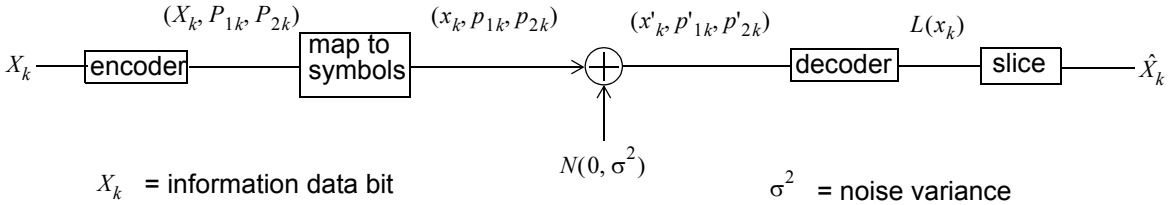
"Turbo Code Primer"

1.0 Introduction

This document gives a quick tutorial on MAP based turbo coders. Section 2 develops the background theory. Section 3 works through a simple numerical example.

2.0 Theory

A highlevel view of a rate 1/3 turbo code system is shown in figure 2.1. We use upper case to denote binary numbers and lower case to denote signal/symbol values. For each source information bit, a pair of convolutional encoders generate a corresponding pair of parity bits. The two parity bits and the original information bit are then mapped to symbols (signal values) for transmisson through an AWGN channel.



X_k = information data bit
P_{1k} = encoder #1 parity bit
P_{2k} = encoder #2 parity bit
x_k = information data symbol
p_{1k} = encoder #1 parity symbol
p_{2k} = encoder #2 parity symbol

σ^2 = noise variance
x'_k = noisy information data symbol
p'_{1k} = noisy encoder #1 parity symbol
p'_{2k} = noisy encoder #2 parity symbol
$L(x_k)$ = decoder soft decision
$\hat{X}_k$ = decoder hard decision

Figure 2.1 Turbo Code System

2.1 Turbo Encoder

The structure of a rate 1/3 encoder is shown in fig 2.2. Two identical convolutional encoders are used.The order of the information bits undergo a pseudo random permutation P prior to been fed into the second convolutional encoder. Accordingly a turbo code is a block coding scheme where a block of K information bits are buffered prior to applying the permutation.

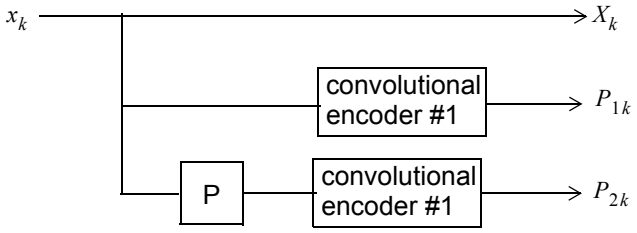
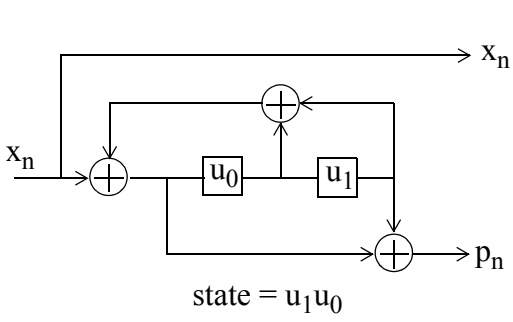


Figure 2.2 Turbo Encoder

Many types of permutation can be used, provided they are sufficiently random. The permutation makes the two constituent encoders appear uncorrelated at the receiver. In the example in section 3 we will use a permutation based on the state sequence of a maximal length shift register (PN sequence).

Each convolutional encoder is based on a recursive systematic convolutional code. Fig 2.3 shows a simple 4 state example. The operation of the encoder is summarised by the trellis diagram which shows the bit pairs output for each possible transition between sucessive states.



x_n	s_n	s_{n+1}	p_n
0	00	00	0
1		10	1
0	01	10	0
1		00	1
0	10	11	1
1		01	0
0	11	01	1
1		11	0

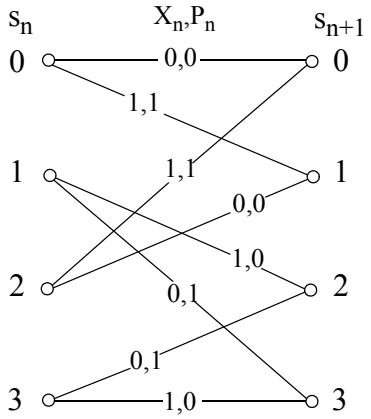


Figure 2.3 Recursive Systematic Code

The state transition diagram in fig 2.3 can be repeatedly drawn to generate a trellis diagram for the encoder. The trellis diagram (labelled with the transmit symbol values) for a sequence of 8 bits is shown in fig 2.4.

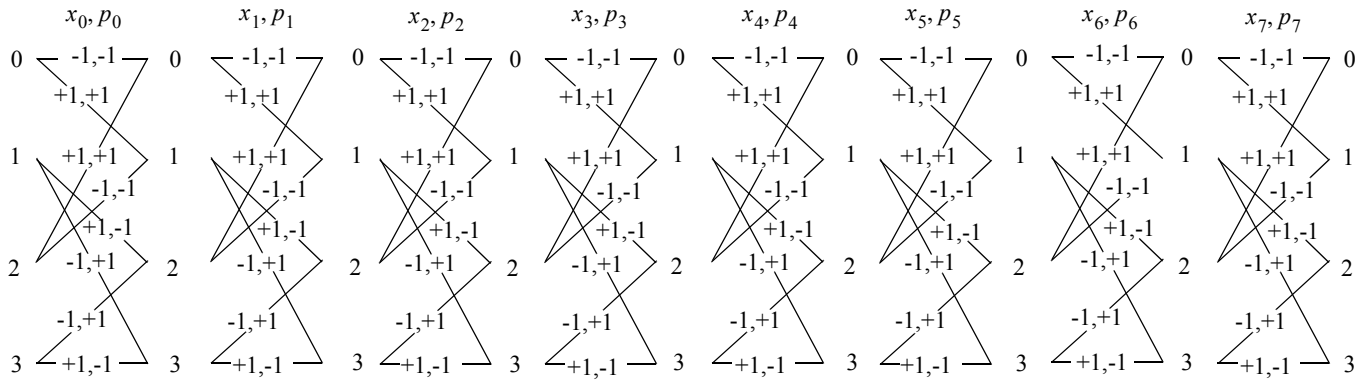


Figure 2.4 Trellis Diagram

2.2 Turbo Decoder

The structure of a turbo decoder is shown in fig 2.5. It consists of a pair of decoders which work cooperatively in order to refine and improve the estimate of the original information bits. The decoders are based on the MAP (maximum aposteriori probability) algorithm and output soft decision information learned from the noisy parity bits. Initially decoder #1 starts without initialisation information (apriori estimates are set to zero). In subsequent iterations, the soft decision information of one decoder is used to initialise the other decoder. The decoder information is cycled around the loop until the soft decisions converge on a stable set of values. The latter soft decisions are then sliced to recover the original binary sequence.

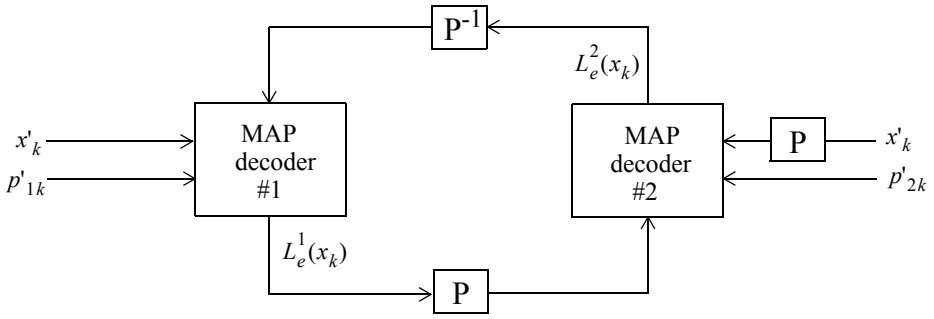


Figure 2.5 Turbo Decoder

2.2.1 MAP Decoder

The MAP algorithm minimises the probability of bit error by using the entire received sequence to identify the most probable bit at each stage of the trellis. The MAP algorithm does not constrain the set of bit estimates to necessarily correspond to a valid path through the trellis. So the results can differ from those generated by a Viterbi decoder which identifies the most probable valid path through the trellis.

We will use the following shorthand for the transmitted/received symbol pairs.

$$\boldsymbol{y}_k = \{x_k, p_k\} \qquad \boldsymbol{y} = y_{1,K} = \{y_1, y_2, \ldots, y_K\} \qquad \boldsymbol{y}'_k = \{x'_k, p'_k\} \qquad \boldsymbol{y}' = y'_{1,K} = \{y'_1, y'_2, \ldots, y'_K\}$$

The MAP soft decisions are defined as the log likelihood ratio:

$$L_{map}(x_k) = \log\left[\frac{Pr(x_k = +1 \mid \boldsymbol{y}')}{Pr(x_k = -1 \mid \boldsymbol{y}')}\right]$$

For a convolutional code we can express the likelihood ratio in terms of the trellis:

$$\frac{Pr(x_k = +1 \mid \boldsymbol{y}')}{Pr(x_k = -1 \mid \boldsymbol{y}')} = \frac{\sum_{(s',s) \in S^+} Pr(s_{k-1} = s', s_k = s \mid \boldsymbol{y}')}{\sum_{(s',s) \in S^-} Pr(s_{k-1} = s', s_k = s \mid \boldsymbol{y}')} = \frac{\sum_{(s',s) \in S^+} Pr(s_{k-1} = s', s_k = s, \boldsymbol{y}')}{\sum_{(s',s) \in S^-} Pr(s_{k-1} = s', s_k = s, \boldsymbol{y}')}$$

The numerator sum is over all possible state transitions associated with a '1' data bit, and the denominator over all possible state transitions associated with a '0' data bit. Fig 2.6 shows the state transitions for the case of our example 2-state code.

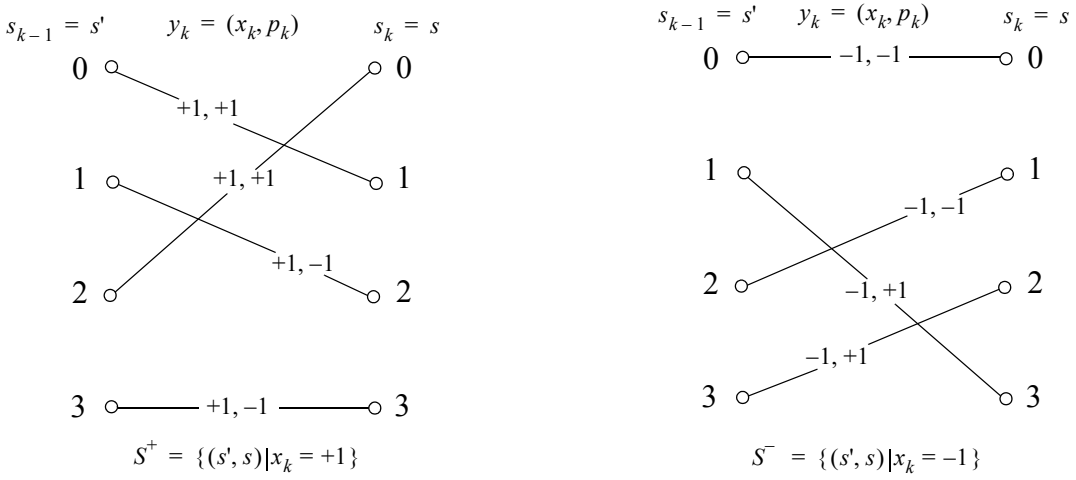


Figure 2.6 Transition Sets

The probability of a particular state transition and the noisy observation associated with a trellis transition can be expressed (using Bayes theorem) as:

$$Pr(s_{k-1} = s', s_k = s, y'_{1,K}) = Pr(s_{k-1} = s', y'_{1,k-1}) \cdot Pr(s_k = s, y'_k | s_{k-1} = s') \cdot Pr(y'_{k+1,K} | s_k = s)$$
$$= \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)$$

Note the probabilities associated with the continuous valued received observation taking a particular value are infinitesimally small. As the final result will be a probability ratio (the likelihood ratio) we can relax the notation and work with probabilities to help keep the mathematics simple.

In terms of the above definitions of $\alpha_{k-1}(s')$, $\gamma_k(s', s)$ and $\beta_k(s)$, the a posteriori likelihood ratio can be rewrriten as the ratio:

$$\frac{Pr(x_k = +1 | \boldsymbol{y}')}{Pr(x_k = -1 | \boldsymbol{y}')} = \frac{\sum_{(s', s) \in S^+} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)}{\sum_{(s', s) \in S^-} \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)}$$

The term $\alpha_k(s')$ is the probability of arriving at a branch in a particular state and the sequence of noisy obervations $y'_{1,k} = y'_1, y'_2, \ldots, y'_k$ which led up to that state. By summing over all paths leading into that state we get a forward recursion for calculating $\alpha_k(s')$ in terms of the values of $\gamma_k(s', s)$.

$$\alpha_k(s) = Pr(s_k = s, y'_{1,k}) = \sum_{s'} Pr(s_{k-1} = s', y'_{1,k-1}) \cdot Pr(s_k = s, y'_k | s_{k-1} = s')$$
$$= \sum_{s'} \alpha_{k-1}(s') \cdot \gamma_k(s', s)$$

To begin the forward recursion we need to initialise the forward state probabilities. In turbo coders all convolutional encoders are started in state 0. Thus we can begin the forward recursion with:

$$\alpha_0(s) = Pr(s_0 = s) = \begin{matrix} 1 & s = 0 \\ 0 & s \neq 0 \end{matrix}$$

The term $\beta_k(s)$ is the probability of exiting a branch via a particular state s and the sequence of noisy observations $y'_{k+1,K} = y'_{k+1}, y'_{k+2}, \ldots, y'_K$ which finish off the trellis. By summing over all paths exiting that state we get a backward recursion for calculating $\beta_k(s)$ in terms of the values of $\gamma_k(s', s)$.

$$\beta_k(s') = Pr(y'_{k+1,K} | s_k = s) = \sum_s Pr(s_{k+1} = s, y'_{k+1} | s_k = s') \cdot Pr(y'_{k+2,K} | s_k = s')$$

$$= \sum_s \gamma_{k+1}(s', s) \cdot \beta_{k+1}(s)$$

To begin the backward recursion we need to initialise the backward state probabilities. Convolutional encoder #1 is usually terminated at state 0. However, in general, the final state for convolutional encoder #2 is data dependent, and unknown beforehand. We will assume a uniform distribution for the final state of encoder #2. A suitable initialisation (where the number of states in the convolutional encoder is 2^v) is as follows:

$$\textit{MAP \#1:} \qquad \beta_K(s) = Pr(s_K = s) = \begin{matrix} 1 & s = 0 \\ 0 & s \neq 0 \end{matrix}$$

$$\textit{MAP \#2:} \qquad \beta_K(s) = Pr(s_K = s) = \frac{1}{2^v} \text{ for all } s$$

We will now derive explicit expressions which can be used in the receiver to calculate $\gamma_k(s', s)$.

For a given state transition, the transmitted signal is the databit and parity pair y_k. Also, for a given starting state, the next state is completely determined by the value of the databit. Using the Bayes theorem the branch probability can be expressed as:

$$\gamma_k(s', s) = Pr(s_k = s, y'_k | s_{k-1} = s') = Pr(y'_k | s_{k-1} = s', s_k = s) \cdot Pr(s_k = s | s_{k-1} = s') = Pr(y'_k | y_k) \cdot Pr(x_k)$$

The probability of the databit taking a particular value can be expressed in terms of the log likelihood of the apriori probability ratio:

$$Pr(x_k) = \frac{\exp\left[\frac{1}{2} L_a(x_k)\right]}{1 + \exp[L_a(x_k)]} \cdot \exp\left[\frac{1}{2} x_k L_a(x_k)\right] \qquad L_a(x_k) = \log\left[\frac{Pr(x_k = +1)}{Pr(x_k = -1)}\right]$$

$$= B_k \cdot \exp\left[\frac{1}{2} x_k L_a(x_k)\right]$$

The probability of the oberved noisy databit and parity symbols taking particular values can be expressed in terms of Gaussian probability distributions as:

$$Pr(y'_k | y_k) = Pr(x'_k | x_k) \cdot Pr(p'_k | p_k)$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[\frac{-(x'_k - x_k)^2}{2\sigma^2}\right] \cdot \Delta \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[\frac{-(p'_k - p_k)^2}{2\sigma^2}\right] \cdot \Delta$$

$$= A_k \cdot \exp\left[\frac{x'_k x_k - p'_k p_k}{\sigma^2}\right]$$

Here the Δ terms are infinitesimally small ranges about the particular values. These terms will cancel out in the final expressions for the likelihood ratios.

We can now express the transition probability $\gamma_k(s', s)$ in terms of log likelihood ratios and the noisy observations as:

$$\gamma_k(s', s) = A_k \cdot B_k \cdot \exp\left[\frac{1}{2}(x_k L_a(x_k) + x_k L_c x'_k + p_k L_c p'_k)\right] \qquad L_c = \frac{2}{\sigma^2}$$

Recalling the definition of the MAP log likelihhood ratio:

(sum of) emission prob, likelihood (sum of) state transition prob , prior

$$L_{map}(x_k) = \log\left[\frac{Pr(x_k = +1\,|y')}{Pr(x_k = -1\,|y')}\right] = \log\left[\frac{Pr(y'_k|x_k = +1)}{Pr(y'_k|x_k = -1)}\right] + \log\left[\frac{Pr(x_k = +1)}{Pr(x_k = -1)}\right]$$

We now compare this definition with the version which we derived based on the trellis structure.

$$L_{map}(x_k) = \log\left[\frac{\sum_{(s',s)\in S^+} \alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}{\sum_{(s',s)\in S^-} \alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}\right] = \log\left[\frac{\sum_{(s',s)\in S^+} \exp\left(\frac{L_a(x_k)}{2}\right)\cdot\exp\left(\frac{L_c x'_k}{2}\right)\cdot\alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}{\sum_{(s',s)\in S^-} \exp\left(\frac{-L_a(x_k)}{2}\right)\cdot\exp\left(\frac{-L_c x'_k}{2}\right)\cdot\alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}\right]$$

Noting that the summation in the numerator is over all state transitions associated with a databit symbol x_k equal to +1, and that the summation in the denominator is over all state transitions associated with a databit symbol x_k equal to -1, we have:

emission prob

transition prob (prior info)

$$L_{map}(x_k) = L_a(x_k) + L_c x'_k + \log\left[\frac{\sum_{(s',s)\in S^+} \alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}{\sum_{(s',s)\in S^-} \alpha_{k-1}(s')\cdot\exp\left(\frac{p_k L_c p'_k}{2}\right)\cdot\beta_k(s)}\right]$$

We can now seperate the MAP log likelihood into three distinct components.

$$L_{map}(x_k) = L_a(x_k) + L_c x'_k + L_e(x_k)$$

The first term $L_a(x_k)$ is the apriori information. This information is our initial estimate prior to running the MAP algorithm.

The second term $L_c x'_k$ is the information provided by that part of the noisy observation which does not depend on the convolutional code constraints.

The third term $L_e(x_k)$ is the information which we learn via the parity constraint. This information is referred to as 'extrinsic' information, and is that information gleaned from the code.

In a turbo decoder the extrinsic information of one MAP decoder is used as the apriori input to the other MAP decoder. In the turbo decoder iterations the extrinsic information is ping ponged back and forth between MAP decoders.

3.0 Worked Example

We will now illustrate the implementation of a turbo coder using a very simple example. We use the 4 state encoders shown in fig 2.3 and a block length of only 6 data bits with 2 trellis termination bits.

Note practical implementations require considerably much longer block lengths in order to approach the Shannon limit. Also in hardware implementations the MAP calculations would be normally performed in the log domain to reduce complexity and improve numerical behavour (when using finite precision arithmetic).

3.1 Encoder

We are given the information databits as:

$$\{X_1, X_2, X_3, X_4, X_5, X_6\} = \{1, 1, 0, 0, 1, 0\}$$

These bits are input to a convolutional encoder #1. The resultant path through the trellis is shown below in fig 3.1.

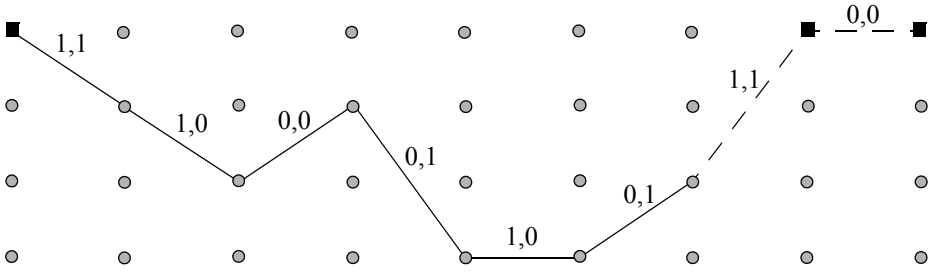


Figure 3.1 Encoder #1 Trellis Path (X_k,P_{1k})

The two bits required to park the trellis state at state 0 are:

$$\{X_7, X_8\} = \{1, 0\}$$

We now feed the block of data through a simple permuter.

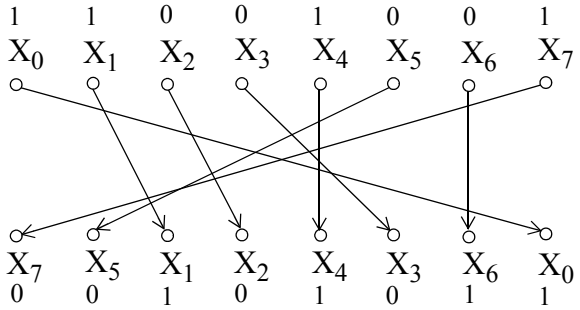


Figure 3.2 Pseudorandom Premutation

In our example we use a 7 element PN sequence with a 0 element appended.

The block of permuted databits are then fed into convolutional encoder #2 resulting in the path through the trellis shown below in fig 3.3.

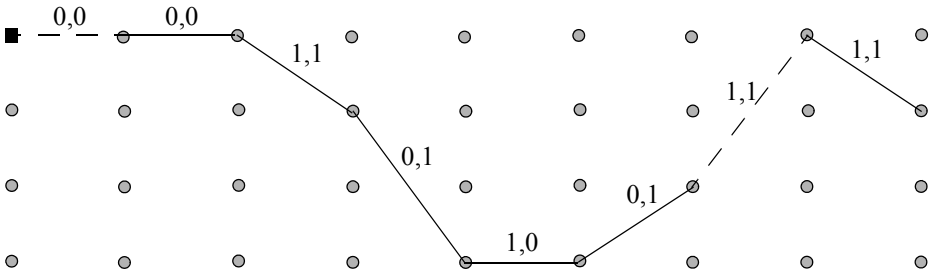


Figure 3.3 Encoder #2 Trellis Path (Perm(X_k), P_{2k})

The databits and the parity bits are mapped to symbols.

$$X_k, P_{1k}, P_{2k} \longrightarrow x_k, p_{1k}, p_{2k}$$

1, 1, 0	+1, +1, -1
1, 0, 0	+1, -1, -1
0, 0, 1	-1, -1, +1
0, 1, 1	-1, +1, +1
1, 0, 0	+1, -1, -1
0, 1, 1	-1, +1, +1
1, 1, 1	+1, +1, +1
0, 0, 1	-1, -1, +1

In our example we will assume the channel adds unity variance Gaussian noise with the following values:

x_k	p_{1k}	p_{2k}	+	AWGN			=	x'_k	p'_{1k}	p'_{2k}
+1	+1	-1		1.966099	2.132927	0.701887		2.966099	3.132927	-1.701887
+1	-1	-1		-1.232363	-0.443420	-0.696641		-0.232363	-1.443420	-1.696641
-1	-1	+1		0.750745	0.823265	0.823463		-0.249255	-0.176735	1.823463
-1	+1	+1		2.832447	-0.088392	1.036052		0.832447	0.911608	2.036052
+1	-1	-1		-1.262811	0.551007	-2.051227		-0.262811	-0.448993	-3.051227
-1	+1	+1		0.205224	0.277622	0.462560		-0.794776	1.277622	1.462560
+1	+1	+1		-0.569778	0.978633	1.105726		0.430222	1.978633	2.105726
-1	-1	+1		0.257169	0.465353	-0.700940		-0.742831	-0.534647	0.299060

3.2 Decoder

3.2.1 MAP Iteration #0

In the very first iteration no extrinsic information is available from MAP decoder #2, so the apriori information is set to 0.

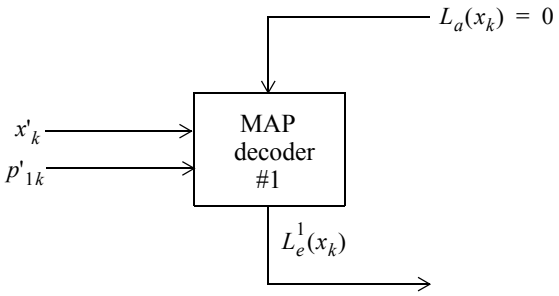


Figure 3.4 Iteration #1

The branch probabilities can be calculated from the noisy observations according to:

$$\gamma_k(s', s) = \exp\left[\frac{1}{2}(x_k L_a(x_k) + x_k L_c x'_k + p_k L_c p'_k)\right] \qquad L_c = \frac{2}{\sigma^2} = 2$$

Using the trellis diagram in fig 2.4 to determine x_k, p_k for the various state transitions we calculate the branch probabilities for the first stage:

$$\gamma_1(0, 0) = \exp(-x'_1 - p'_1) = \exp(-2.966099 - 3.132927) = 0.002245$$

$$\gamma_1(0, 1) = \exp(x'_1 + p'_1) = \exp(2.966099 + 3.132927) = 445.4235$$

$$\gamma_1(1, 2) = \exp(x'_1 - p'_1) = \exp(2.966099 - 3.132927) = 0.846345$$

$$\gamma_1(1, 3) = \exp(-x'_1 + p'_1) = \exp(-2.966099 + 3.132927) = 1.181551$$

$$\gamma_1(2, 0) = \exp(x'_1 + p'_1) = \exp(2.966099 + 3.132927) = 445.4235$$

$$\gamma_1(2, 1) = \exp(-x'_1 - p'_1) = \exp(-2.966099 - 3.132927) = 0.002245$$

$$\gamma_1(3, 2) = \exp(-x'_1 + p'_1) = \exp(-2.966099 + 3.132927) = 1.181551$$

$$\gamma_1(3, 3) = \exp(x'_1 - p'_1) = \exp(2.966099 - 3.132927) = 0.846345$$

Repeating for all stages of the trellis we get:

s', s	$\gamma_1(s', s)$	$\gamma_2(s', s)$	$\gamma_3(s', s)$	$\gamma_4(s', s)$	$\gamma_5(s', s)$	$\gamma_6(s', s)$	$\gamma_7(s', s)$	$\gamma_8(s', s)$
0,0	0.002245	5.342976	1.531106	0.174810	2.037665	0.617024	0.089918	3.587581
0,1	445.4235	0.187162	0.653123	5.720494	0.490758	1.620682	11.12121	0.278739
1,2	0.846345	3.357034	0.930048	0.923891	1.204642	0.125884	0.212586	0.812058
1,3	1.181551	0.297882	1.075214	1.082378	0.830122	7.943851	4.703989	1.231439
2,0	445.4235	0.187162	0.653123	5.720494	0.490758	1.620682	11.12121	0.278739
2,1	0.002245	5.342976	1.531106	0.174810	2.037665	0.617024	0.089918	3.587581
3,2	1.181551	0.297882	1.075214	1.082378	0.830122	7.943851	4.703989	1.231439
3,3	0.846345	3.357034	0.930048	0.923891	1.204642	0.125884	0.212586	0.812058

The forward recursion can be calculated according to:

$$\alpha_k(s) = \sum_{s'} \alpha_{k-1}(s') \cdot \gamma_k(s', s)$$

In our example the resultant values are:

s	$\alpha_0(s)$	$\alpha_1(s)$	$\alpha_2(s)$	$\alpha_3(s)$	$\alpha_4(s)$	$\alpha_5(s)$	$\alpha_6(s)$	$\alpha_7(s)$	$\alpha_8(s)$
0	1	0.002245	0.000027	0.599899	0.279384	0.272254	0.657920	5.329575	2.241861
1	0	445.4235	0.000001	1.406312	1.588757	0.442903	0.423994	1.436693	0.788824
2	0	0	3.57017	0.087631	0.636660	0.779049	2.510613	1.502256	0.197021
3	0	0	0.297881	0.075800	0.733850	0.680187	1.657465	0.447013	0.244644

We then normalise the forward probabilities by dividing by the sum of the forward probabilities across all 4 states at each stage resulting in:

s	$\alpha_0(s)$	$\alpha_1(s)$	$\alpha_2(s)$	$\alpha_3(s)$	$\alpha_4(s)$	$\alpha_5(s)$	$\alpha_6(s)$	$\alpha_7(s)$	$\alpha_8(s)$
0	1	0.000005	0.000007	0.276497	0.086265	0.125209	0.125318	0.611503	0.645632
1	0	0.999995	0	0.648177	0.490561	0.203690	0.080761	0.164843	0.227173
2	0	0	0.918491	0.040390	0.196582	0.358283	0.478213	0.172365	0.056740
3	0	0	0.081501	0.034937	0.226591	0.312817	0.315708	0.051289	0.070455

The backward recursion can be calculated according to

$$\beta_{k-1}(s') = \sum_{s'} \gamma_k(s', s) \cdot \beta_k(s)$$

In our example the resultant values are:

s	$\beta_8(s)$	$\beta_7(s)$	$\beta_6(s)$	$\beta_5(s)$	$\beta_4(s)$	$\beta_3(s)$	$\beta_2(s)$	$\beta_1(s)$	$\beta_0(s)$
0	1	3.587581	0.083436	0.007095	0.024528	2.202316	1.069152	2.676299	102.2741
1	0	0	0.015326	0.371193	0.791577	0.645871	0.206642	1.094823	0.204413
2	0	0.278739	10.31944	0.013449	0.094789	0.135413	0.684043	0.606906	250.0070
3	0	0	0.339130	7.624454	1.147166	0.564830	0.189074	0.390194	0.219648

We normalise the backward probabilities by dividing by the sum of the backward probabilities across all 4 states at each stage resulting in:

s	$\beta_8(s)$	$\beta_7(s)$	$\beta_6(s)$	$\beta_5(s)$	$\beta_4(s)$	$\beta_3(s)$	$\beta_2(s)$	$\beta_1(s)$	$\beta_0(s)$
0	1	0.927906	0.007756	0.000885	0.011918	0.62 0645	0.4975 2	0.561278	0.289971
1	0	0	0.001425	0.046305	0.384623	0.18 2016	0.0961 1	0.229608	0.000580
2	0	0.072094	0.959294	0.001678	0.046058	0.03 3161	0.3183 1	0.127281	0.708827
3	0	0	0.031526	0.951132	0.557402	0.15 9177	0.0879 6	0.081832	0.000623

Fig summarises the overall results for the trellis associated with encoder # .

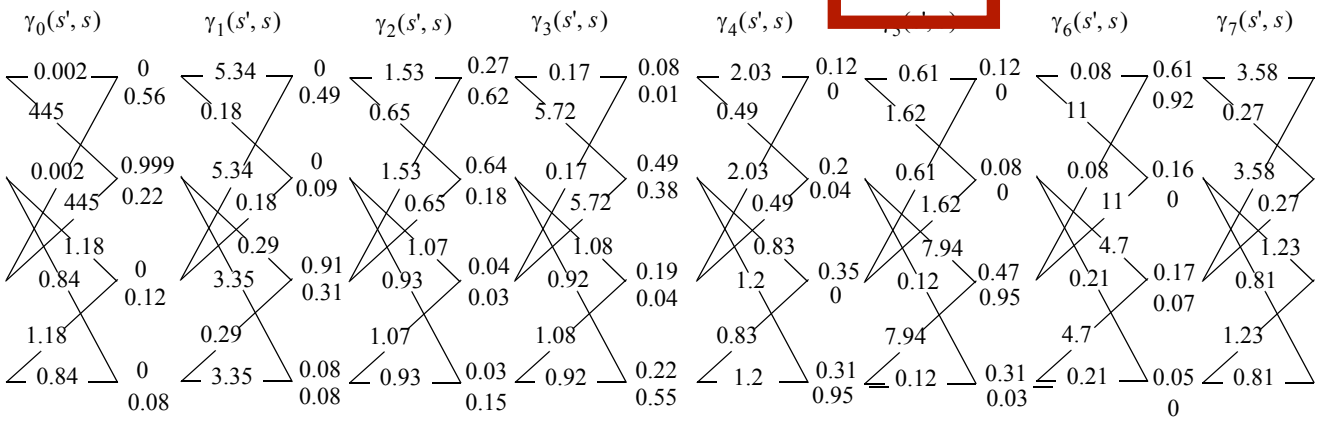


Figure 3.5 Iteration #1 - Annotated trellis for MAP decoder #1

We now compute the soft decisions of the MAP decoder according to:

$$L_{map}(x_k) = \log\left[\frac{\sum_{(s',s)\in S^+} \alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}{\sum_{(s',s)\in S^-} \alpha_{k-1}(s')\cdot\gamma_k(s',s)\cdot\beta_k(s)}\right]$$

$$= \frac{\alpha_{k-1}(0)\cdot\gamma_k(0,1)\cdot\beta_k(1)+\alpha_{k-1}(1)\cdot\gamma_k(1,2)\cdot\beta_k(2)+\alpha_{k-1}(2)\cdot\gamma_k(2,0)\cdot\beta_k(0)+\alpha_{k-1}(3)\cdot\gamma_k(3,3)\cdot\beta_k(3)}{\alpha_{k-1}(0)\cdot\gamma_k(0,0)\cdot\beta_k(0)+\alpha_{k-1}(1)\cdot\gamma_k(1,3)\cdot\beta_k(3)+\alpha_{k-1}(2)\cdot\gamma_k(2,1)\cdot\beta_k(1)+\alpha_{k-1}(3)\cdot\gamma_k(3,2)\cdot\beta_k(2)}$$

The soft and hard decisions are:

$L_{map}(x_k)$	$\hat{X}_k = sign[L_{map}(x_k)]$
11.304209	1
3.707485	1
0.393571	1
0.505558	1
-0.436344	0
-4.375969	0
3.737708	1
-3.821260	0

Comparing with the original databits we see that using the hard decisions from the first iteration would result in 3 bit errors.

The extrinsic information output from the first iteration can be calculated by subtracting out the apriori and signal components.

$L_{map}(x_k)$	$-[\ L_a(x_k)$	$+\ L_c \cdot x'_k\ ]$	$=\ L_e(x_3)$
11.304209	0	5.932198	5.372011
3.707485	0	-0.464725	4.172210
0.393571	0	-0.498510	0.892081
0.505558	0	1.664894	-1.159337
-0.436344	0	-0.525622	0.089278
-4.375969	0	-1.589551	-2.786418
3.737708	0	0.860444	2.877265
-3.821260	0	-1.485662	-2.335598

3.2.2 MAP Iteration #1

In the second iteration the extrinsic information from MAP decoder #1 is used as the apriori information for the MAP decoder #2. The latter log likelihood ratios are permuted along with the noisy databits to match the order which was originally used in encoder #2. The effective relabelling of the inputs to decoder #2 is illustrated below in fig 3.6.

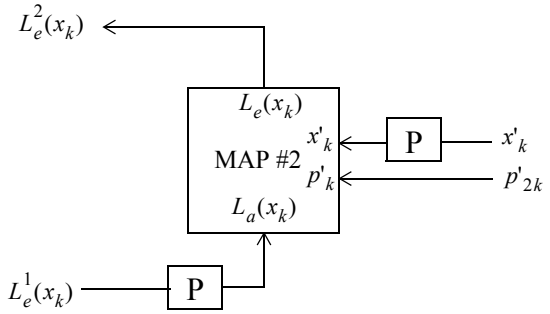


Figure 3.6 Iteration #2

After the relabelling we have the following inputs to MAP decoder #2.

$L_a(x_k)$	x'_k	p'_k
-2.335598	-0.742831	-1.701887
-2.786418	-0.794776	-1.696641
4.172210	-0.232363	1.823463
0.892081	-0.249255	2.036052
0.089278	-0.262811	-3.051227
-1.159337	0.832447	1.462560
2.877265	0.430222	2.105726
5.372011	2.966099	0.299060

Using the trellis diagram in fig 2.4 to determine x_k, p_k for the various state transitions we calculate the branch probabilities for the first stage:

$$\gamma_1(0,0) = \exp((-L_a(x_1)/2 - x'_1) - p'_1) = \exp(2.335598/2 + 0.742831 + 1.701887) = 37.059215$$

$$\gamma_1(0,1) = \exp(+L_a(x_1)/2 + x'_1 + p'_1) = \exp(-2.335598/2 - 0.742831 - 1.701887) = 0.026984$$

$$\gamma_1(1,2) = \exp(+L_a(x_1)/2 + x'_1 - p'_1) = \exp(-2.335598/2 - 0.742831 + 1.701887) = 0.811604$$

$$\gamma_1(1,3) = \exp(-L_a(x_1)/2 - x'_1 + p'_1) = \exp(2.335598/2 + 0.742831 - 1.701887) = 1.232128$$

$$\gamma_1(2,0) = \exp(+L_a(x_1)/2 + x'_1 + p'_1) = \exp(-2.335598/2 - 0.742831 - 1.701887) = 0.026984$$

$$\gamma_1(2,1) = \exp(-L_a(x_1)/2 - x'_1 - p'_1) = \exp(2.335598/2 + 0.742831 + 1.701887) = 37.059215$$

$$\gamma_1(3,2) = \exp(-L_a(x_1)/2 - x'_1 + p'_1) = \exp(2.335598/2 + 0.742831 - 1.701887) = 1.232128$$

$$\gamma_1(3,3) = \exp(+L_a(x_1)/2 + x'_1 - p'_1) = \exp(-2.335598/2 - 0.742831 + 1.701887) = 0.811604$$

Repeating for all stages of the trellis we get:

s', s	$\gamma_1(s', s)$	$\gamma_2(s', s)$	$\gamma_3(s', s)$	$\gamma_4(s', s)$	$\gamma_5(s', s)$	$\gamma_6(s', s)$	$\gamma_7(s', s)$	$\gamma_8(s', s)$
0,0	37.05921	48.64874	0.025294	0.107224	26.29555	0.179903	0.018787	0.002603
0,1	0.026984	0.020556	39.53576	9.326298	0.038029	5.558561	53.22778	384.2007
1,3	1.232128	1.634510	0.970175	6.291923	0.058833	3.352752	1.267279	0.004734
1,2	0.811604	0.611804	1.030742	0.158934	16.99731	0.298262	0.789092	211.2503
2,1	37.05921	48.64874	0.025294	0.107224	26.29555	0.179903	0.018787	0.002603
2,0	0.026984	0.020556	39.53576	9.326298	0.038029	5.558561	53.22778	384.2007
3,2	1.232128	1.634510	0.970175	6.291923	0.058833	3.352752	1.267279	0.004734
3,3	0.811604	0.611804	1.030742	0.158934	16.99731	0.298262	0.789092	211.2503

Performing the forward recursion and normalising at each stage gives:

s	$\alpha_0(s)$	$\alpha_1(s)$	$\alpha_2(s)$	$\alpha_3(s)$	$\alpha_4(s)$	$\alpha_5(s)$	$\alpha_6(s)$	$\alpha_7(s)$	$\alpha_8(s)$
0	1	0.999272	0.999544	0.000648	0.000028	0.000096	0.006442	0.987545	0.003026
1	0	0.000728	0.000422	0.999329	0.000937	0.037461	0.000354	0.007571	0.991757
2	0	0	0.000009	0.000012	0.024624	0.004238	0.880803	0.003006	0.004181
3	0	0	0.000024	0.000011	0.974411	0.958204	0.112401	0.001878	0.001037

Performing the backward recursion and normalising at each stage gives:

s	$\beta_8(s)$	$\beta_7(s)$	$\beta_6(s)$	$\beta_5(s)$	$\beta_4(s)$	$\beta_3(s)$	$\beta_2(s)$	$\beta_1(s)$	$\beta_0(s)$
0	0.25	0.322611	0.341684	0.035868	0.052754	0.490631	0.422588	0.931880	0.979359
1	0.25	0.177389	0.017336	0.057222	0.411621	0.364305	0.004247	0.016093	0.001702
2	0.25	0.322611	0.621132	0.432317	0.084044	0.068420	0.568904	0.009759	0.017625
3	0.25	0.177389	0.019848	0.474593	0.451581	0.076643	0.004262	0.042268	0.001314

We again compute the soft decisions of the MAP decoder and seperate out the extrinsic information:

$L_{map}(x_k)$	$-[\ L_a(x_k)$	$+\ L_c \cdot x'_k\ ]$	$= L_e(x_k)$
-11.283877	-2.335598	-1.485662	-7.462618
-11.007680	-2.786418	-1.589551	-6.631711
7.054060	4.172210	-0.464725	3.346575
-5.188530	0.892081	-0.498510	-5.582101
4.845146	0.089278	-0.525622	5.281491
-4.571122	-1.159337	1.664894	-5.076680
5.737180	2.877265	0.860444	1.999472
11.890362	5.372011	5.932198	0.586153

If we slice the soft decisions we get the hard decisions 00101011. Comparing this with the encoder #2 trellis path in fig 3.3 we see that the decoder has correctly estimated all databits.

3.2.3 Further MAP Iterations

The following table summarises the extrinsic information generated during the first 8 MAP iterations.

#0	#1	#2	#3	#4	#5	#6	#7
$L_e^1(x_k)$	$L_e^2(x_k)$	$L_e^1(x_k)$	$L_e^2(x_k)$	$L_e^1(x_k)$	$L_e^2(x_k)$	$L_e^1(x_k)$	$L_e^2(x_k)$
5.372011	-7.462618	14.356022	-20.020202	17.392653	-20.069679	17.393598	-20.069679
4.172210	-6.631711	10.686105	-18.196690	18.623659	-18.247587	18.671608	-18.247588
0.892081	3.346575	-9.218456	18.149927	-19.978938	18.631031	-20.027948	18.631031
-1.159337	-5.582101	-11.293812	-13.189756	-14.852595	-13.697800	-14.852997	-13.697800
0.089278	5.281491	8.702221	13.189756	21.371859	13.237730	21.445320	13.237731
-2.786418	-5.076680	-10.895412	-7.733734	-19.724002	-7.734682	-19.759764	-7.734682
2.877265	1.999472	12.633857	4.809138	15.302263	4.809561	15.303211	4.809561
-2.335598	0.586153	-12.029344	0.598121	-21.083360	0.598121	-21.130653	0.598121

Inthe case of our example the extrinsic information converges to a steady state solution by the 4th iteration.

4.0 References

[1] S.A. Barbulescu, "Iterative Decoding of Turbo Codes and Other Concatenated Codes"

http://www.itr.unisa.edu.au/~steven/thesis/sab.ps.gz

[2] W.E. Ryan,"A Turbo Code Tutorial" New Mexico State University

http://vada.skku.ac.kr/ClassInfo/digital-com2000/slides/turbo2c.pdf

[3] M.C. Reed, S.S. Pietrobon, "Turbo-code termination schemes and a novel alternative for short frames" IEEE Int. Symp. on Personal, Indoor and Mobile Radio Commun., Taipei, Taiwan, pp. 354-358, Oct. 1996.

http://www.itr.unisa.edu.au/~steven/turbo/PIMRC96.ps.gz

[4] J. Fei, “On a Turbo Coder Design for Low Power Dissipation“

http://scholar.lib.vt.edu/theses/available/etd-07212000-03370015/unrestricted/total.pdf