$$

i=2:

$$\gamma_2^{RA}(0,0) = \sum_{X} P(S_2 = 0 / S_1 = 0).P(X_2 = x / \{S_1 = 0, S_2 = 0\}).P(Y_2 / X_2 = x)$$
$$= P(S_i = 0 / S_1 = 0).P(X_2 = 0 / \{S_1 = 0, S_2 = 0\}).P(Y_2 / X_2 = 0)$$
$$+ P(S_2 = 0 / S_1 = 0).P(X_2 = 1 / \{S_1 = 0, S_2 = 0\}).P(Y_2 / X_2 = 1) = 0.5x1x0.4 + 0.5x0x0.1$$
$$= 0.2$$

$$\gamma_2^{RA}(0,1) = P(S_2 = 1 / S_1 = 0).P(X_2 = 0 / \{S_1 = 0, S_2 = 1\}).P(Y_2 / X_2 = 0)$$
$$+ P(S_2 = 1 / S_1 = 0).P(X_2 = 1 / \{S_1 = 0, S_2 = 1\}).P(Y_2 / X_2 = 1) = 0.5x0x0.4 + 0.5x1x0.1$$
$$= 0.05$$
$$\gamma_2^{RA}(1,0) = P(S_2 = 0 / S_1 = 1).P(X_2 = 0 / \{S_1 = 1, S_2 = 0\}).P(Y_2 / X_2 = 0)$$
$$+ P(S_2 = 0 / S_1 = 1).P(X_2 = 1 / \{S_1 = 1, S_2 = 0\}).P(Y_2 / X_2 = 1) = 0.5x0x0.4 + 0.5x1x0.1$$
$$= 0.05$$
$$\gamma_2^{RA}(1,1) = P(S_2 = 1 / S_1 = 1).P(X_2 = 0 / \{S_1 = 1, S_2 = 1\}).P(Y_2 / X_2 = 0)$$
$$+ P(S_2 = 1 / S_1 = 1).P(X_2 = 1 / \{S_1 = 1, S_2 = 1\}).P(Y_2 / X_2 = 1) = 0.5x1x0.4 + 0.5x0x0.1$$
$$= 0.2$$

i=5:

$$\gamma_5^{RA}(0,0) = 1x1x0.4 = 0.4$$

$\gamma_5^{RA}(1,0) = 1x1x0.1 = 0.1$

Forward recursive calculation of the values $\alpha_i^{RA}(u)$ is started by setting the initial conditions $\alpha_0^{RA}(0) = 1$, $\alpha_0^{RA}(m) = 0; m \neq 0$:

$$\alpha_1^{RA}(0) = \sum_{u'=0}^{U-1} \alpha_0^{RA}(u').\gamma_1^{RA}(u',u) = \sum_{u'=0}^{1} \alpha_0^{RA}(u').\gamma_1^{RA}(u',u) = \alpha_0^{RA}(0).\gamma_1^{RA}(0,0) + \alpha_0^{RA}(1).\gamma_1^{RA}(1,0)$$
$$= 1x0.15 + 0x0 = 0.15$$

$$\alpha_1^{RA}(1) = \alpha_0^{RA}(0).\gamma_1^{RA}(0,1) = 1x0.1 = 0.1$$

$$\alpha_2^{RA}(0) = \alpha_1^{RA}(0).\gamma_2^{RA}(0,0) + \alpha_1^{RA}(1).\gamma_2^{RA}(1,0) = 0.15x0.2 + 0.1x0.05 = 0.035$$
$$\alpha_2^{RA}(1) = \alpha_1^{RA}(0).\gamma_2^{RA}(0,1) + \alpha_1^{RA}(1).\gamma_2^{RA}(1,1) = 0.15x0.05 + 0.1x0.2 = 0.0275$$

$$\alpha_5^{RA}(0) = \alpha_2^{RA}(0).\gamma_5^{RA}(0,0) + \alpha_2^{RA}(1).\gamma_5^{RA}(1,0) = 0.035x0.4 + 0.0275x0.1 = 0.01675$$

Backward recursive calculation of the values $\beta_i^{RA}(u)$ is done by setting the contour conditions $\beta_5^{RA}(0) = 1$, $\beta_5^{RA}(m) = 0; m \neq 0$:

$$\beta_2^{RA}(0) = \beta_5^{RA}(0).\gamma_5^{RA}(0,0) = 1x0.4 = 0.4$$
$$\beta_2^{RA}(1) = \beta_5^{RA}(0).\gamma_5^{RA}(1,0) = 1x0.1 = 0.1$$

$$\beta_1^{RA}(0) = \beta_2^{RA}(0).\gamma_2^{RA}(0,0) + \beta_2^{RA}(1).\gamma_2^{RA}(0,1) = 0.4x0.2 + 0.1x0.05 = 0.085$$
$$\beta_1^{RA}(1) = \beta_2^{RA}(0).\gamma_2^{RA}(1,0) + \beta_2^{RA}(1).\gamma_2^{RA}(1,1) = 0.4x0.05 + 0.1x0.2 = 0.04$$

$$\beta_0^{RA}(0) = \beta_1^{RA}(0).\gamma_1^{RA}(0,0) + \beta_1^{RA}(1).\gamma_1^{RA}(0,1) = 0.085x0.15 + 0.04x0.1 = 0.01675$$

Once the values $\gamma_i^{RA}(u',u)$, $\alpha_i^{RA}(u)$ and $\beta_i^{RA}(u)$ have been determined, then the values $\lambda_i^{RA}(u)$ and $\sigma_i^{RA}(u',u)$ can be calculated as:

$$\lambda_1^{RA}(0) = \alpha_1^{RA}(0).\beta_1^{RA}(0) = 0.15x0.085 = 0.01275$$
$$\lambda_1^{RA}(1) = \alpha_1^{RA}(1).\beta_1^{RA}(1) = 0.1x0.04 = 0.004$$

The coefficients $\lambda_i^{RA}(u)$ determine the estimates for input symbols '1' and '0' when there is only one branch or transition of the trellis arriving at a given node, which then defines the value of that node. This happens for instance in the trellis of figure at nodes $\lambda_1^{RA}(0)$ and $\lambda_1^{RA}(1)$:

$$\frac{\lambda_1^{RA}(0)}{\lambda_1^{RA}(0) + \lambda_1^{RA}(1)} = \frac{0.01275}{0.01275 + 0.004} = 0.76119$$

Soft decision for '0' at position 1

$$\frac{\lambda_1^{RA}(1)}{\lambda_1^{RA}(0)+\lambda_1^{RA}(1)} = \frac{0.004}{0.01275+0.004} = 0.23881$$

Soft decision for '1' at position 1

Coefficients $\sigma_i^{RA}(u',u)$ are then utilized for determining the soft decisions when there are two or more transitions or branches arriving at a given node of the trellis, and when these branches are assigned the different input symbols:

$$\sigma_2^{RA}(0,0) = \alpha_1^{RA}(0).\gamma_2^{RA}(0,0).\beta_2^{RA}(0) = 0.15x0.2x0.4 = 0.012$$
$$\sigma_2^{RA}(1,0) = \alpha_1^{RA}(1).\gamma_2^{RA}(1,0).\beta_2^{RA}(0) = 0.1x0.05x0.4 = 0.002$$

$$\sigma_2^{RA}(0,1) = \alpha_1^{RA}(0).\gamma_2^{RA}(0,1).\beta_2^{RA}(1) = 0.15x0.05x0.1 = 0.00075$$
$$\sigma_2^{RA}(1,1) = \alpha_1^{RA}(1).\gamma_1^{RA}(1,1).\beta_2^{RA}(1) = 0.1x0.2x0.1 = 0.002$$

$$\sigma_5^{RA}(0,0) = \alpha_2^{RA}(0).\gamma_5^{RA}(0,0).\beta_5^{RA}(0) = 0.035x0.4x1 = 0.014$$
$$\sigma_5^{RA}(1,0) = \alpha_2^{RA}(1).\gamma_5^{RA}(1,0).\beta_5^{RA}(0) = 0.0275x0.1x1 = 0.00275$$

These values allow us to determine soft decisions for the corresponding nodes. For instance, for position $i=2$, the trellis transition probabilities involved in the calculation of a soft decision for '0' are:

$$\frac{\sigma_2^{RA}(0,0)+\sigma_2^{RA}(1,1)}{\sigma_2^{RA}(0,0)+\sigma_2^{RA}(1,1)+\sigma_2^{RA}(0,1)+\sigma_2^{RA}(1,0)} = \frac{0.012+0.002}{0.012+0.002+0.00075+0.002}$$
$$= 0.8358$$

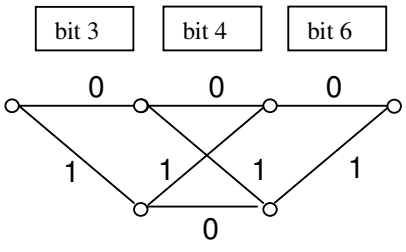
Which is a soft decision for '0' at position 2. The soft decision for '1' at that position is then $1-0.8352=0.1642$. For position $i=5$, the trellis transition probabilities involved in the calculation of a soft decision for '0' are:

$$\frac{\sigma_5^{RA}(0,0)}{\sigma_5^{RA}(0,0)+\sigma_5^{RA}(1,0)} = \frac{0.014}{0.014+0.00275} = 0.8358$$

And the soft decision for '1' at position $i=5$ is:

$$\frac{\sigma_5^{RA}(1,0)}{\sigma_5^{RA}(0,0)+\sigma_5^{RA}(1,0)} = \frac{0.00275}{0.014+0.00275} = 0.1642$$

For the row code of bits 3, 4 and 6, whose trellis is seen in the following figure:



i=3:

$\gamma_3^{RB}(0,0) = 0.5x1x0.1 = 0.05$
$\gamma_3^{RB}(0,1) = 0.5x1x0.4 = 0.2$

i=4:

$\gamma_4^{RB}(0,0) = 0.5x1x0.4 = 0.2$
$\gamma_4^{RB}(0,1) = 0.5x1x0.1 = 0.05$
$\gamma_4^{RB}(1,0) = 0.5x1x0.1 = 0.05$
$\gamma_4^{RB}(1,1) = 0.5x1x0.4 = 0.2$

i=6:

$\gamma_6^{RB}(0,0) = 1x1x0.4 = 0.4$
$\gamma_6^{RB}(1,0) = 1x1x0.1 = 0.1$

Forward recursive calculation of the values $\alpha_i^{RB}(u)$ is started by setting the initial conditions $\alpha_2^{RB}(0) = 1$, $\alpha_2^{RB}(m) = 0; m \neq 0$:

$\alpha_3^{RB}(0) = \alpha_2^{RB}(0).\gamma_3^{RB}(0,0) + \alpha_2^{RB}(1).\gamma_3^{RA}(1,0) = 1x0.05 + 0x0 = 0.05$
$\alpha_3^{RB}(1) = \alpha_2^{RB}(0).\gamma_3^{RB}(0,1) = 1x0.2 = 0.2$

$\alpha_4^{RB}(0) = \alpha_3^{RB}(0).\gamma_4^{RB}(0,0) + \alpha_3^{RB}(1).\gamma_4^{RB}(1,0) = 0.05x0.2 + 0.2x0.05 = 0.02$
$\alpha_4^{RB}(1) = \alpha_3^{RB}(0).\gamma_4^{RB}(0,1) + \alpha_3^{RB}(1).\gamma_4^{RB}(1,1) = 0.05x0.05 + 0.2x0.2 = 0.0425$

$\alpha_6^{RB}(0) = \alpha_4^{RB}(0).\gamma_6^{RB}(0,0) + \alpha_4^{RB}(1).\gamma_6^{RB}(1,0) = 0.02x0.4 + 0.0425x0.1 = 0.01225$

Backward recursive calculation of the values $\beta_i^{RB}(u)$ is done by setting the contour conditions $\beta_6^{RB}(0) = 1$, $\beta_6^{RB}(m) = 0; m \neq 0$:

$\beta_4^{RB}(0) = \beta_6^{RB}(0).\gamma_6^{RB}(0,0) = 1x0.4 = 0.4$
$\beta_4^{RB}(1) = \beta_6^{RB}(0).\gamma_6^{RB}(1,0) = 1x0.1 = 0.1$

$\beta_3^{RB}(0) = \beta_4^{RB}(0).\gamma_4^{RB}(0,0) + \beta_4^{RB}(1).\gamma_4^{RB}(0,1) = 0.4x0.2 + 0.1x0.05 = 0.085$
$\beta_3^{RB}(1) = \beta_4^{RB}(0).\gamma_4^{RB}(1,0) + \beta_4^{RB}(1).\gamma_4^{RB}(1,1) = 0.4x0.05 + 0.1x0.2 = 0.04$

Once the values $\gamma_i^{RB}(u',u)$, $\alpha_i^{RB}(u)$ and $\beta_i^{RB}(u)$ have been determined, then the values $\lambda_i^{RB}(u)$ and $\sigma_i^{RB}(u',u)$ can be calculated as:

$\lambda_3^{RB}(0) = \alpha_3^{RB}(0).\beta_3^{RB}(0) = 0.05x0.085 = 0.00425$

$$\lambda_3^{RB}(1) = \alpha_3^{RB}(1).\beta_3^{RB}(1) = 0.2x0.04 = 0.008$$

coefficients $\lambda_i^{RA}(u)$ determine the estimates for input symbols '1' and '0' when there is only one branch or transition of the trellis arriving at a given node, which then defines the value of that node. This happens for instance in the trellis of figure at nodes $\lambda_3^{RB}(0)$ and $\lambda_3^{RB}(1)$:

$$\frac{\lambda_3^{RB}(0)}{\lambda_3^{RB}(0)+\lambda_3^{RB}(1)} = \frac{0.00425}{0.00425+0.008} = 0.34694$$

Soft decision for '0' at position 3

$$\frac{\lambda_3^{RB}(1)}{\lambda_3^{RB}(0)+\lambda_3^{RB}(1)} = \frac{0.008}{0.00425+0.008} = 0.65306$$

Soft decision for '1' at position 3

Coefficients $\sigma_i^{RB}(u',u)$ are then utilized for determining the soft decisions when there are two or more transitions or branches arriving at a given node of the trellis, and when these branches are assigned the different input symbols:

$$\sigma_4^{RB}(0,0) = \alpha_3^{RB}(0).\gamma_4^{RB}(0,0).\beta_4^{RB}(0) = 0.05x0.2x0.4 = 0.004$$
$$\sigma_4^{RB}(1,0) = \alpha_3^{RB}(1).\gamma_4^{RB}(1,0).\beta_4^{RB}(0) = 0.2x0.05x0.4 = 0.004$$

$$\sigma_4^{RB}(0,1) = \alpha_3^{RB}(0).\gamma_4^{RB}(0,1).\beta_4^{RB}(1) = 0.05x0.05x0.1 = 0.00025$$
$$\sigma_4^{RB}(1,1) = \alpha_3^{RB}(1).\gamma_4^{RB}(1,1).\beta_4^{RB}(1) = 0.2x0.2x0.1 = 0.004$$

$$\sigma_6^{RB}(0,0) = \alpha_4^{RB}(0).\gamma_6^{RB}(0,0).\beta_6^{RB}(0) = 0.02x0.4x1 = 0.008$$
$$\sigma_6^{RB}(1,0) = \alpha_4^{RB}(1).\gamma_6^{RB}(1,0).\beta_6^{RB}(0) = 0.0425x0.1x1 = 0.00425$$

These values allow us to determine soft decisions for the corresponding nodes. For instance, for position $i = 4$, the trellis transition probabilities involved in the calculation of a soft decision for '0' are:

$$\frac{\sigma_4^{RB}(0,0)+\sigma_4^{RB}(1,1)}{\sigma_4^{RB}(0,0)+\sigma_4^{RB}(1,1)+\sigma_4^{RB}(0,1)+\sigma_4^{RB}(1,0)} = \frac{0.004+0.004}{0.004+0.004+0.00025+0.004}$$
$$= 0.65306$$

Which is a soft decision for '0' at position 4. The soft decision for '1' at that position is then $1 \quad 0.65306 = 0.34694$. For position $i = 6$, the trellis transition probabilities involved in the calculation of a soft decision for '0' are:

$$\frac{\sigma_6^{RB}(0,0)}{\sigma_6^{RB}(0,0)+\sigma_6^{RB}(1,0)} = \frac{0.008}{0.008+0.00425} = 0.65306$$

And the soft decision for '1' at position $i = 6$ is:

$$\frac{\sigma_6^{RB}(1,0)}{\sigma_6^{RB}(0,0)+\sigma_6^{RB}(1,0)} = \frac{0.00425}{0.008+0.00425} = 0.34694$$

A decision taken at this point generates the decoded vector $d = (0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0)$. The decoding has to continue in order to determine the whole decoded vector. The second decoder can use the a priori information provided as extrinsic information by the first decoder.
The following calculations are done in logarithmic form in order to determine the extrinsic information that the first decoder sends to the second decoder. Estimates in logarithmic form are equal to:

$$LLRR_1^{(1)} = ln(0.23881/0.76119) = -1.15921$$
$$LLRR_2^{(1)} = ln(0.1642/0.8358) = -1.6275$$
$$LLRR_3^{(1)} = ln(0.6531/0.3469) = +0.6325$$
$$LLRR_4^{(1)} = ln(0.3469/0.0.6531) = -0.6325$$
$$LLRR_5^{(1)} = ln(0.1642/0.8358) = -1.6275$$
$$LLRR_6^{(1)} = ln(0.3469/0.6531) = -0.6325$$

LLRR is the Logarithmic Likelihood Ratio for the row code.
The information to be subtracted from the logarithmic estimates is evaluated as:

$$LCR_1^{(1)} = ln(0.2/0.3) = -0.4055$$
$$LCR_2^{(1)} = ln(0.1/0.4) = -1.3863$$
$$LCR_3^{(1)} = ln(0.4/0.1) = +1.3863$$
$$LCR_4^{(1)} = ln(0.1/0.4) = -1.3863$$
$$LCR_5^{(1)} = ln(0.1/0.4) = -1.3863$$
$$LCR_6^{(1)} = ln(0.1/0.4) = -1.3863$$

The extrinsic information that is going to be passed as a priori information of the second decoder is determined by doing:

$$LER_1^{(1)} = LLRR_1^{(1)} - LCR_1^{(1)} = -1.1592 + 0.4055 = -0.7538$$
$$LER_2^{(1)} = LLRR_2^{(1)} - LCR_2^{(1)} = -1.6275 + 1.3863 = -0.2412$$
$$LER_3^{(1)} = LLRR_3^{(1)} - LCR_3^{(1)} = 0.6325 - 1.3863 = -0.7538$$
$$LER_4^{(1)} = LLRR_4^{(1)} - LCR_4^{(1)} = -0.6325 + 1.3863 = +0.7538$$
$$LER_5^{(1)} = LLRR_5^{(1)} - LCR_5^{(1)} = -1.6275 + 1.3863 = -0.2412$$
$$LER_6^{(1)} = LLRR_6^{(1)} - LCR_6^{(1)} = -0.6325 + 1.3863 = +0.7538$$

These estimates are converted into the apriori information $L_bC_i^{(1)}$ for the second (Column) decoder. The two independent column codes have the same type of trellis,

which is also the same as for the two independent row codes. We now take into account the permutation rule by properly assigning the apriori information.

$$L_bC_1^{(1)} = LER_1^{(1)} = -0.7538$$
$$L_bC_2^{(1)} = LER_2^{(1)} = -0.2412$$
$$L_bC_3^{(1)} = LER_3^{(1)} = -0.7538$$
$$L_bC_4^{(1)} = LER_4^{(1)} = +0.7538$$

These a priori estimates can be converted into probabilities by using expression:

$$P_bC_i^{(1)}(b_i = \pm 1) = \frac{e^{-L_bC_i^{(1)}/2}}{1+e^{-L_bC_i^{(1)}}} e^{-b_i L_bC_i^{(1)}/2}$$

these a priori probabilities are the updated information for the second decoder. They can be calculated using the above expression, and they are equal to:

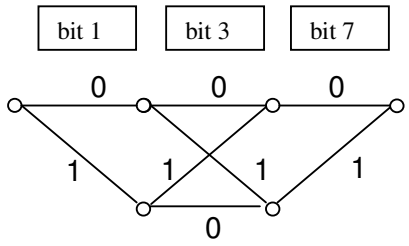
$$P_bC_1^{(1)}(b_1 = -1) = 0.6800\,,\ P_bC_1^{(1)}(b_1 = +1) = 0.3200$$
$$P_bC_2^{(1)}(b_2 = -1) = 0.5600\,,\ P_bC_2^{(1)}(b_2 = +1) = 0.4400$$
$$P_bC_3^{(1)}(b_3 = -1) = 0.6800\,,\ P_bC_3^{(1)}(b_3 = +1) = 0.3200$$
$$P_bB_4^{(1)}(b_4 = -1) = 0.3200\,,\ P_bB_4^{(1)}(b_4 = +1) = 0.6800$$

since parity bits for the first decoder are different form those of the second decoder, probabilities for the redundancy bits of the second decoder are:

$$P_bC_7^{(1)}(b_7 = -1) = 1.0000 \text{ or } P_bC_7^{(1)}(b_7 = +1) = 1.0000$$
$$P_bC_8^{(1)}(b_8 = -1) = 1.0000 \text{ or } P_bC_8^{(1)}(b_8 = +1) = 1.0000$$

In the column code, bits 1, 3 and 7 are related as in the trellis seen in the figure below. The same happens to bits 2, 4 and 8, which constitute a parity check code that is independent from the code of bits 1, 3 and 7. For these two codes we can calculate the values $\gamma_i(u',u)$. Coefficient for the column code of bits 1, 3 and 7 will be denoted with a superindex $\gamma_i^{CA}(u',u)$, whereas coefficients of the row code of bits 2, 4 and 8 will be denoted as $\gamma_i^{CB}(u',u)$.

The trellis for the column code A (sub-code 137) is then:



By taking into account that the trellis involves bits 1, 3 and 7, we are also directly taking into account the permutation rule of the whole turbo code.

With these updated probabilities, we can now calculate coefficients $\gamma_i^{CA}(u',u)$ for the column code. Thus, for instance, and for i = 1:

i=1:

$\gamma_1^{CA}(0,0) = 0.68x1x0.3 = 0.2040$
$\gamma_1^{CA}(0,1) = 0.32x1x0.2 = 0.0640$

i=3:

$\gamma_3^{CA}(0,0) = 0.0680$
$\gamma_3^{CA}(0,1) = 0.1280$
$\gamma_3^{CA}(1,0) = 0.1280$
$\gamma_{23}^{CA}(1,1) = 0.0680$

i=7:

$\gamma_7^{CA}(0,0) = 0.4000$
$\gamma_7^{CA}(1,0) = 0.1000$

Forward recursive calculation of the values $\alpha_i^{CA}(u)$ is started by setting the initial conditions $\alpha_0^{CA}(0) = 1$, $\alpha_0^{CA}(m) = 0; m \neq 0$, thus:

$\alpha_1^{CA}(0) = \alpha_0^{CA}(0).\gamma_1^{CA}(0,1) = 0.2040$
$\alpha_1^{CA}(1) = \alpha_0^{CA}(0).\gamma_1^{CA}(0,1) = 0.0640$

$\alpha_3^{CA}(0) = 0.022064$
$\alpha_3^{CA}(1) = 0.020464$

$\alpha_7^{CA}(0) = 0.011872$

Backward recursive calculation of the values $\beta_i^{CA}(u)$ is done by setting the contour conditions $\beta_7^{CA}(0) = 1$, $\beta_7^{CA}(m) = 0; m \neq 0$:

$\beta_3^{CA}(0) = 0.4000$
$\beta_3^{CA}(1) = 0.1000$

$\beta_1^{CA}(0) = 0.0400$
$\beta_1^{CA}(1) = 0.0580$

Once the values $\gamma_i^{CA}(u',u)$, $\alpha_i^{CA}(u)$ and $\beta_i^{CA}(u)$ have been determined, then the values $\lambda_i^{CA}(u)$ and $\sigma_i^{CA}(u',u)$ can be calculated as:

$\lambda_i^{CA}(0) = 0.00816$
$\lambda_i^{CA}(1) = 0.003712$

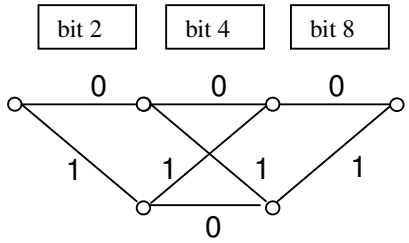
Coefficients $\sigma_i^{CA}(u', u)$ are then utilized for determining the soft decisions when there are two or more transitions or branches arriving at a given node of the trellis, and when these branches are assigned the different input symbols:

$\sigma_3^{CA}(0,0) = 0.0055488$
$\sigma_3^{CA}(1,0) = 0.0032768$

$\sigma_3^{CA}(0,1) = 0.0026112$
$\sigma_3^{CA}(1,1) = 0.0004352$

$\sigma_7^{CA}(0,0) = 0.0088256$
$\sigma_7^{CA}(1,0) = 0.0030464$

The trellis for the column code B is seen below:



By taking into account that the trellis involves bits 2, 4 and 8, we are also directly taking into account the permutation rule of the whole turbo code.

With these updated probabilities, we can now calculate coefficients $\gamma_i^{CB}(u', u)$ for the column code.

i=2:

$\gamma_2^{CB}(0,0) = 0.2240$
$\gamma_2^{CB}(0,1) = 0.0440$

i=4:

$\gamma_4^{CB}(0,0) = 0.1280$
$\gamma_4^{CB}(0,1) = 0.0680$
$\gamma_4^{CB}(1,0) = 0.0680$
$\gamma_4^{CB}(1,1) = 0.1280$

i=8:

$\gamma_8^{CB}(0,0) = 0.4000$
$\gamma_8^{CB}(1,0) = 0.1000$

Forward recursive calculation of the values $\alpha_i^{CB}(u)$ is started by properly setting the initial conditions:

$\alpha_2^{CB}(0) = 0.2240$
$\alpha_2^{CB}(1) = 0.0440$

$\alpha_4^{CB}(0) = 0.031664$
$\alpha_4^{CB}(1) = 0.020864$

$\alpha_8^{CB}(0) = 0.014752$

Backward recursive calculation of the values $\beta_i^{CB}(u)$ is done by setting the contour conditions $\beta_8^{CB}(0) = 1$, $\beta_8^{CB}(m) = 0; m \neq 0$:

$\beta_4^{CB}(0) = 0.4000$
$\beta_4^{CB}(1) = 0.1000$

$\beta_2^{CB}(0) = 0.0580$
$\beta_2^{CB}(1) = 0.0400$

Once the values $\gamma_i^{CB}(u',u)$, $\alpha_i^{CB}(u)$ and $\beta_i^{CB}(u)$ have been determined, then the values $\lambda_i^{CB}(u)$ and $\sigma_i^{CB}(u',u)$ can be calculated as:

$\lambda_2^{CB}(0) = 0.012992$
$\lambda_2^{CB}(1) = 0.001760$

Coefficients $\sigma_i^{CB}(u',u)$ are then utilized for determining the soft decisions when there are two or more transitions or branches arriving at a given node of the trellis, and when these branches are assigned the different input symbols:

$\sigma_i^{CB}(u',u)$

$\sigma_4^{CB}(0,0) = 0.0114688$
$\sigma_4^{CB}(1,0) = 0.0011968$

$\sigma_4^{CB}(0,1) = 0.0015232$
$\sigma_4^{CB}(1,1) = 0.0005632$

$\sigma_8^{CB}(0,0) = 0.0126656$
$\sigma_8^{CB}(1,0) = 0.0020864$

With all the values already calculated, an estimate or soft decision can be made for each step i of the decoded sequence.
We will calculate directly the LLRs of the decoding of the second decoder:

$$LLRC_1^{(1)} = -0.7877$$
$$LLRB_2^{(1)} = -1.9990$$
$$LLRC_3^{(1)} = -0.0162$$
$$LLRC_4^{(1)} = -1.4869$$
$$LLRC_7^{(1)} = -1.0637$$
$$LLRC_8^{(1)} = -1.8034$$

These estimates can be converted into probabilities:

$$P_bDC_1^{(1)}(b_1 = -1) = 0.6873,\ P_bDC_1^{(1)}(b_1 = +1) = 0.3127$$
$$P_bDC_2^{(1)}(b_3 = -1) = 0.8807,\ P_bDC_2^{(1)}(b_3 = +1) = 0.1193$$
$$P_bDC_3^{(1)}(b_2 = -1) = 0.5040,\ P_bDC_3^{(1)}(b_2 = +1) = 0.4960$$
$$P_bDC_4^{(1)}(b_4 = -1) = 0.8156,\ P_bDC_4^{(1)}(b_4 = +1) = 0.1844$$
$$P_bDC_7^{(1)}(b_7 = -1) = 0.7434,\ P_bDC_7^{(1)}(b_7 = +1) = 0.2566$$
$$P_bDC_8^{(1)}(b_8 = -1) = 0.8586 \text{ or } P_bDC_8^{(1)}(b_8 = +1) = 0.1414$$

A decision taken at this point by the column decoder generates the decoded vector $d = (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0)$.

The second decoder makes use of a priori information that should be subtracted from these estimations to calculate the extrinsic information that the second decoder passes to the first one.
The information to be subtracted from the logarithmic estimates of the second decoder in the first iteration is evaluated as:

$$LCC_1^{(1)} = ln(0.2/0.3) = -0.4055$$
$$LCC_2^{(1)} = ln(0.1/0.4) = -1.3863$$
$$LCC_3^{(1)} = ln(0.4/0.1) = +1.3863$$
$$LCC_4^{(1)} = ln(0.1/0.4) = -1.3863$$

and:

$$LbC_1^{(1)} = ln(0.32/0.68) = -0.7538$$
$$LbC_2^{(1)} = ln(0.44/0.56) = -0.2412$$
$$LbC_3^{(1)} = ln(0.32/0.68) = -0.7538$$
$$LbC_4^{(1)} = ln(0.68/0.32) = +0.7538$$

The extrinsic information that is going to be passed from the second decoder as a priori information of the first decoder is determined by doing:

$$LEC_1^{(1)} = LLRC_1^{(1)} - LCC_1^{(1)} - LbC_1^{(1)} = +0.3716$$
$$LEC_2^{(1)} = LLRC_2^{(1)} - LCC_2^{(1)} - LbC_2^{(1)} = -0.6487$$
$$LEC_3^{(1)} = LLRC_3^{(1)} - LCC_3^{(1)} - LbC_3^{(1)} = -0.3716$$
$$LEC_4^{(1)} = LLRC_4^{(1)} - LCC_4^{(1)} - LbC_4^{(1)} = -0.8544$$

These estimates are converted into the apriori information $L_bR_i^{(2)}$ for the second iteration of the first decoder. Thus,

$$L_bR_1^{(2)} = LEC_1^{(1)} = +0.3716$$
$$L_bR_2^{(2)} = LEC_2^{(1)} = -0.6487$$
$$L_bR_3^{(2)} = LEC_3^{(1)} = -0.3716$$
$$L_bR_4^{(2)} = LEC_4^{(1)} = -0.8544$$

These a priori estimates can be converted into probabilities by using expression:

$$P_bR_i^{(2)}(b_i = \pm 1) = \frac{e^{-L_bR_i^{(2)}/2}}{1 + e^{-L_bR_i^{(2)}}} e^{-b_i L_bR_i^{(2)}/2}$$

these a priori probabilities are the updated information for the first decoder. They can be calculated using the above expression and they are equal to:

$$P_bR_1^{(2)}(b_1 = -1) = 0.4082,\ P_bR_1^{(2)}(b_1 = +1) = 0.5918$$
$$P_bR_2^{(2)}(b_3 = -1) = 0.6567,\ P_bR_2^{(2)}(b_3 = +1) = 0.3433$$
$$P_bR_3^{(2)}(b_2 = -1) = 0.5918,\ P_bR_3^{(2)}(b_2 = +1) = 0.4082$$
$$P_bR_4^{(2)}(b_4 = -1) = 0.7015,\ P_bR_4^{(2)}(b_4 = +1) = 0.2985$$

since parity bits for the second decoder are different form those of the first decoder, probabilities for the redundancy bits of the second decoder are:

$$P_bR_5^{(2)}(b_5 = -1) = 1.0000 \text{ or } P_bR_5^{(2)}(b_5 = +1) = 1.0000$$
$$P_bR_6^{(2)}(b_6 = -1) = 1.0000 \text{ or } P_bR_6^{(2)}(b_6 = +1) = 1.0000$$

The decoding needs to continue with the following iteration. The iterative decoding performs as detailed above, and we summarize the resulting estimates for each iteration as follows. The first decoder performs the second iteration generating the estimates:

$$LLRA_1^{(2)} = -0.9379$$
$$LLRA_2^{(2)} = -1.7782$$
$$LLRA_3^{(2)} = -0.3204$$
$$LLRA_4^{(2)} = -1.8107$$
$$LLRA_5^{(2)} = -1.4102$$

$$LLRA_6^{(2)} = -0.7999$$

so that decoded probabilities are:

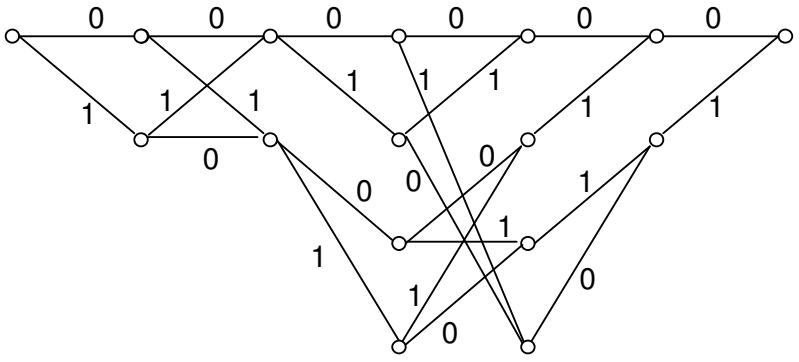
$$P_bDR_1^{(2)}(b_1 = -1) = 0.7187,\ P_bDR_1^{(2)}(b_1 = +1) = 0.2813$$
$$P_bDR_2^{(2)}(b_2 = -1) = 0.8555,\ P_bDR_2^{(2)}(b_2 = +1) = 0.1445$$
$$P_bDR_3^{(2)}(b_3 = -1) = 0.5794,\ P_bDR_3^{(2)}(b_3 = +1) = 0.4206$$
$$P_bDR_4^{(2)}(b_4 = -1) = 0.8594,\ P_bDR_4^{(2)}(b_4 = +1) = 0.1406$$
$$P_bDR_5^{(2)}(b_5 = -1) = 0.8038,\ P_bDR_5^{(2)}(b_5 = +1) = 0.1962$$
$$P_bDR_6^{(2)}(b_6 = -1) = 0.6900,\ P_bDR_6^{(2)}(b_6 = +1) = 0.3100$$

The decoded vector is now a code vector for the row code. The second decoder performs its second iteration and the resulting estimates are:

$$LLRC_1^{(2)} = -1.1136$$
$$LLRC_2^{(2)} = -0.3910$$
$$LLRC_3^{(2)} = -1.9536$$
$$LLRC_4^{(2)} = -1.7190$$
$$LLRC_5^{(2)} = -1.1987$$
$$LLRC_6^{(2)} = -1.9394$$

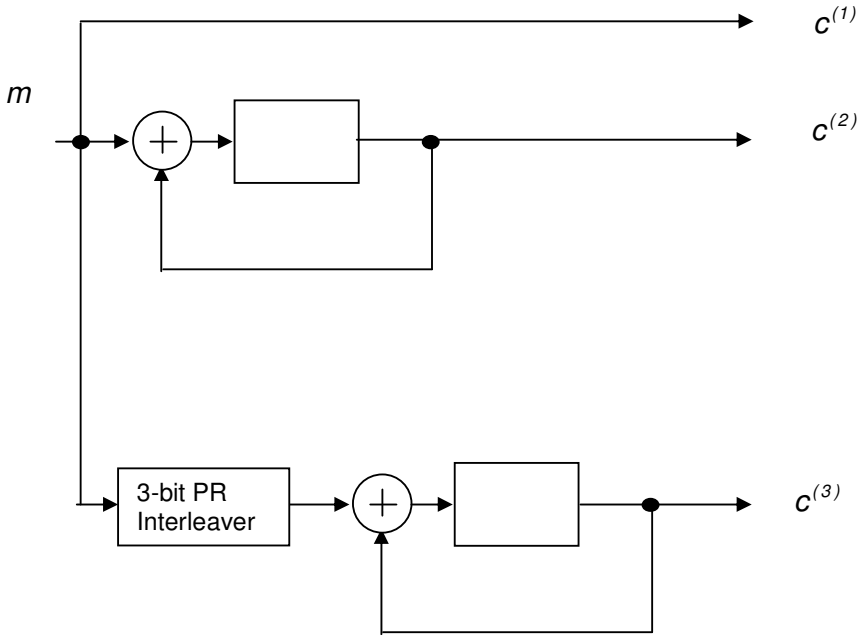
The decoded message is the same for both decoders at the second iteration, so that the whole decoded vector is the code vector $\boldsymbol{d} = \boldsymbol{c} = (0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0)$, the decoded information bits are 0000, and the decoding process was successfully completed.

A rather inefficient, and unnecessarily more complex way of solving this decoding is by means of the construction of a trellis for the whole row code, such as is seen in the following figure.

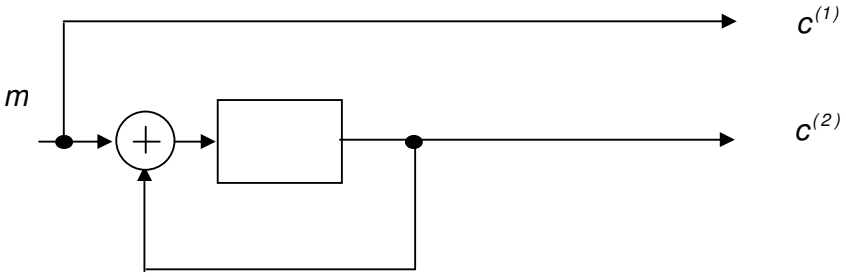


This trellis can be used also for the column code provided that bits 2 and 3 are permuted. It can be verified that the decoding using this trellis results into the same values of estimates, but utilizing a much more complex set of calculations in comparison to the method described above.

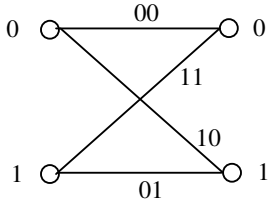
7.4)
The 1/3-rate turbo code encoded shown in the following figure has two constituent encoders of rate ½.

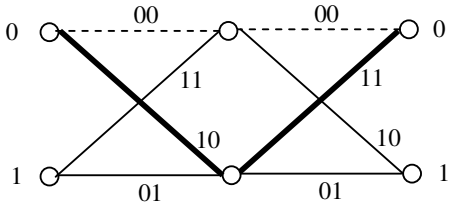


Each constituent encoder is of the form:



The following figures show the trellis of each of the constituent encoders and the way its minimum Hamming free distance is calculated:





The minimum Hamming free distance of each constituent code is then $d_{free} = 3$.

The minimum Hamming free distance of the whole code is determined in a simplified way. We look for the minimum weight sequence that starts at, and returns to the all-zero state, considering input sequences of three bits. For this, both encoders have to end at the all-zero state.

Let us see as an example the output sequence of the turbo code for the input $\boldsymbol{m} = (001)$. The fist bit to be input is the rightmost bit. The permutation rule

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

is applied as seen in the following table.

m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
1		0				
0		1				
0		0				

Thus, for $\boldsymbol{m} = (100)$:

m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
1	1	0	0	1	0	0
0	1	1	1	0	1	0
0	1	0	1	0	1	1

This case does not end at the all zero state.
Let us see as an example the output sequence of the turbo code for the input $\boldsymbol{m} = (010)$.

m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
0	0	0	0	0	0	0
1	1	0	0	1	0	0
0	1	1	1	0	1	0

With: $\boldsymbol{m} = (001)$

m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
0	0	1	1	0	0	0
0	0	0	1	0	0	1
1	1	0	1	1	0	1

With $\boldsymbol{m} = (110)$:

m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
1	1	0	0	**1**	0	0
1	0	1	1	**1**	**1**	0
0	0	1	0	0	0	**1**

With $\boldsymbol{m} = (101)$:

m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
1	1	1	1	1	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	0

With: $\boldsymbol{m} = (011)$

m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
0	0	1	1	0	0	0
1	1	0	1	1	0	1
1	0	1	0	1	1	1

With $\boldsymbol{m} = (111)$:

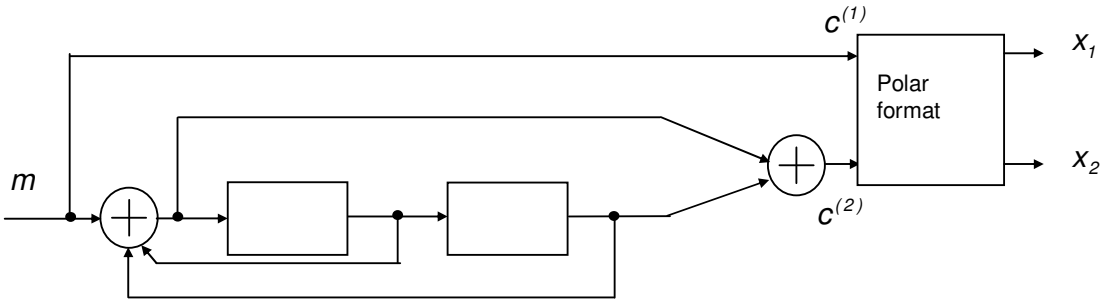
m1	State 1	m2	State 2	$c^{(1)}$	$c^{(2)}$	$c^{(3)}$
1	1	1	1	1	0	0
1	0	1	0	1	1	1
1	1	1	1	1	0	0

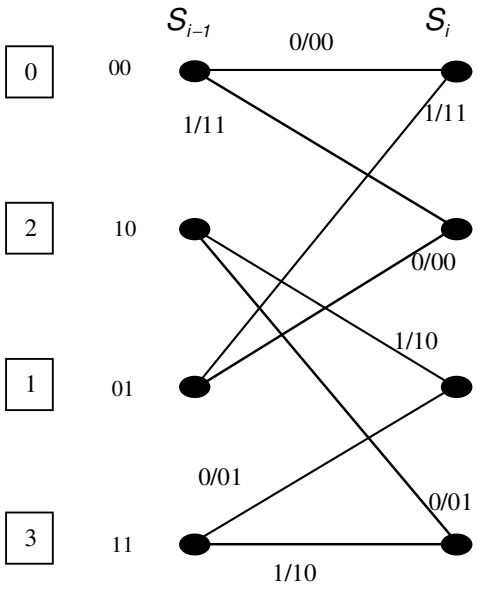
In spite of that some cases do not end at the all-zero state, this very simple method indicates that the minimum Hamming free distance is 4, which is determined by those inputs for which the sequence of the whole code starts and ends at the all-zero state. This is of course not rigorous, because there could be longer sequences that could end at having smaller weights, so that the code could have an even smaller minimum Hamming free distance. As a handmade exercise however, the solution is considered proper enough. The case $\boldsymbol{m} = (111)$ is also one in which the minimum Hamming free distance is 4, but here this happens in the middle of the input sequence, that is, the machine ends at the all-zero state after the second bit is input. Looking at length 6 input sequences in the two cases where both states are non-zero and the distance is less than 4 after inputting the first three bits confirms that the free distance is in fact 4.

We apologise for the mistake seen in the Answers to Problems for this item, which wrongly indicates that the minimum Hamming free distance of the whole code is 5 instead of 4.

7.5)
Unfortunately there is a text error in the description of this problem. Since the terminated code case was presented as an example in Chapter 7 of the book, we wanted to propose here a turbo code whose first encoder is not terminated. The text should say:
" the input or message vector is:
$\boldsymbol{m} = (-1 \;\; -1 \;\; -1 \;\; +1 \;\; -1 \;\; +1 \;\; -1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; -1)$. This input vector makes the first encoder sequence be **non-**terminated. "

We describe the encoder of each constituent code, and the corresponding trellis in the following two figures:





The decimal number at the left of each state identifies the decimal representation of each state. The following lists will describe the calculated values of the different parameters of the turbo decoding.
The input of the second decoder is determined by a proper permutation (as in Example 7.3 on page 243) of the input of the first encoder. We describe these input vectors below:

$$\boldsymbol{m}_{1rst} = (-1 \;\; -1 \;\; -1 \;\; +1 \;\; -1 \;\; +1 \;\; -1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; -1)$$

$$\boldsymbol{m}_{2nd} = (-1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; -1 \;\; -1 \;\; -1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; -1)$$

After being punctured and transmitted, and corrupted by AWGN, the received sequence, as tabulated in the following table, is then applied to the decoder:

Input sequence	Received sequence
-1	-0.5290 -0.3144
-1	-0.01479 -0.1210
-1	-0.1959 +0.03498
+1	+1.6356 -2.0913
-1	-0.9556 + 1.2332
+1	+1.7448 -0.7383
-1	-0.3742 -0.1085
-1	-1.2812 -1.8162
+1	+0.5848 +0.1905
+1	+0.6745 -1.1447
-1	-2.6226 -0.5711
+1	+0.7426 + 1.0968
+1	+1.1303 -1.6990
-1	-0.6537 -1.6155
+1	+2.5879 -0.5120
-1	-1.3861 -2.0449

These received values correspond to a noise dispersion of $\sigma = 1.1$. Bits were transmitted in normalized polar format ± 1. In the received sequence column of the above table the first (LH) value is that of the received information (systematic) bits and the second (RH) that of the alternately transmitted parity bits.

The decoding of this received vector is performed below. The received vector for the first decoder is:

Message bit	Redundancy bit
-0.5290	-0.3144
-0.01479	0.0000
-0.1959	+0.03498
+1.6356	0.0000
-0.9556	+ 1.2332
+1.7448	0.0000
-0.3742	-0.1085
-1.2812	0.0000
+0.5848	+0.1905
+0.6745	0.0000
-2.6226	-0.5711
+0.7426	0.0000
+1.1303	-1.6990
-0.6537	0.0000
+2.5879	-0.5120
-1.3861	0.0000

The values $\gamma_i(u',u)$ are first calculated and then $\alpha_i(u)$ and $\beta_i(u)$ can be also determined.
Values of $\gamma_i(u',u)$ for the first decoder in the first iteration, are described in the following list:

i = 1

2.0078	0	0.4980	0
0.4980	0	2.0078	0
0	0.8375	0	1.1940
0	1.1940	0	0.8375

2.0078 is for instance the value of $\gamma_1(0,0)$, and 0.4980 is the value of $\gamma_1(0,2)$. The other values of the list are not needed to be calculated for i=1.

i = 2

1.0123	0	0.9878	0
0.9878	0	1.0123	0
0	0.9878	0	1.0123
0	1.0123	0	0.9878

i = 3

1.1423	0	0.8754	0
0.8754	0	1.1423	0
0	0.8263	0	1.2103
0	1.2103	0	0.8263

i = 4

0.2588	0	3.8643	0
3.8643	0	0.2588	0
0	3.8643	0	0.2588
0	0.2588	0	3.8643

i = 5

0.7950	0	1.2579	0
1.2579	0	0.7950	0
0	0.1638	0	6.1044
0	6.1044	0	0.1638

i = 6

0.2365	0	4.2291	0
4.2291	0	0.2365	0
0	4.2291	0	0.2365
0	0.2365	0	4.2291

i = 7

1.4903	0	0.6710	0
0.6710	0	1.4903	0
0	0.8029	0	1.2455
0	1.2455	0	0.8029

i = 8

2.8831	0	0.3469	0
0.3469	0	2.8831	0
0	0.3469	0	2.8831
0	2.8831	0	0.3469

i = 9

0.5269	0	1.8979	0
1.8979	0	0.5269	0
0	1.3852	0	0.7219
0	0.7219	0	1.3852

i = 10

0.5727	0	1.7462	0
1.7462	0	0.5727	0
0	1.7462	0	0.5727
0	0.5727	0	1.7462

i = 11

14.0060	0	0.0714	0
0.0714	0	14.0060	0
0	0.1835	0	5.4492
0	5.4492	0	0.1835

i = 12

0.5413	0	1.8473	0
1.8473	0	0.5413	0
0	1.8473	0	0.5413
0	0.5413	0	1.8473

i = 13

1.6000	0	0.6250	0
0.6250	0	1.6000	0
0	10.3632	0	0.0965
0	0.0965	0	10.3632

i = 14

1.7164	0	0.5826	0
0.5826	0	1.7164	0
0	0.5826	0	1.7164
0	1.7164	0	0.5826

i = 15

0.1799	0	5.5600	0
5.5600	0	0.1799	0
0	12.9598	0	0.0772
0	0.0772	0	12.9598

i = 16

3.1441	0	0.3181	0
0.3181	0	3.1441	0
0	0.3181	0	3.1441
0	3.1441	0	0.3181

Forward recursive calculation of the values $\alpha_i(u)$ is started by setting the initial conditions $\alpha_0(0)=1$, $\alpha_0(m)=0; m\neq 0$:

The following list describes the calculated values of $\alpha_i(u)$ for the first decoder, in the first iteration:

1.0000000e+000	0.0000000e+000	0.0000000e+000	0.0000000e+000
2.0078386e+000	0.0000000e+000	4.9804801e-001	0.0000000e+000
2.0325442e+000	4.9199423e-001	1.9834333e+000	5.0417627e-001
2.7524567e+000	2.2490203e+000	2.3413637e+000	2.8170801e+000
9.4030924e+000	9.7766548e+000	1.1218224e+001	1.1491842e+001
1.9773114e+001	7.1988040e+001	1.9600210e+001	7.0362600e+001
3.0911792e+002	9.9528490e+001	1.0064406e+002	3.0220294e+002
5.2745834e+002	4.5720729e+002	3.5574788e+002	3.6798509e+002
1.6792767e+003	1.1843146e+003	1.5011058e+003	1.1532786e+003
3.1325225e+003	2.9118911e+003	3.8111199e+003	2.6811907e+003
6.8787051e+003	8.1905026e+003	7.1376304e+003	6.8644573e+003
9.6928058e+004	3.8715628e+004	1.1520746e+005	4.0154071e+004
1.2398932e+005	2.3455750e+005	2.0001161e+005	1.3654182e+005
3.4498211e+005	2.0859446e+006	4.5278975e+005	1.4343163e+006
1.8074125e+006	2.7257222e+006	3.7813988e+006	1.6128206e+006
1.5480141e+007	4.9130581e+007	1.0539485e+007	2.1193595e+007

Values are given in exponential form (for example, $1.5480141e+007 = 1.5480141x10^7$). Again, the above list is read as a matrix with entries i (rows, or bit positions) and j (columns, or state values in the trellis). 1.0000000e+000 is the value of $\alpha_0(0)$. 2.0078386e+000 is the value of $\alpha_1(0)$, and 4.9199423e-001 is the value of $\alpha_2(1)$, for example.

Backward recursive calculation of the values $\beta_i(u)$ is done by setting the contour conditions, $\beta_{16}(m)=1; for\ all\ m$ (First code is **not** terminated):

0.0000000e+000	0.0000000e+000	0.0000000e+000	0.0000000e+000
1.3671631e+008	1.3656783e+008	1.1856153e+008	1.1873363e+008
7.1350992e+007	6.2837297e+007	6.5280862e+007	5.5801411e+007
4.9204714e+007	1.7385051e+007	1.7300082e+007	4.2070032e+007
3.6626355e+006	3.7647496e+006	1.2487991e+007	1.0634829e+007
1.9193438e+006	1.6884863e+006	1.6987243e+006	2.0004382e+006

3.7505399e+005	3.7640694e+005	4.3287517e+005	4.5197500e+005
1.7301921e+005	2.3755943e+005	1.7467061e+005	1.9441164e+005
5.0835024e+004	6.1026893e+004	7.6282601e+004	5.3243200e+004
2.6783118e+004	4.8103672e+004	1.9349272e+004	1.3367137e+004
2.5230783e+004	9.6030247e+003	7.0634442e+003	4.5056208e+003
1.7979759e+003	7.8407703e+002	6.7647010e+002	1.2698304e+003
1.5230542e+002	1.8023503e+002	9.2867662e+002	6.3458787e+002
6.0405314e+001	8.9050093e+001	8.9050093e+001	6.0405314e+001
1.9872124e+001	1.9872124e+001	4.5135464e+001	4.5135464e+001
3.4621180e+000	3.4621180e+000	3.4621180e+000	3.4621180e+000
1.0000000e+000	1.0000000e+000	1.0000000e+000	1.0000000e+000

Once the values $\gamma_i(u',u)$, $\alpha_i(u)$ and $\beta_i(u)$ have been determined, then the values of the LLR estimates can be calculated. The following list describes these values:

Position of b_i	$L_1^{(1)}(b_i / \mathbf{Y})$	Estimated bits	Input or message bits
1	-1.5365936	-1	-1
2	-0.076558632	-1	-1
3	-0.87707531	-1	-1
4	+2.8030878	+1	+1
5	-1.7221617	-1	-1
6	+2.8949539	+1	+1
7	-0.65338000	-1	-1
8	-2.1014182	-1	-1
9	+0.99084643	+1	+1
10	+1.1271298	+1	+1
11	-4.4088385	-1	-1
12	+1.3086825	+1	+1
13	+1.7894979	+1	+1
14	-1.2174239	-1	-1
15	+4.3467953	+1	+1
16	-2.2910284	-1	-1

In this case, the error event was such that the first decoder could successfully decode the received vector. We could stop decoding here. In practice, however, we do not

know when the number of iterations is enough to arrive at the correct decision, and we usually set a given number of iterations to be performed.