Intuitive Guide to Principles of Communications
www.complextoreal.com

Turbo decoding using the MAP algorithm

Part 2 – A Step by step example

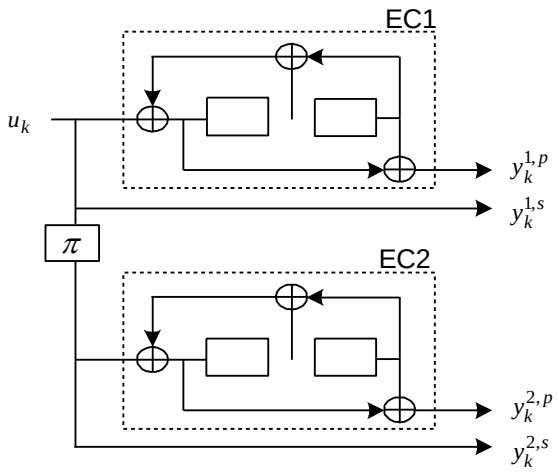

Figure 1 – A rate 1/3 PCCC Turbo code

In this example, we will use a rate 1/3 Turbo Code which has two identical convolutional codes. Note that since $y_k^{2,s}$ is a deterministically reshuffle version of $y_k^{1,s}$, it is not transmitted. The second decoder is given $y_k^{1,s}$, and it then de-interleaves it to get its copy of $y_k^{2,s}$ which it needs for decoding.

The coding trellis for each is given by the figure below. The blue lines show transitions in response to a 0 and red lines in response to a 1. The notation 1/11, the first number is the input bit, the next are two code bits. Of these, the first is what we called the **systematic bit**, and as you can see, it is same as the input bit. The second bit is the parity bit. Each code uses this trellis for encoding.

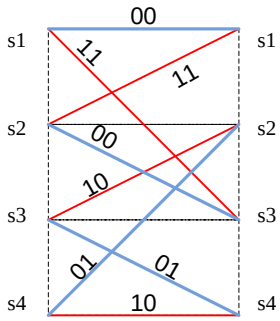

Figure 2 – the trellis diagram of the rate 1/2 code RSC code

For the example, we pick the information bit sequence: 1, 0, 1, 0, 1, 0, 0, which in bi-polar form is shown in Table 1. Code 1 encodes each of these bits as systematic and parity bits using the trellis in Fig. 7. The bits are coded in a bipolar format.

Table 1 – Coded bits through first Encoder

Code 1	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Systematic	**1**	**-1**	**1**	**-1**	**1**	**-1**	**-1**
Parity bit, ENC1	1	1	-1	1	1	-1	-1

The sequence looks like this on the trellis diagram.

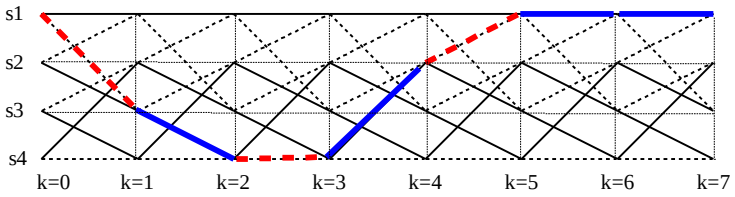

Figure 3 – Trellis diagram for information (1, -1, 1, -1, 1, -1, -1) through first encoder

The code 2 receives the same information bits but interleaved by a given pattern. Interleaving is an important issue in turbo coding. There are several different types of interleaving patterns other than the row column method you know. Here I have made up a "pseudo-random" interleaving pattern shown in Table 2. The deinterleaving pattern puts the bits back in the original order. Simulations shows that pseudorandom interleaving works best with turbo codes in AWGN channel.

Table 2 – Interleaving patter from Encoder 1 to Encoder 2 and then deinterleaving back to Encoder 1 from Encoder 2

Interleaving 1-2	1-3	2-4	3-1	4-5	5-2	6-6	7-7
Deinterleaving 2-1	1-3	2-5	3-1	4-2	5-4	6-6	7-7

After reordering the information sequence **(1, -1, 1, -1, 1, -1, -1)** into (1, 1, 1, -1, -1, -1, -1) Encoder 2 codes the new sequence as in Table 3.

Table 3 – Interleaved bits coded by Encoder 2

Code 2	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Systematic	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**
Parity bit, ENC2	1	-1	1	-1	-1	-1	-1

On the trellis diagram this new sequence is coded as shown in Figure 8.

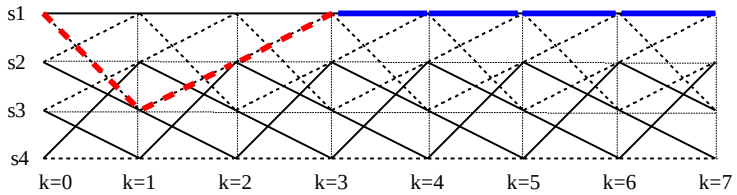

Figure 4 – Trellis diagram for information (1, 1, 1, -1, -1, -1, -1) through second encoder

Note that the systematic bits from Code 2 are redundant and will not be transmitted. This gives us a rate 1/3 code if you don't count the tail bits. The transmitted data is given in columns or: 1 1 1 -1 1-1 1-1 1 1 …

Table 4 – Transmitted data, one systematic bit and two parity bits from each encoder

Transmitted	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Systematic	**1**	**-1**	**1**	**-1**	**1**	**-1**	**-1**
Parity bit, ENC1	1	1	-1	1	1	-1	-1
Parity bit, ENC2	1	-1	1	-1	-1	-1	-1

That's all for the encoding. On the receiving end, assume that this signal goes through a AWGN channel with signal Es/N0 = 0.25, which is quite small. A bunch of errors popup and the received data with some channel gain is received as shown in Table 5.

Table 5 – Received signal amplitudes (soft data)

Received Data	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Systematic bit	2	1	3	-2	2	-4	-5
Parity bit	-5	2	-1	-2	1	-2	-1
Parity bit	6	-1	2	-2	-5	5	-6

These values are called soft inputs. Although the above are integers, they don't have to be. I happen to pick integers for this example.

We have total of four errors. One in the systematic bits and three in parity bits.

Unlike Viterbi and other codes, in MAP algorithm channel SNR effects decoding and it does that by exaggerating the effect of the systematic bits. So if there is an error in the systematic bit, it also gets exaggerated. In Table 5, I have assumed that the second systematic bits is in error, as well as the first and fourth parity bits. The term L_c is the measure of the signal SNR.

L_c = 4 Es/N0 = 4 x 0.25 = 1.0

Multiply the received signal by L_c which is defined above and is equal to 1.0 for this example. L_c is indicates the condition of the channel. High is good.

Table 6 – Input signal level multiplied by channel L_c

DEC1 Received Data	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$Lc \cdot y_k^{1,s}$	2	1	3	-2	2	-4	-5
$Lc \cdot y_k^{2,p}$	-5	2	-1	-2	1	-2	-1
DEC2 Received Data	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$Lc \cdot y_k^{1,s}$	3	2	2	1	-2	-4	-5
$Lc \cdot y_k^{2,p}$	6	-1	2	-2	-5	5	-6

Decoding

For each time tick k, decoding is done by calculating the L-values of a +1 bit. If it is positive, the decision is in favor of a +1. Calculation of the L-values or $L(u_k)$ is quite a complex process. The main equation used to calculate the L-value is this.

$$L(u_k) = \left[L^e(u_k) + Lc \cdot y_k^{1,s}\right] + \log \frac{\sum_{u^+} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s', s)}{\sum_{u^-} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s', s)} \qquad (1)$$

From this computation if $L(u_k)$ is positive, then u_k is equal to +1.

The first term, $L^e(u_k)$ is the a-priori value from Decoder 2. This is the L-value of the a-priori probability of the bit in question. At first, the decoder has no idea what it is. A good guess is to assume it is 0.5. The second term, $L_c y_k^{1,s}$ is computed by multiplying the systematic information with L_c as in Table 6. This value is the **channel L-value** and gives an indication of channel SNR. The third big term with all kinds of squiggly items is the a-posteriori probability. This number is calculated for

each trellis segment. Remember that we can only have one result for a trellis section, a +1 or -1. The $L(u_k)$ calculation tells us what that number is,

The big equation can be written simply as sum of three pieces of information.

L-apriori – This is our initial guess about a +1 bit in first iteration

L-Channel - This is related to channel SNR and systematic bit and is equal to $Lc \cdot y_k^{1,s}$

L_e – the computed information in each iteration is called the a-posteriori L-value.

Eq. 1 can be now written as the sum of these as

$$= L\_apriori + L\_channel + L^e(u_k) \qquad (2)$$

The L-channel value does not change from iteration to iteration since is it given by $L\_channel = L_c y_k^{1,s}$. Neither the L_c nor the systematic bit changes from iteration to iteration, So lets called it K. The only two items that change are the a-priori and a-posteriori L-values.

$$L(u_k) = L\_apriori + K + L\_posteriori$$

A-priori value goes in, it is used to compute the new a-posteriori probability and then it can be used to compute $L(u_k)$ or is passed to the next decoder. Although in the example we compute $L(u_k)$ each time, during actual decoding this is not done. Only a-posteriori metric is computed and decoders and keep doing this either a fixed number of times or until it converges. L-posteriori is also called **Extrinsic information**.

Iteration 1

$$L_1(u_k) = 0 + K_1 + L^e_{1,1}$$
$$L_2(u_k) = L^e_{1,1} + K_2 + L^e_{1,2}$$

Iteration2

$$L_1(u_k) = L^e_{1,2} + K_1 + L^e_{2,1}$$
$$L_2(u_k) = L^e_{2,1} + K_2 + L^e_{2,2}$$

Iteration 3

$$L_1(u_k) = L^e_{2,2} + K_1 + L^e_{3,1}$$
$$L_2(u_k) = L^e_{3,1} + K_2 + L^e_{3,2}$$

So the whole objective for the calculations is to compute the extrinsic L-value, keep an eye on them and when they get big, then compute L(uk) and make a decision. The computation

of these L-values uses an algorithm called Forward-Backward Recursion and this is of course another name for BCJR and many others.

In equation, we have three terms in the ratio that need to be computed. They are

$$\tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s', s)$$

The first term is called the forward metric. It is computed by a forward recursion using the last term called branch metrics. It is given by

$$\alpha_k = \sum_s \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s', s)$$

This is a recursive process. The forward branch metric is the product of previous forward metric times the branch metric. (The reason we call these metrics is that these are not really probabilities in a strict sense, just a measure of probability.) So to compute the forward metric, we first need to compute branch metrics.

In real situations, if the block is 10,000 bits long, then we would need to compute these metrics for 10000 times the number of states times 2. The example here has just 7 bits. It has 4 states so we will do this 7 x 4 x 2 = 56 times.

A branch metric is the correlation of the received signal with its trellis values. In a nutshell, if the received values are the same sign as the coded bits, then the correlation will be high. For each decoder, there are full transition branch metrics which incorporate, the systematic and the parity bits and partial branch metrics which incorporate only the parity bits.

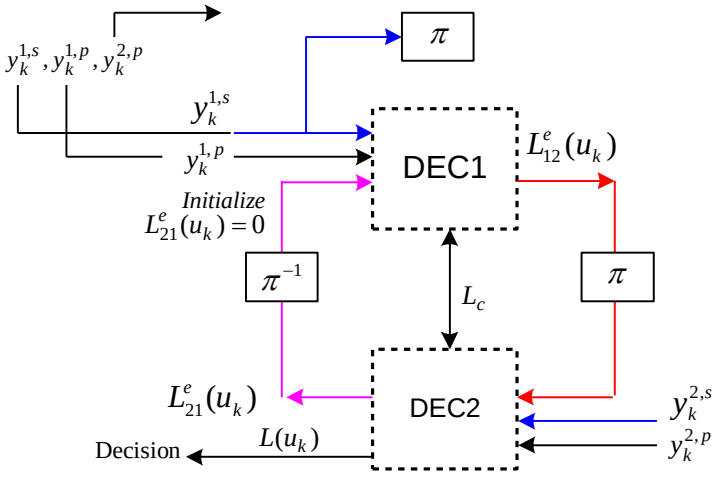

Figure 5 – Iteration cycle of MAP Turbo Decoding

1. Computing full branch metrics

The full branch metric is given by the equation

$$\gamma(s', s) \propto \exp\left[\frac{1}{2} \cdot L^e(c_k^1) \cdot c_k^1 + Lc \cdot \frac{1}{2} \cdot y_k^{1,s} \cdot c_k^1\right] \exp\left[\sum_{i=2}^{q} \left(Lc \cdot \frac{1}{2} \cdot y_k^{i,p} \cdot c_k^i\right)\right]$$

The data required is L_c, the systematic value, parity value and trellis coded values. We compute this for each branch of the trellis for all seven time ticks.

Example: For branch 1-1 (from state1 to state1), k = 1,

$y_1^{1,s} = 2,\ y_1^{1,p} = -5,\ L_e = 0$

What is L_e and why is it zero? L_e is our initial L-value. In the start, we do not know if the encountered bit was a 0 or a 1, so we assume that the probability of a 0 is equal to a probability of 1. The L-value is given by

$$L(u_k) = \ln \frac{P(u_k = +1)}{P(u_k = -1)}$$

If 0 and 1 are equally likely, a safe bet, the then log of 1.0 is zero. And that is what L_e is. It is the initial value of the Extrinsic L-value. The Extrinsic L-value is the key information that the decoders share. It works something like this.

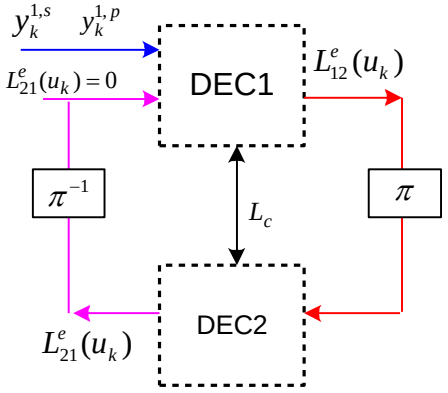

Figure 6 – Feedback nature of the Extrinsic, value. The value produced by each decoder1 ($L_{12}^e(u_k)$) becomes the input for the next decoder.

This thing – Extrinsic L-value is what goes around from one code to the next, It is also called the a-posteriori probability at the output of a decoder and becomes the apriori probability for the next decoder in a feedback manner going round and round and getting larger each iteration.. Nice thing about L-values is that large means better reliability.

Computing Full branch metrics requires five inputs, shown in columns 1-5 below. Branch value is calculated using the equation

Table 7 – Computing Full branch metrics

c_k^1	c_k^2	Le	$Lc \cdot y_k^{1,s}$	$Lc \cdot y_k^{2,p}$	$\gamma(s',s)$
-1	**-1**	0	2	-5	0.0302
-1	**-1**	0	2	-5	0.0302

-1	**1**	0	2	-5	4.4817
-1	**1**	0	2	-5	4.4817
1	**1**	0	2	-5	33.1155
1	**1**	0	2	-5	33.1155
1	**-1**	0	2	-5	0.2231
1	**-1**	0	2	-5	0.2231

Table 8 – Full branch metrics for full length of the signal, by DEC1

$\gamma(s',s)$	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$\gamma(1,1)$	0.0302	0.2231	0.3679	7.3891	0.2231	20.0855	20.0855
$\gamma(2,3)$	0.0302	0.2231	0.3679	7.3891	0.2231	20.0855	20.0855
$\gamma(3,4)$	4.4817	1.6487	0.1353	1.0000	0.6065	2.7183	7.3891
$\gamma(4,2)$	4.4817	1.6487	0.1353	1.0000	0.6065	2.7183	7.3891
$\gamma(1,3)$	33.1155	4.4817	2.7183	0.1353	4.4817	0.0498	0.0498
$\gamma(2,1)$	33.1155	4.4817	2.7183	0.1353	4.4817	0.0498	0.0498
$\gamma(3,2)$	0.2231	0.6065	7.3891	1.0000	1.6487	0.3679	0.1353
$\gamma(4,4)$	0.2231	0.6065	7.3891	1.0000	1.6487	0.3679	0.1353

2. Compute partial branch metrics

These values are based only on parity bits. It is last part in Eq. 1. We will use these to compute the total extrinsic L-values.

The equation for calculating the partial metrics is given by

$$\gamma^{e}(s',s) \triangleq \exp\left[\sum_{i=2}^{q}\left(Lc\cdot\frac{1}{2}\cdot y_k^{i,p}\cdot c_k^{i}\right)\right] \qquad (3)$$

Note that there is nothing here that changes from iteration to iteration, so these only need to be computed once.

Table 9 – Partial branch metrics stay constant for each decoder

$\gamma^{e}(s',s)$	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$\gamma^{e}(1,1)$	0.0821	0.3679	1.6487	2.7183	0.6065	2.7183	1.6487
$\gamma^{e}(2,3)$	0.0821	0.3679	1.6487	2.7183	0.6065	2.7183	1.6487
$\gamma^{e}(3,4)$	12.1825	2.7183	0.6065	0.3679	1.6487	0.3679	0.6065
$\gamma^{e}(4,2)$	12.1825	2.7183	0.6065	0.3679	1.6487	0.3679	0.6065
$\gamma^{e}(1,3)$	12.1825	2.7183	0.6065	0.3679	1.6487	0.3679	0.6065
$\gamma^{e}(2,1)$	12.1825	2.7183	0.6065	0.3679	1.6487	0.3679	0.6065
$\gamma^{e}(3,2)$	0.0821	0.3679	1.6487	2.7183	0.6065	2.7183	1.6487
$\gamma^{e}(4,4)$	0.0821	0.3679	1.6487	2.7183	0.6065	2.7183	1.6487

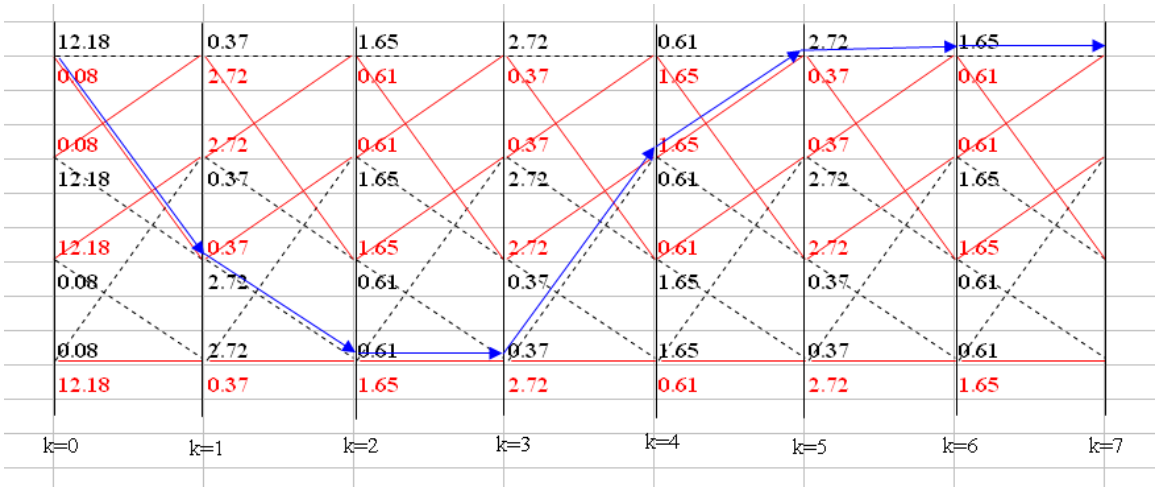

Figure 7 - Partial branch metrics on the trellis

Calculation of forward metrics

Encoding always starts in state 1. So we assume that signal can only be in state 1 at time k =0. The initial value of the forward metric is set at 1.0. The other three states are given value of 0.0. Meaning, these are not possible states. Thereafter the forward metrics are computed recursively for each state (not branches).

The equation is

$$\tilde{\alpha}_k(s) = \frac{\sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}{\sum_{s}\sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)} \qquad (4)$$

Computation of forward metrics will be done in two steps, first we will compute the numerator at time k, then we will compute the denominator.

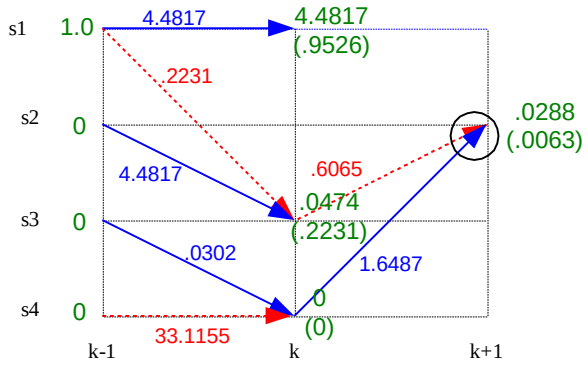

Figure 8 – Computing forward metric

We will compute the forward metric at state 3, k = 1 and then at state 2, k =2. Here is how you think about this. There are two branches converging into state 3 at time k =1. Each of these brings its full branch metric to bear on the converging state. Computation involves multiplying the branch metrics with the starting state value and then summing these for all converging branches at the state in question. This is two branches. We do this for each state based on the specific branches that are coming into it. We can think of as the amount of traffic these roads are bringing to the intersection.

We compute the forward state metric at state 3 as

α_1^3 = 1.0 x 0.2231 + 0.0 x 4.4817 = 0.2231

The only other non-zero forward metric at this point is state 1, which is

α_1^1 = 1.0 x 4.4817 + 0.0 x 0.2231 = 4.4817

Before we continue on to k = 2, we need to normalize these values, which is denominator in Eq . The purpose of normalization is to control the numerical overflow which may occur due to the large number of numbers that are being multiplied together. Before we proceed, we normalize the values at each time. This gives them more of a flavor of probabilities and makes it easier to understand and reduces the numerical overload.

To normalize, we use the sum of forward metrics which is 4.7048. Normalizing the two metrics with this sum, gives us the two values in col 3, Table.

Now to move forward to state 2, it has two branches (from s3 and from s4) converging into it. Its forward metric is calculated by

α_1^1 = 0.0474 x 0.6065 + 0.0 x 0.0302 = 0.0288

We compute these for all four states and we get the following values. The column is the added to compute the normalizing sum. It is then used in Table to normalize all forward metrics.

Table 10 – Computation of forward metrics

$\tilde{\alpha}(s)$	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$\tilde{\alpha}(1)$	1	0.0302	0.0002	0.7297	0.8538	1.4908	7.8900	14.3567
$\tilde{\alpha}(2)$	0	0.0000	0.6060	0.1121	0.8671	0.2806	0.5400	0.7589
$\tilde{\alpha}(3)$	0	33.1155	0.0041	0.0990	0.1464	1.4702	1.5040	1.0180
$\tilde{\alpha}(4)$	0	0.0000	1.6472	5.3918	0.8671	0.5553	1.1063	1.0201
Denominator	1	33.1456	2.2575	6.3326	2.7344	3.7969	11.0403	17.1537

Table 11 – Normalized forward state metrics

$\tilde{\alpha}(s)$	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$\tilde{\alpha}(1)$	1	0.0009	0.0001	0.1152	0.3123	0.3926	0.7147	0.8369
$\tilde{\alpha}(2)$	0		0.2684	0.0177	0.3171	0.0739	0.0489	0.0442
$\tilde{\alpha}(3)$	0	0.9991	0.0018	0.0156	0.0535	0.3872	0.1362	0.0593
$\tilde{\alpha}(4)$	0		0.7297	0.8514	0.3171	0.1462	0.1002	0.0595

3. Computing backward state metrics

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_s \tilde{\beta}_k(s)\gamma_k(s',s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-2}(s')\gamma_{k-1}(s',s)} \qquad (5)$$

We assume that the signal will end in state 1. (Tail bits are added to force it end in state 1.) The ending state is always s1, so it gets a value of 1.0 just as the forward state metric. The rest of the three states at k = 7 are given a backward state value of zero. Look at how these propagate backwards.

$$\sum_s \tilde{\beta}_k(s)\gamma_k(s',s)$$

We will compute the backward state metric at state 3. There are two branches coming into state s3 at k = 5. So we will compute their backward metrics first. For state s2, it has two branches coming into it, 2-1 and 2-3.

β_7^2 = 1.0 x 0.0498 + 0.0 x 1.6487 = 0.0498

β_7^4 = 0 (It has two branches coming into it but both have starting backward metrics of 0 so, it is zero as well.)

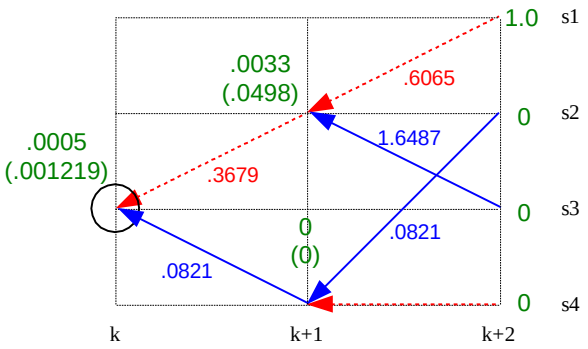

Figure 9 – Computing backward state metrics

Compute all four as shown at k = 7. Now again, we normalize. But we normalize with the same set of numbers as the forward metrics so they are both on the same footing. Divide each backward state metric after computation and normalize before proceeding to the next time tick.

Table 12 –Backward state metrics

$\tilde{\beta}(s)$	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$\tilde{\beta}(1)$	1.8773	7.1116	5.8102	2.1494	36.5413	20.0855	1
$\tilde{\beta}(2)$	14.1768	3.4108	0.2180	43.1318	0.0906	0.0498	0
$\tilde{\beta}(3)$	14.3680	0.5916	15.7811	0.0413	0.001659	0.0000	0
$\tilde{\beta}(4)$	7.4396	18.4184	15.7811	0.0198	0.0123	0.0000	0

Table 13 –Normalized Backward state metrics

$\tilde{\beta}(s)$	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$\tilde{\beta}(1)$	0.0566	3.1502	0.9175	0.7861	9.6240	1.8193	1
$\tilde{\beta}(2)$	0.4277	1.5109	0.0344	15.7739	0.0239	0.0045	0
$\tilde{\beta}(3)$	0.4335	0.2621	2.4920	0.0151	0.0004		0
$\tilde{\beta}(4)$	0.2245	8.1588	2.4920	0.0072	0.0032		0
Denominator	33.1456	2.2575	6.3326	2.7344	3.7969	11.0403	17.1537

4. Computing Extrinsic L-values

Now we will compute the Extrinsic L-value. Which are given by the product of the all three metrics.

$$\sigma_k(s) = \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s)$$

This is multiplication of the forward, backward the partial branch metric for each branch. We get one value for each branch, or eight values per time as shown below.

Table 14 – Computation of final branch metrics

$\sigma(s)$	k=1	k=2	k=3	k=4	k=5	k=6	k=7	Branch	Product of
$\sigma(1)$	0.0046	0.0011	0.0001	0.2462	1.8227	1.9417	1.1783	11	$\tilde{\alpha}(1)\,\gamma^e(1,1)\,\tilde{\beta}(1)$
$\sigma(2)$	0.0000	0.0000	1.1029	0.0007	0.0001	0.0000	0.0000	23	$\tilde{\alpha}(2)\,\gamma^e(2,3)\,\tilde{\beta}(2)$
$\sigma(3)$	0.0000	22.1578	0.0027	0.0000	0.0003	0.0000	0.0000	34	$\tilde{\alpha}(3)\,\gamma^e(3,4)\,\tilde{\beta}(3)$
$\sigma(4)$	0.0000	0.0000	0.0152	4.9408	0.0125	0.0002	0.0000	42	$\tilde{\alpha}(4)\,\gamma^e(4,2)\,\tilde{\beta}(4)$
$\sigma(1)$	5.2809	0.0006	0.0001	0.0006	0.0002	0.0000	0.0000	13	$\tilde{\alpha}(1)\,\gamma^e(1,3)\,\tilde{\beta}(1)$
$\sigma(2)$	0.0000	0.0000	0.1494	0.0051	5.0315	0.0495	0.0297	21	$\tilde{\alpha}(2)\,\gamma^e(2,1)\,\tilde{\beta}(2)$
$\sigma(3)$	0.0000	0.5553	0.0001	0.6703	0.0008	0.0047	0.0000	32	$\tilde{\alpha}(3)\,\gamma^e(3,2)\,\tilde{\beta}(3)$
$\sigma(4)$	0.0000	0.0000	2.9980	0.0168	0.0006	0.0000	0.0000	44	$\tilde{\alpha}(4)\,\gamma^e(4,4)\,\tilde{\beta}(4)$

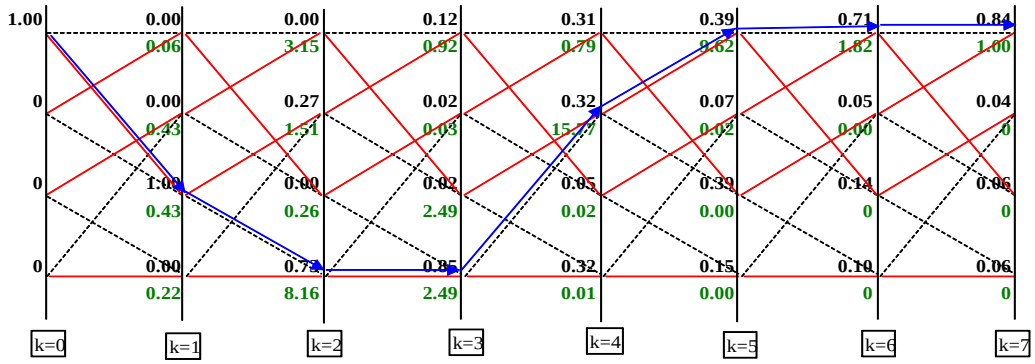

Figure 10 – Forward and backward state metrics for the trellis

To compute the L-value we need the ratio of these numbers for the +1 and -1 branches. For that we add the top four branch metrics and divide by the sum of the bottom four branch metrics to satisfy the following equation

$$\log \frac{\sum_{u^+} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s',s)}{\sum_{u^-} \tilde{\alpha}_{k-1}(s') \cdot \tilde{\beta}_k(s) \cdot \gamma_k^e(s',s)}$$

Take the natural log of the ratio. This is the extrinsic value output for this iteration. These values after interleaving go to Decoder 2 and become it's a-priori probabilities.

Normally the decoder at this point would stop because it has done its job of computing the extrinsic value. But I will compute the L-value of the bit to show you how the bit decisions can be made at any iteration.

The L-value of the bit is given by

$$= L\_apriori + L\_channel + L^e(u_k)$$

At each iteration, we have all three of these as shown in Table 17.

Table 15 – Computing $L(u_k)$ and making a decision

L(extrinsic)	7.035	-3.685	1.032	-2.013	1.009	-3.579	-3.682
L(uk)	9.035	-2.685	4.032	-4.013	3.009	-7.579	-8.682
Decision	**1**	**0**	**1**	**0**	**1**	**0**	**0**
Transmitted	**1**	**0**	**1**	**0**	**1**	**0**	**0**

Notice that we make independent decisions in each time tick, with no regard to the trellis, so that makes this algorithm a bit error rate minimization algorithm, where in Viterbi decoding the decision is made at the sequence level, so that makes it a sequence error minimization algorithm.

From here the L_e, interleaved numbers go to the DEC2 as follows.

Table 16 – Transfer of a-posteriori to DEC2 from DEC1

Values In	k=1	k=2	k=3	k=4	k=5	k=6	k=7
$Lc \cdot y_k^{1,s}$	1	2	3	2	-2	-4	-5
$Lc \cdot y_k^{2,p}$	6	-1	2	-2	-5	5	-6
L_k^e	7.035	1.032	1.009	-3.685	-2.013	-3.579	-3.682

The L_e values for this decoder are no longer 0 as they were for DEC1. They are now giving a hint as to what was transmitted. So far the relationship is not clear, but

as iterations continue these values become more stable and hopefully have the same sign as the transmitted bit.

Here are all the tables giving pertinent information for the four iterations. The code makes errors in earlier iterations but then converges to the correct value. Only the first iteration is independent, after that each iteration is modifying the other slowly. The extrinsic values do converge as we can see in the plot of these value for each decoder for k = 1 to k = 4.

Table 17 – Summary of decisions after 4 iterations

DEC1 Decisions	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Iteration 1	1	0	1	0	1	0	0
Iteration 2	1	0	1	0	1	0	0
Iteration 3	1	0	1	0	1	0	0
Iteration 4	**1**	**0**	**1**	**0**	**1**	**0**	**0**

DEC2 Decisions	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Iteration 1	1	1	1	0	0	0	0
Iteration 2	1	1	1	0	0	0	0
Iteration 3	1	1	1	0	0	0	0
Iteration 4	**1**	**1**	**1**	**0**	**0**	**0**	**0**

The L-values that are passed back and forth are shown in table 20.

Table 18 – Extrinsic output and L(uk) after four iterations

DEC1 Extrinsic Output	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Iteration 1	7.04	-3.69	1.03	-2.01	1.01	-3.58	-3.68
Iteration 2	6.25	-7.64	7.67	-12.53	5.37	-12.61	-14.94
Iteration 3	26.63	-22.21	18.36	-20.62	19.19	-22.01	-22.65
Iteration 4	38.36	-34.01	32.19	-35.20	36.60	-37.62	-34.36

DEC1 L(uk)	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Iteration 1	9.04	-2.69	4.03	-4.01	3.01	-7.58	-8.68
Iteration 2	20.79	-15.01	19.28	-20.78	15.01	-19.28	-29.54
Iteration 3	45.27	-39.40	38.37	-41.27	39.40	-38.37	-41.28
Iteration 4	71.58	-68.61	67.81	-67.56	68.61	-67.81	-67.58

DEC2 Extrinsic Output	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Iteration 1	12.54	8.61	7.64	-8.37	-6.25	-2.67	-9.59
Iteration 2	16.64	17.01	18.21	-18.19	-18.65	-12.36	-13.64
Iteration 3	31.21	32.62	30.01	-35.60	-30.36	-26.19	-28.21
Iteration 4	43.01	47.20	45.62	-47.31	-44.19	-43.60	-40.01

DEC2 L(uk)	k=1	k=2	k=3	k=4	k=5	k=6	k=7
Iteration 1	20.58	11.64	11.65	-10.06	-10.26	-10.25	-18.28
Iteration 2	23.89	26.68	26.59	-23.83	-33.18	-28.97	-33.58
Iteration 3	58.84	52.98	52.20	-55.81	-52.98	-52.20	-55.86
Iteration 4	82.37	81.39	85.22	-79.32	-81.39	-85.22	-79.37

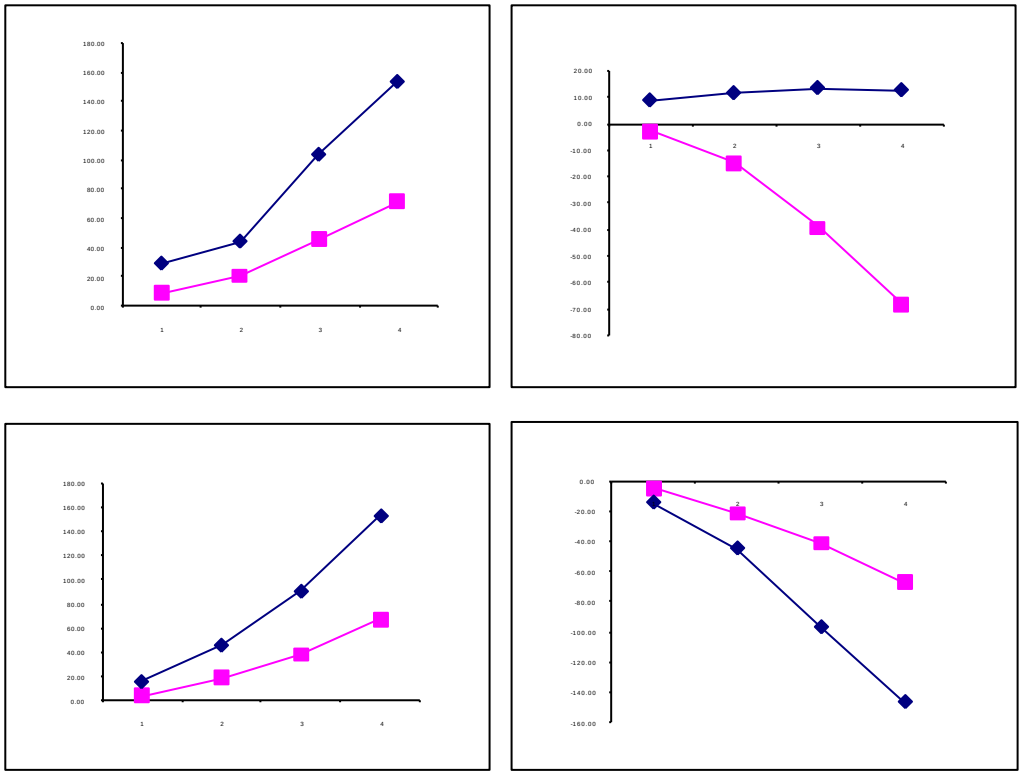

Figure 11 – Extrinsic values over 4 iterations, for DEC1 and DEC2

Charan Langton

Bibliography

Tech online, Course on Turbo Codes: From Theory to Practice

Turbo Code Tutorial, William E. Ryan, New Mexico University, Online paper

Turbo Code in IS-2000 Code Division Multiple Access Communications Under Fading by Jian Qi, Thesis, 1999, http://hometown.aol.com/jxqi/qi/thesis/ (Contains a worked out example for one decoder.)

Material by Binton Cooper on Convolutional codes, http://www.ece.jhu.edu/~cooper/courses/466/09va.pdf

Turbo Decoding, a brief introduction, charts by Brinton Cooper III

http://www.ece.jhu.edu/~cooper/courses/466/12turbo.pdf

and many others – which I will add later.

