Intuitive Guide to Principles of Communications

www.complextoreal.com

Turbo Coding and MAP decoding - Part 1

Baye's Theorem of conditional probabilities

Let's first state the theorem of conditional probability, also called the Baye's theorem.

$$P(A\, and\, B) = P(A)P(B\, given A)$$

Which we can write in more formal terminology as

$$P(A, B) = P(A)P(B|A) \tag{A}$$

where $P(B|A)$ is referred to as the probability of event B given that A has already occurred. If event A always occurs with event B, then we can write the following expression for the absolute probability of event A.

$$P(A) = \sum_B P(A, B) \tag{B}$$

If events A and B are independent from each other then (A) degenerates to

$$P(A, B) = P(A)$$

$$P(A, B) = P(A)P(B) \tag{C}$$

The relationship C is very important and we will use it heavily in the explanation of Turbo decoding in this chapter.

If there are three independent events A, B and C, then the Baye's rule becomes

$$P(A,B|C) = P(A|C)\ \ P(B|A,C) \qquad \text{D}$$

A-priori and a-posteriori probabilities

Here is Bayes' theorem again.

$$\begin{aligned} \underbrace{P(A,B)}_{\substack{\text{Probability of} \\ \text{both } A \text{ and } B}} &= \underbrace{P(AB)}_{\substack{a-posteriori \\ \text{probability of} \\ \text{event } A}}\ \ \underbrace{P(B)}_{\substack{a-priori \\ \text{probability of} \\ \text{event } B}} \\ &= \underbrace{P(B|A)}_{\substack{a-posteriori \\ \text{probability of} \\ \text{event } B}}\ \ \underbrace{P(A)}_{\substack{a-priori \\ \text{probability of} \\ \text{event } A}} \end{aligned} \qquad \text{E}$$

The probability of event A conditioned on event B, is given by the probability of A given B times the probability of event a.

The probability of A, or P(A) is the base probability of even A and is called the **a-priori** probability. The term P(A,B) the conditional probability is called the **a-posteriori probability or APP**. One is independent probability, the other depends on some event occurring. We will be using the acronym APP a lot, so make sure you remember that is the same a-posteriori probability. In other words, the APP of an event is a function of an another event also occurring at the same time. We can write (E) as

$$P(A,B) = \underbrace{P(AB)}_{APP} = \frac{P(B|A)P(A)}{P(B)} \qquad \text{F}$$

This says that we can determine the APP of an event by taking the conditional probability of that event divided by it's a-priori probability.

What these mean is best explained by the following two quotes.

In epistemological terms " A priori" and "a posteriori" refer primarily to how, or on what basis, a proposition might be known. In general terms, a proposition is knowable a priori if it is knowable independently of experience, while a proposition knowable a posteriori is knowable on the basis of experience. The distinction between a priori and a posteriori knowledge thus broadly corresponds to the distinction between empirical and non-empirical knowledge." [2]

"But how do we decide when we have gathered enough data to justify modifying our prediction of the probabilities? That is one of the essential problems of decision theory. How do we make the transition from a priori statistics to a posteriori probability?" [3]

The MAP algorithm helps us make the transition from a-priori knowledge to knowledge based on received data.

Structure of a Turbo Code

According to Shannon, the ultimate code would be one where a message is sent infinite times, each time shuffled randomly. The receiver has an infinite versions of the message albeit corrupted randomly. From these copies, the decoder would be able to decode with near error-free probability the message sent. This is the theory of an ultimate code, the one that can correct all errors for a virtually signal.

Turbo code is a step in that direction. But it turns out that for an acceptable performance we do not really need to send the information infinite number of times, just two or three times provides pretty decent results for our earthly channels.

In Turbo codes, particularly the parallel structure, Recursive systematic convolutional (RSC) codes working in parallel are used to create the “random” versions of the message.

The parallel structure uses two or more RSC codes, each with a different interleaver. The purpose of the interleaver is to offer each encoder an uncorrelated or a “random” version of the information, resulting in parity bits from each RSC that are independent. How “independent” these parity bits are, is essentially a function of the type and length/depth of the interleaver. The design of interleaver in itself is a science. In a typical Viterbi code, the messages are decoded in blocks of only about 200 bits or so, where as in Turbo coding the blocks are on the order of 16K bits long. The reason for this length is to effectively randomize the sequence going to the second encoder. The longer the block length, the better is its correlation with the message from the first encoder, i.e. the correlation is low.

On the receiving side, there are same number of decoders as on the encoder side, each working on the same information and an independent set of parity bit. This type of structure is called Parallel Concatenated Convolutional Code or PCCC.

The convolutional codes used in turbo codes usually have small constraint length. Where a longer constraint length is an advantage in stand-alone convolutional codes, it does not lead to better performance in TC and increases computation complexity and delay. The codes in PCCC must be RSC. The RSC property allows the use of systematic bit as a standard to which the independent parity bits from the different coders are used to assess its reliability. The decoding most often applied is an iterative form of decoding.

When we have two such codes, the signal produced is rate 1/3. If there are three encoders, then the rate is ¼ and so on. Usually two encoders are enough as increasing the number of encoders reduces bandwidth efficiency and does not buy proportionate increase in performance.

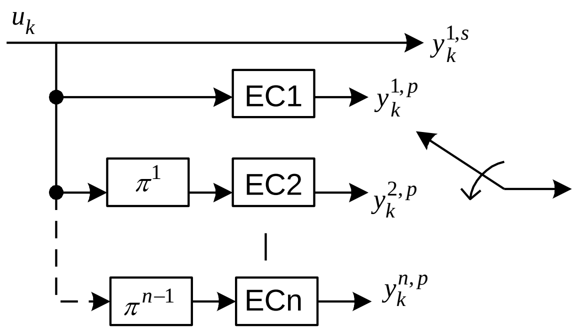


Figure 1 – A rate 1/(n+1) Parallel Concatenated Convolutional Code (PCC) Turbo Code

Turbo codes also come as Serial Concatenated Convolutional Code or SCCC. The SCCC codes appear to have better performance at higher SNRs. Where the PCCC codes require both constituent codes to be RSC, in SCCC, only the inner code must be RSC. PCCC codes also seem to have a flattening of performance around 10^{-6} which is less evident in SCCC. The SCCC constituent code rates can also be different as shown below. The outer code can even be a block code.

In general the PCCC is a special form of SCCC. We can even think of concatenation of RS/Convolutional codes, used in line-of-sight links as a form of SCCC. A Turbo SCCC may look like the figure below with different rate constituent codes.

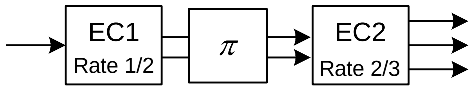


Figure 2 – Serially concatenated constituent coding (SCCC)

Then there are also hybrid versions that use both PCCC and SCCC such as shown in figure below.

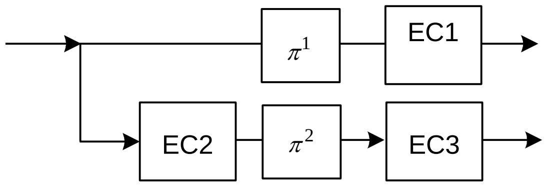
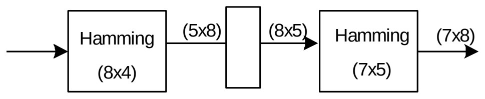


Figure 3 – Hybrid Turbo Codes

There is an another form called **Turbo Product Code** or TPC. This form has a very different structure from the PCCC or SCCC. TPC use block codes instead of convolutional codes. Two different block codes (usually Hamming codes) are concatenated serially without an interleaver in between. Since the two codes are independent and operate in rows and columns, this alone offers enough randomization that no interleaver is required. TPC codes, like PCCC also perform well in low SNR and can be formed by concatenating any type of block codes. Typical coding method is to array the coded data in rows and then the second code uses the columns of the new data for its coding. The following shows a TPC code created from a (7x5) and a (8x4) Hamming code. The 8x4 code first codes the 4 info bits into 8, by adding 4 p1 party bits. These are arrayed in five rows. Then the 7x5 code works on these in columns and creates (in this case, both codes are systematic) new parity bits p2 for each column. The net code rate is 5/14 ~ 0.33. The decoding is done along rows and then columns.



i	i	i	i	p1	p1	p1	p1
i	i	i	i	p1	p1	p1	p1
i	i	i	i	p1	p1	p1	p1
i	i	i	i	p1	p1	p1	p1
i	i	i	i	p1	p1	p1	p1
p2	p2	p2	p2	p2	p2	p2	p2
p2	p2	p2	p2	p2	p2	p2	p2

Figure 4 – Turbo Product codes

What makes all these codes "Turbo" is not their structure but a form of feedback iterative decoding. If the structure of a SCCC does not use the iterative coding then it would be just a plain old concatenated code, not a turbo code.

Maximum a-posteriori Probability (MAP) decoding algorithm

Turbo codes are decoded using a method called the Maximum Likelihood Detection or MLD. Filtered signal is fed to the decoders, and the decoders work on the signal amplitude to output a soft "decision" The a priori probabilities of the input symbols is used, and a *soft* output indicating the *reliability* of the decision (amounting to a suggestion by decoder 1 to decoder 2) is calculated which is then iterated between the two decoders.

The form of MLD decoding used by turbo codes is called the Maximum a-posteriori Probability or MAP. In communications, this algorithm was first identified in BCJR. And that is how it is known for Turbo applications. The MAP algorithm is related to many other algorithms, such as Hidden Markov Model, HMM which is used in voice recognition, genomics and music processing. Other similar algorithms are Baum-Welch algorithm, Expectation maximization, Forward-Backward algorithm, and more. MAP is a complex algorithm, hard to understand and hard to explain.

In addition to MAP algorithm, another algorithm called SOVA , based on Viterbi decoding is also used. SOVA uses Viterbi decoding method but with soft outputs instead of hard. SOVA maximizes the probability of the sequence, whereas MAP maximizes the bit probabilities at each time, even if that makes the sequence not-legal. MAP produces near optimal decoding.

In turbo codes, the MAP algorithm is used iteratively to improve performance. It is like the 20 questions game, where each previous guess helps to improve your knowledge of the hidden information. The number of iteration is often preset as in 20 questions. More iteration are done when the SNR is low, when SNR is high, lesser number of iterations are required since the results converge quickly. Doing 20 iteration maybe a waste if signal quality is good. Instead of making a decision ad-hoc, the algorithm is often pre-set with number of iterations. On the average, seven iterations give adequate results and no more 20 are ever required. These numbers have relationship to the Central Limit Theorem.

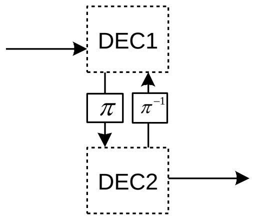


Figure 5 – Iterative decoding in MAP algorithm

Although used together, the terms MAP and iterative decoding are separate concepts. MAP algorithm refers to specific math. The iterative process on the other hand can be applied to any type of coding including block coding which is not trellis based and may not use MAP algorithm.

I am going to concentrate only on PCCC decoding using iterative MAP algorithm In part 2, we will go through a step-by-step example. In this part, we will cover the theory of MAP algorithm.

We are going to describe MAP decoding using a Turbo code in shown Figure 5 with two RSC encoders. Each RSC has two memory registers so the trellis has four states with constraint length equal to 3.

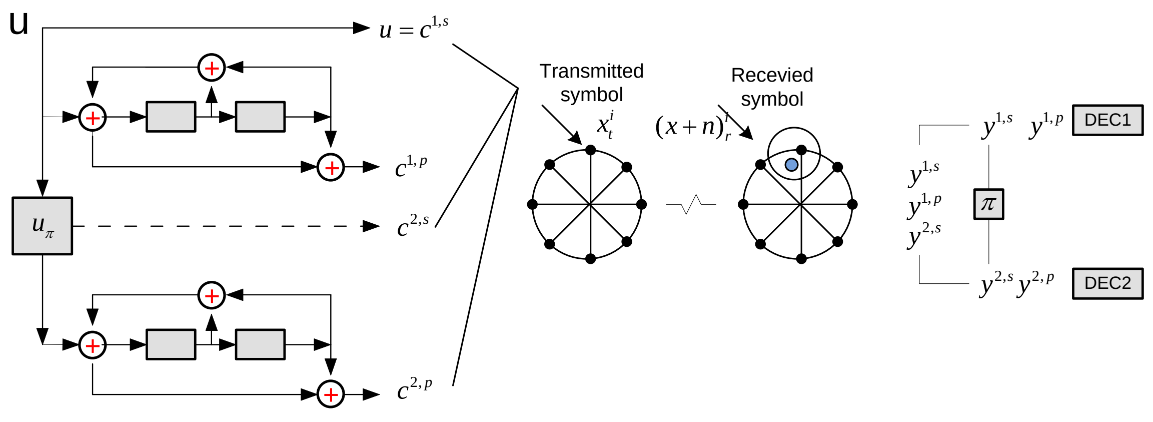


Figure 6 – A rate 1/3 PCCC Turbo code in a 8PSK channel

The rate 1/3 code shown here has two identical RSC convolutional codes. The coding trellis for each is given by the figure 7. The blue lines show transitions in response to a 0 and red lines in response to a 1. The notation 1/11, the first is the input bit, the next are code bits. Of these, the first is what we called the systematic bit, and as you can see it is the same as the input bit. The second bit is the parity bit. Each code uses the same trellis for encoding. The labels along the branches can be read as $u_k / c_k^1\ c_k^2$. Since this a RSC, the first code bit is same as the information bit or $u_k = c_k^1$.

The info bits are called u_k. The coded bits are referred to by the vector c. Then the coded bits are transformed to an analog symbol x and transmitted. On the receive side, a noisy version of x is received. By looking at how far the received symbol is from the decision regions, a metric of confidence is added to each of the three bits in the symbol. Often Gray coding is used, which means that not all bits in the symbol have same level of confidence for decoding purposes. There are special algorithms for mapping the symbols (one received voltage value, to M soft-decisions, with M being the M in M-PSK.) Let's assume that after the mapping and creating of soft-metrics, the vector y is received. One pair of these decoded soft-bits are sent to the first decoder and another set, using a de-interleaved version of the systematic bit and the second parity bit are sent to the second decoder.

Each decoder works only on these bits of information and pass their confidence scores to each other until both agree within a certain threshold. Then the process restarts with next symbol in a sequence or block consisting of N symbols (or bits.)

Definitions

N is the frame size of transmitted symbols. So for a M-PSK, there would be 3N bits transmitted per frame, of these 1/3 will be the information bit. In a typical Turbo code, there may be as many as 20,000 smbols in a frame.

The sequence of information bits is given by u. The first encoder gets $u_k = (u_1, u_2, u_3, \ldots u_N)$ and the second encoder gets a preset reordering of this same sequence. For example, we may pick this mapping function.

$$u = \left[u_1, u_2, u_3, u_4, u_5, u_6, u_7 \ldots u_N\right]$$
$$u_\pi = \left[u_3, u_4, u_1, u_2, u_5, u_6, u_7 \ldots u_N\ \right]$$

The encoder mapping is be given by the vector c. The $c_k = (c_k^{1,s}, c_k^{1,p})$ are two bits produced by the first encoder, and $c_k = (c_k^{2,s}, c_k^{2,p})$ are the two bits produced by the second encoder. The information bit to code bits mapping is done a trellis such as the one described in Table I.

Table I – Mapping of information bits to code bits

State	$s_{k-1}\ \ s_k$	End State	c_k^1	c_k^2	Uk	Branch ID
1	1,1	1	-1	-1	-1	1,1
2	2,3	3	-1	-1	-1	2,3
3	3,5	4	-1	1	-1	3,4
4	4,2	2	-1	1	-1	4,2
1	1,3	3	1	1	1	1,3
2	2,1	1	1	1	1	2,1
3	3,2	2	1	-1	1	3,2
4	4,4	4	1	-1	1	4,4

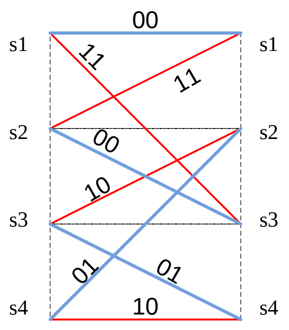


Figure 7 – the trellis diagram of the rate 1/2 code RSC code

The symbol described by vector x (3 bits per vector) is sent for each time i. Let's call this vector x_i. There would N of these symbols transmitted.

$$x_1 = \left[1, 1, 1\right] \quad x_2 = \left[-1, 1, -1\right]$$

$y_k = (y_k^{1,s}, y_k^{1,p}, y_k^{2,p})$ are the soft mapped bits. The goal is to take these and make a guess about the transmitted x vector and hence code bits which inturn decode u, the information bit. Of this three soft bits, each decoder gets just two of these values. The first decoder gets $y_{k1} = (y_k^{1,s}, y_k^{1,p})$ and the second decoder gets $y_{k2} = (y_k^{2,s}, y_k^{2,p})$ which are its respective received data. Each decoder works with just two values. The second decoder however gets a reordered version of the systematic bit, so it is getting only one bit of independent information.

Log-likelihood Ratio (LLR)

Let's take the information bit, a binary variable, u_k, where u is its value at time k. Its Log-likelihood Ratio (LLR) is defined as the natural log of its base probabilities.

$$L(u_k) = \ln \frac{P(u_k = +1)}{P(u_k = -1)} \tag{1.1}$$

If u has two values, +1 and -1 volts representing 0 and 1 bit and since these are equally likely, as they are in most communication system, then this ratio is equal to zero. This metric is used in most error correction coding and is called the **log likelihood ratio** or **LLR**. This is a sensitive metric, quite a bit better than a linear metric. Logs make it easy to deal with very small and very large numbers, as you well know.

Now note what happens to this metric, if the the binary variable is not equally likely, as happens in trellis decoding. From elementary probability theory, we know that sum of all probabilities of an event add to 1. So we can write that the probability of u = +1 as 1 minus the probability of u = -1.

$$P(u_k = -1) = 1 - P(u_k = +1)$$

Now using equation (1.1), rewrite the expression of LLR from (1.2) as.

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

(1.2) is plotted in Figure 8 as a function of the probability of one of the events. Let's say we are given $L(u_k) = 1.0$. Then the probability that $u_k = +1$ is equal to 0.73

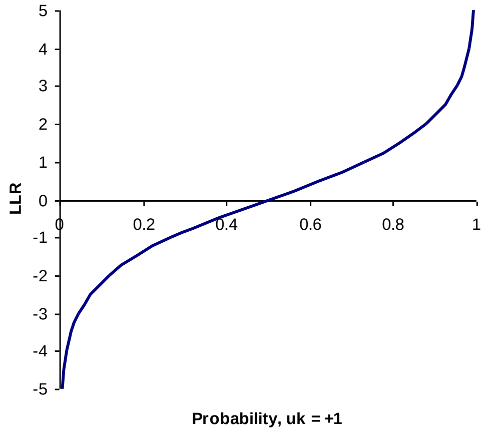


Figure 8 – The range of LLR is from –∞ to +∞ and is a direct indication of the bit.

As we can see, if LLR, a non-dimensional metric, is positive, it is a pretty good indicator of the sign of the bit. This is an even better indicator than the Euclidean distance since its range is so large. LLR a very useful parameter and we will see how it is used in Turbo decoding.

In (1.2) formulate the LLR of a decoded bit u_k, at time k, conditioned on the received signal, a sequence of N bits. The lower index of y means it starts at time 1 and the upper index means the ending point at N.

$$L(u_k) = \ln \frac{P(u_k = +1 \mid y_1^N)}{P(u_k = -1 \mid y_1^N)} \tag{1.2}$$

We can reformulate the numerator and denominator using Baye's rule *C*.

$$L(u_k) = \ln \frac{P(y_1^N, u_k = +1)/P(y_1^N)}{P(y_1^N, u_k = -1)/P(y_1^N)} = \ln \frac{P(y_1^N, u_k = +1)}{P(y_1^N, u_k = -1)} \tag{1.3}$$

This formulation of the LLR includes joint probabilities between the received bits and the information bit, the numerator of which can be written as (1.4). For RSC trellis, each path is uniquely specified by any pair of these: 1. the start state s', 2. the ending state s, 3. The input bit. If we know any two of these pieces of information, we can identify the correct path without error. In this equation, we only know the whole sequence y_1^N, we do not know u_k, nor do we have any other of the piece of information. But what we do know is that saying a u_k = 1 is same as saying that the correct path is one of the four possible paths shown in Fig. 8.

So the joint probability of $(y_1^N, u_k = +1)$ (having received the N bit sequence, the probability of u_k at time k = 1) is the same as replacing the information bit with ending and starting states. These formulations are equivalent.

$$\sum_N P(u_k = +1, y_1^N) = \sum_N P(s', s, y_1^N) \tag{1.4}$$

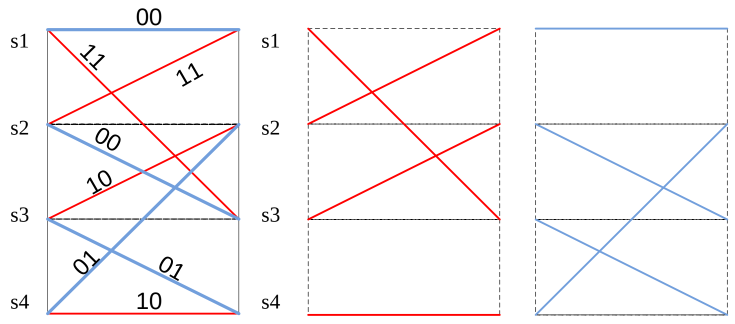


Figure 9 – Trellis transition possible at any time k in response to a +1 and -1.

We have changed the left hand side of the equation by replacing $u_k = +1$ with two states s' and s. The s' is the starting state and s is the ending state for all allowable transitions for which the uk = +1. There are four valid transitions related to decoding a +1. Assume that there is a probability associated with each of these four transitions. This says that the probability of making a decoding decision in favor of a +1 is the sum of the probability of all four of these possible paths.

Now plug **Error! Reference source not found.** into (1.3) to get the log likelihood ratio of u_k as

$$L(u_k) = \ln \frac{P(y_1^N, u_k = +1)}{P(y_1^N, u_k = -1)} = \frac{\sum_{u_k=+1} P(s', s, y_1^N)}{\sum_{u_k=-1} P(s', s, y_1^N)} \tag{1.5}$$

Now we need to make one more conceptual leap. The probability of deciding which road, i.e. the +1 road or the -1 road taken is a function of where the path started and where it will end which are usually given as boundary conditions. If we split the whole received sequence into manageable parts, it may help us identify the starting or ending states. We incorporate these ideas into (1.5) to get,

$$\begin{aligned} y_1^N &= y_1^{k-1}, y_k, y_{k+1}^N \\ &= y_p, y_k, y_f \end{aligned}$$

We take the N bit sequence and separate it into three pieces, from 1 to k-1, then the kth point, and then from k+1 to N. We adopt a slightly easier terminology in (1.6).

$$P(s', s, \underline{\underline{y_1^N}}) = P(s', s, \underline{\underline{y_p, y_k, y_f}}) \tag{1.6}$$

y_p	The past sequence, the part that came before the current data point.
y_k	The current data point
y_f	The sequence points that come after the current point.

Rewrite using (1.6) using Baye's rule of joint probabilities (D).

$$\begin{aligned} P(s', s, \underline{y}) &= P(s', s, y_p, y_k, y_f) \\ &= P(y_f \mid s', s, y_p, y_k) P(s', s, y_p, y_k) \end{aligned} \qquad (1.7)$$

This looks complicated but we are just applying Bayes rule to simplify (1.6) and to breakdown the terms in terms of <u>past</u>, <u>present</u> and <u>future parts of the sequence</u>. So that whenever we are making a decision about a bit +1 or a -1, a cumulative metric will take into account the starting and the ending point of the sequence. The starting and ending points of a convolutional sequence are known and we use this information to winnow out the likelier sequences.

The term y_f is the future sequence and we make an assumption that it is independent of the past and only depends on the present state s. We can remove these dependencies to simplify (1.7).

$$\begin{aligned} P(s', s, \underline{y}) &= P(y_f \mid \cancel{s'}, s, \cancel{y_p}, \cancel{y_k}) P(s', s, y_p, y_k) \\ &= P(y_f \mid s) P(s', s, y_p, y_k) \end{aligned} \qquad (1.8)$$

Now apply Baye's rule to the last term in (1.8), to get

$$P(s', s, y_p, y_k) = P(s, y_k \mid s', y_p) \; P(s', y_k)$$

$$P(s', s, \underline{y}) = P(y_f \mid s) \, P(s, y_k \mid s', y_p) \; P(s', y_p) \qquad (1.9)$$

Now define a few new terms.

$$\begin{aligned}
\alpha_{k-1}(s') &= P(s', y_p) \\
\beta_k(s) &= P(y_f \mid s) \qquad (1.10) \\
\gamma_k(s', s) &= P(s, y_c \mid s', y_p)
\end{aligned}$$

The terms α, β, γ on the right are the metrics we will compute in MAP decoding. Now the numerator term of (1.5) becomes.

$$P(s', s, \underline{y}) = \alpha_{k-1}(s')\beta_k(s)\gamma_k(s', s) \qquad (1.11)$$

We can plug this into (1.5) to get the LLR equation for MAP algorithm.

$$L(u_k) = \frac{\sum_{u_k=+1} \alpha_{k-1}(s')\beta_k(s)\gamma_k(s', s)}{\sum_{u_k=-1} \alpha_{k-1}(s')\beta_k(s)\gamma_k(s', s)} \qquad (1.12)$$

$\alpha_{k-1}(s')$ This first term in (1.11) is called the **Forward metric**.

$\beta_k(s)$ is called the **Backward metric**.

$\gamma_k(s', s)$ is called **Transition metric**.

The MAP algorithm allows us to compute these metrics. Decision is made at each time k, about the transmitted bit u_k. Unlike Viterbi decoding where decision is made in favor if the likeliest sequence by carrying forward the sum of metrics, here the decision is made for the likeliest bit.

The MAP algorithm is also known as **Forward-Backward Algorithm** because we are assessing probabilities in both forward and backward time.

How to calculate the Forward metrics

We will start with the forward metric at time k.

$$\alpha_k(s) = \sum_{\text{All states}} P(s, s', y_p, y_k)$$

Which is also equal to the probabilities at the previous state s' times the transition probability to the current state s.

$$\alpha_k(s) = \sum_{All\ states} P(s, s', y_p, y_k) = \sum_{All\ s'} \gamma(s', s).\,\alpha_{k-1}(s') \quad (1.13)$$

The initial condition is that since we always start at state 1,

$$\alpha_0(s) = \begin{cases} 1 & \text{if s} = 1 \\ 0 & otherwise \end{cases}$$

Example:

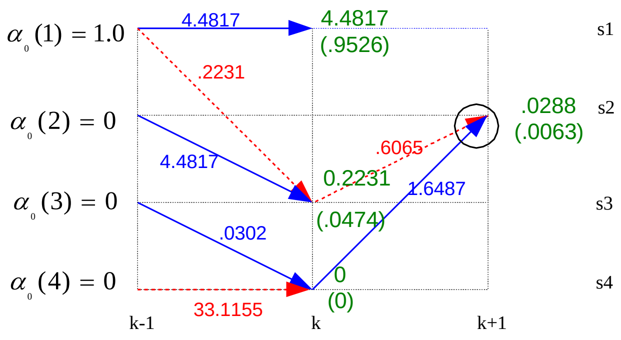


Figure 10 - Computing Forward metrics

The left most value at the top is the starting point, so it has a value of $\alpha_0(1) = 1$, the starting value of α is = 0 at all the remaining three states. The ending forward metric at t = k is

$$\alpha_k(s) = \sum_{All\ s'} \gamma(s', s).\,\alpha_{k-1}(s')$$
$$\alpha_k(1) = 4.4812 \times 1.0 = 4.4812$$
$$\alpha_k(2) = 0.2231 \times 1.0 + 4.4817 \times 0.0 = 0.2231$$

Notice that these numbers are larger than 1, which means, they are not probabilities but are called Metrics. It also means that since a typical Turbo code frame is thousands of trellis sections long that multiplication of these numbers may result in numerical overflow. For that reason, both α and β are normalized.

Backward Metric $\beta_k(s)$

Same as the initial values of $\alpha_0(0) = 1$, we assume that since trellis always ends at state 1, all probabilities at this point are equal to 1 or 0.

$\beta_N(1) = 1$ and zero elsewhere.

This probability is equal to the product of all transition probabilities times the probability at the last state working backwards.

$$\beta_{k-1}(s') = \sum_{All\ s} \beta_k(s)\ \gamma(s', s) \tag{1.14}$$

Compare this with forward metric equation.

$$\alpha_k(s) = \sum_{Alls'} .\alpha_{k-1}(s')\ \ \gamma(s', s)$$

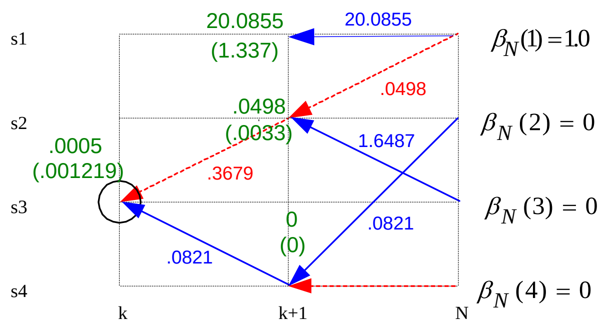


Figure 11 - Computing Backward metrics

$$\beta_{k-1}(s') = \sum_{Alls} \beta_k(s)\ \gamma(s', s)$$
$$\beta_{k+2}(1) = 1.0 \times 20.0885 = 20.0885$$
$$\beta_{k+2}(2) = 1.0 \times .0498 = 0.0498$$

You can see these values at time k+1 for state 1 and 2. These are then normalized.

Numerical issues

In order to prevent data overflow that can happen in the very large trellis of a Turbo codes,, $\alpha_k(s)$ and $\beta_{k-1}(s')$ need to be normalized. For forward metrics, we normalized them by their sum.

$$\tilde{\alpha}_k(s) = \frac{\alpha_k(s)}{\sum_s \alpha_k(s)} \tag{1.15}$$

The reverse metric is similalry normalized by the same quantity as above. It s formulation looks different but it is the same thing as the term that normalized the forward metric. (Change index to k instead of k-1.)

$$\tilde{\beta}_{k-1}(s') = \frac{\tilde{\beta}_{k-1}(s')}{\sum_s \sum_{s'} \tilde{\alpha}_{k-2}(s')\gamma_{k-1}(s',s)}. \tag{1.16}$$

How to calculate transition probabilities

Computing transition metrics turns out to be the hardest part. To restate the definition of the transition metric from (1.10)

$$\gamma_k(s',s) = P(s, y_c \mid s', y_p)$$

The transition itself does not depend on the past, so we can remove the dependency on the past and then apply the Baye's theorem.

$$\begin{aligned}\gamma_k(s',s) &= P(s, y_k \mid s', \cancel{y_p})\\ &= P(s, y_k \mid s') \\ &= P(y_k \mid s', s) \, . \, P(s \mid s')\end{aligned} \tag{1.17}$$

$$\gamma_k(s',s) = \underline{P(y_k \mid s', s)} \, . \, \underline{P(u_k)} \tag{1.18}$$

There are two terms in this final definition of the transition metric. Let's take the last one first. Here P(u_k) is a-priori probability of the input bit. Let's take its log likelihood.

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

Or by taking this expression to power of e, we get

$$e^{L(u_k)} = \frac{P(u_k = 1)}{1 - P(u_k = 1)}$$

From here, some clever algebra gives us

$$\begin{aligned} P(u_k = +1) &= e^{L(u_k)}(1 - P(u_k = +1)) \\ &= \frac{e^{L(u_k)}}{1 + e^{L(u_k)}} \\ &= \frac{1}{1 + e^{-L(u_k)}} \\ &= \frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}}\ e^{L(u_k)/2} \end{aligned}$$

Now we get the general expression for both, +1 and -1.

$$P(u_k = \pm 1) = \underline{\frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}}}\ e^{\pm L(u_k)/2} \qquad (1.19)$$

The term underlined is a common factor and designated by

$$A_k = \frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}}$$

The a-priori probability is now given by

$$P(u_k = \pm 1) = A_k\ e^{\pm L(u_k)/2} \qquad (1.20)$$

Now for the second term in transition metric expression of (1.18), $P(y_k \mid s^{'}, s)$ can be written as

$$P(y_k \mid s^{'}, s) = P(y_k \mid c_k) = \prod_{l=1}^{n} P(y_{kl} \mid c_{kl})$$

For rate ½, we can write this simply as,

$$P(y_k \mid x_k) = y_k^{1,s} c_k^1 + y_k^{1,p} c_k^2$$

This is just a correlation of the input signal values with the trellis values, c. The received signal values y_k are assumed to be corrupted by an AWGN process. The probability P($y_{kl} \mid c_{kl}$) is given in a Gaussian channel by

$$P(y_{kl} \mid c_{kl}) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(y_{kl}-c_{kl})^2\right) \tag{1.21}$$

Apply (1.21) to (1.18), to get

$$P(y_k / u_k) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right) EXP\left(-\frac{\left(y_k^{1,s}-c_k^{1,s}\right)^2}{2\sigma_n^2} - \sum_{i=2}^{q} \frac{\left(y_k^{i,p}-c_k^{i,p}\right)^2}{2\sigma_n^2} \right)$$

Expanding this we get,

$$= \left(\frac{1}{\sqrt{2\pi}\sigma}\right) EXP\left(-\frac{(y_k^{1,s})^2 + \overset{=1}{(c_k^{1,s})^2}}{2\sigma_n^2} - \sum_{i=2}^{q} \frac{\left(y_k^{i,p}\right)^2 + \overset{=1}{\left(c_k^{p,i}\right)^2}}{2\sigma_n^2} \right) EXP\left(\frac{y_k^{1,s}c_k^{1,s}}{\sigma_n^2} + \sum_{i=2}^{q} \frac{y_k^{i,p}c_k^{i,p}}{\sigma_n^2} \right)$$

The square of the two c (always equal to 1 or -1) values is equal to 1, since square of +1 and -1 is the same. This gives the following equation

$$= B_k \; EXP\left(\frac{y_k^{1,s}c_k^{1,s}}{\sigma_n^2} + \sum_{i=2}^{q} \frac{y_k^{i,p}c_k^{i,p}}{\sigma_n^2} \right) \tag{1.22}$$

Where

$$B_k = EXP\left(-\frac{(y_k^{1,s})^2 + 1}{2\sigma_n^2} - \sum_{i=2}^{q} \frac{\left(y_k^{i,p}\right)^2 + 1}{2\sigma_n^2} \right)$$

.

Now we can write the full equation for transition metric by combining (1.22) and (1.19).

$$\gamma_k(s',s) = \underline{P(y_k \mid s', s)}\,.\;\underline{P(u_k)}\,.$$

$$= B_k \exp\left(\frac{y_k^{1,s} c_k^{1,s}}{\sigma_n^2} + \sum_{i=2}^{q} \frac{y_k^{i,p} c_k^{i,p}}{\sigma_n^2} \right) A_k\, e^{\pm L(c_k)/2} \qquad (1.23)$$

A$_k$ and B$_k$ are constants and are not important because when we put this expression in the LLR form, they will cancel out. The index q = 2 for rate ½ code we are using for example.

Now we define,

$$\sigma_n^2 = \frac{N_0}{2} = \frac{E_c}{2 \cdot coderate \cdot E_b / N_0} = \frac{p}{2 \cdot E_b / N_0} \qquad (1.24)$$

Where p is inverse of code rate, i.e. equal to 2 for code rate = 1/2

$$\frac{E_b}{N_0} = \frac{E_c}{rate \cdot N_0}\,. \qquad (1.25)$$

where $\frac{E_b}{N_0}$ is ratio of the un-coded bit energy to noise PSD, E$_c$ is coded bit energy, and E$_c$= code rate × E$_b$.

Substitute for $\sigma_n^2 = \frac{p}{2 \cdot E_b / N_0}$ into (1.23) we have

$$= A_k \cdot B_k \cdot \exp\left[\frac{4 \cdot E_b / N_0}{p} \cdot \left(\frac{y_k^{1,s} \cdot c_k^1}{2} + \sum_{i=2}^{q} \frac{y_k^{i,p} \cdot c_k^i}{2} \right) + \frac{1}{2} \cdot L(c_k^1) \cdot c_k^1 \right]$$

$$= A_k \cdot B_k \cdot \exp\left[\left(\frac{4 \cdot E_b / N_0}{p} \cdot \frac{1}{2} \cdot \left(y_k^{1,s} \cdot c_k^1 + \sum_{i=2}^{q} y_k^{i,p} \cdot c_k^i\right) + \frac{1}{2} \cdot L(c_k^1) \cdot c_k^1\right)\right]$$

$$= A_k \cdot B_k \cdot \exp\left[\frac{1}{2} \cdot L(c_k^1) \cdot c_k^1 + \frac{4 \cdot E_b / N_0}{p} \cdot \frac{1}{2} \cdot y_k^{1,s} \cdot c_k^1\right] \exp\left[\sum_{i=2}^{q}\left(\frac{4 \cdot E_b / N_0}{p} \cdot \frac{1}{2} \cdot y_k^{i,p} \cdot x_k^i\right)\right]$$

With $L_c = \frac{4 \cdot E_b / N_0}{p} = 4.E_s / N_0$

$$= A_k \cdot B_k \cdot \exp\left[\frac{1}{2} \cdot L(c_k^1) \cdot c_k^1 + Lc \cdot \frac{1}{2} \cdot y_k^{1,s} \cdot c_k^1\right] \underline{\exp\left[\sum_{i=2}^{q}\left(Lc \cdot \frac{1}{2} \cdot y_k^{i,p} \cdot c_k^i\right)\right]} \qquad (1.26)$$

where we define a new partial transition metric, the underlined part of (1.26).

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

In summary the LLR of the information bit u_k at time k is,

$$L(u_k) = \log\left[\frac{\sum_{u^+} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)\tilde{\beta}_k(s)}{\sum_{u^-} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)\tilde{\beta}_k(s)}\right] \qquad (1.27)$$

$$\tilde{\alpha}_k(s) = \frac{\sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}{\sum_{s}\sum_{s'} \tilde{\alpha}_{k-1}(s')\gamma_k(s',s)}, \quad \tilde{\alpha}_0(s) = \begin{cases} 1 & \text{if s} = 1 \\ 0 & \textit{otherwise} \end{cases} \qquad (1.28)$$

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_{s} \tilde{\beta}_k(s)\gamma_k(s',s)}{\sum_{s}\sum_{s'} \tilde{\alpha}_{k-2}(s')\gamma_{k-1}(s',s)}, \quad \tilde{\beta}_N(s) = \begin{cases} 1 & \text{if s} = 1 \\ 0 & otherwise \end{cases} \tag{1.29}$$

$$\gamma(s',s) = \exp\left[\frac{1}{2}\cdot L^e(c_k^1)\cdot c_k^1 + Lc\cdot\frac{1}{2}\cdot y_k^{1,s}\cdot c_k^1\right]\exp\left[\sum_{i=2}^{q}\left(Lc\cdot\frac{1}{2}\cdot y_k^{i,p}\cdot c_k^i\right)\right] \tag{1.30}$$

Substitute (1.30) into (1.27), to get

$$= \log\frac{\sum_{u^+} \tilde{\alpha}_{k-1}(s')\cdot\tilde{\beta}_k(s)\cdot\exp\left[\frac{1}{2}L^e(c_k^1)\cdot c_k^1 + \frac{1}{2}Lc\cdot y_k^{1,s}\cdot c_k^1\right]\cdot\exp\left[\sum_{i=2}^{q}\left(Lc\cdot\frac{1}{2}\cdot y_k^{i,p}\cdot c_k^i\right)\right]}{\sum_{u^-} \tilde{\alpha}_{k-1}(s')\cdot\tilde{\beta}_k(s)\cdot\exp\left[\frac{1}{2}L^e(c_k^1)\cdot c_k^1 + \frac{1}{2}Lc\cdot y_k^{1,s}\cdot c_k^1\right]\cdot\exp\left[\sum_{i=2}^{q}\left(Lc\cdot\frac{1}{2}\cdot y_k^{i,p}\cdot c_k^i\right)\right]}$$

(1.31)

If q = 2, the equation become

$$= \log\frac{\sum_{u^+} \tilde{\alpha}_{k-1}(s')\cdot\tilde{\beta}_k(s)\cdot\exp\left[\frac{1}{2}L^e(c_k^1)\cdot c_k^1 + \frac{1}{2}Lc\cdot y_k^{1,s}\cdot c_k^1\right]\cdot\exp\left[Lc\cdot\frac{1}{2}\cdot y_k^{2,p}\cdot c_k^2\right]}{\sum_{u^-} \tilde{\alpha}_{k-1}(s')\cdot\tilde{\beta}_k(s)\cdot\exp\left[\frac{1}{2}L^e(c_k^1)\cdot c_k^1 + \frac{1}{2}Lc\cdot y_k^{1,s}\cdot c_k^1\right]\cdot\exp\left[Lc\cdot\frac{1}{2}\cdot y_k^{2,p}\cdot c_k^2\right]}$$

$L(u_k^1)$ *and* $Lc\cdot y_k^{1,s}$ can be pulled out because both numerator and denominator are constant. The factor ½ and c_k^1 cancel. We get,

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

Where

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

This equation can be broken into three distinct numbers. They are

$$= L\_apriori + L\_channel + L\_extrinsic \qquad (1.32)$$

$L(c_k^1)$ is the a-priori information about u$_k$, $Lc\cdot y_k^{1,s}$ is the channel value calculated from the knowledge of the SNR and the received signal. The third term is called the a-posteriori term, also called the extrinsic L value.

$$L\_extrinsic = \log\frac{\sum_{u^+}\tilde{\alpha}_{k-1}(s')\cdot\tilde{\beta}_k(s)\cdot\gamma_k^e(s',s)}{\sum_{u^-}\tilde{\alpha}_{k-1}(s')\cdot\tilde{\beta}_k(s)\cdot\gamma_k^e(s',s)} \qquad (1.33)$$

During each iteration the decoder produces the extrinsic value using the first two numbers as input. The extrinsic value it produces becomes the input to the next decoder.

$$L_1(c_k^1) = Lc\cdot y_k^{1,s} + L_{21}^e(c_k^1) + L_{12}^e(c_k^1),$$

And eventually a decision is made about the bit in question by looking at the sign of the L value.

$$\tilde{u}_k = \text{sign}\{\text{L}_1(c_k^1)\},$$

The process can continue until the extrinsic values stop changing with a preset threshold. Or the algorithm can allow just a fixed number of iterations.

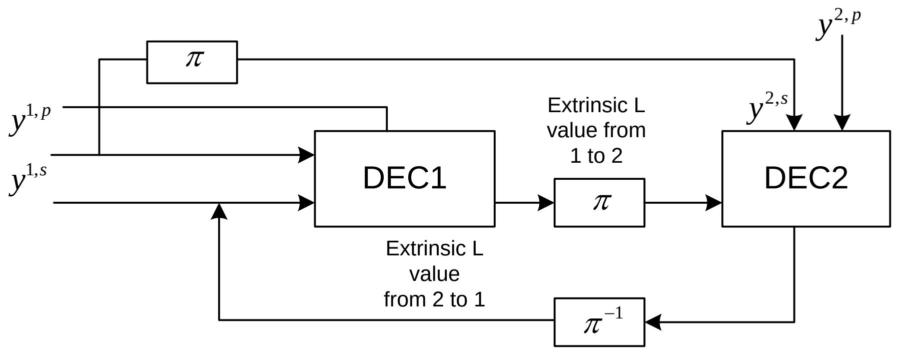


Figure 12 – Iterative nature of Turbo decoding.

So that's about it. I hope it was helpful. The next tutorial goes through the MAP algorithm in a step-by-step example.

Please contact me if you find errors.

Charan Langton
www.complextoreal.com

References

[1] Turbo code in IS-2000 Division Multiple-Access Communications Under Fading By Jian Qi. Masters Thesis, Wichita State University, Fall 1999. Comment: Available on the Internet. I used Ms. Qi's terminology and heavily relied on this work. Its an excellent reference.

[2] http://www.iep.utm.edu/a/apriori.htm, • "A Priori and A Posteriori" - an article by Jason Baehr in the *Internet Encyclopedia of Philosophy*.

[3] http://www.lockergnome.com/it/2007/06/13/a-priori-statistics-to-a-posteriori-probability/

[4] Not complete…