

From BCJR to turbo decoding: MAP algorithms made easier

© Silvio A. Abrantes*

April 2004

In 1974 Bahl, Cocke, Jelinek and Raviv published the decoding algorithm based on a posteriori probabilities later on known as the *BCJR*, *Maximum a Posteriori (MAP)* or *forward-backward* algorithm. The procedure can be applied to block or convolutional codes but, as it is more complex than the Viterbi algorithm, during about 20 years it was not used in practical implementations. The situation was dramatically changed with the advent of turbo codes in 1993. Their inventors, Berrou, Glavieux and Thithimajshima, used a modified version of the BCJR algorithm, which has reborn vigorously that way.

There are several simplified versions of the MAP algorithm, namely the log-MAP and the max-log-MAP algorithms. The purpose of this tutorial text is to clearly show, without intermediate calculations, how all these algorithms work and are applied to turbo decoding. A complete worked out example is presented to illustrate the procedures. Theoretical details can be found in the Appendix.

1 The purpose of the BCJR algorithm

Let us consider a block or convolutional encoder described by a trellis, and a sequence $\mathbf{x} = x_1 x_2 \dots x_N$ of N n -bit codewords or symbols at its output, where x_k is the symbol generated by the encoder at time k . The corresponding information or message input bit, u_k , can take on the values -1 or $+1$ with an *a priori* probability $P(u_k)$, from which we can define the so-called *log-likelihood ratio (LLR)*

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

This log-likelihood ratio is zero with equally likely input bits.

Suppose the coded sequence \mathbf{x} is transmitted over a memoryless additive white gaussian noise (AWGN) channel and received as the sequence of nN real numbers $\mathbf{y} = y_1 y_2 \dots y_N$, as shown in Fig. 1. This sequence is delivered to the decoder and used by the BCJR [1], or any other, algorithm in order to estimate the original bit sequence u_k . for which the algorithm computes the a posteriori log-likelihood ratio $L(u_k | \mathbf{y})$, a real number defined by the ratio

$$L(u_k | \mathbf{y}) = \ln \frac{P(u_k = +1 | \mathbf{y})}{P(u_k = -1 | \mathbf{y})}. \quad (2)$$

The numerator and denominator of Eq. (2) contain a posteriori conditional probabilities, that is, probabilities computed after we know \mathbf{y} . The positive or negative sign of $L(u_k | \mathbf{y})$ is an indication of which bit, $+1$ or -1 , was coded at time k . Its magnitude is a measure of the confidence we have in the preceding

* Faculdade de Engenharia da Universidade do Porto (FEUP), Porto, Portugal.

decision: the more the $L(u_k|\mathbf{y})$ magnitude is far away from the zero threshold decision the more we trust in the bit estimation we have made. This soft information provided by $L(u_k|\mathbf{y})$ can then be transferred to another decoding block, if there is one, or simply converted into a bit value through a hard decision: as it was said before, if $L(u_k|\mathbf{y}) < 0$ the decoder will estimate bit $u_k = -1$ was sent. Likewise, it will estimate $u_k = +1$ if $L(u_k|\mathbf{y}) > 0$.

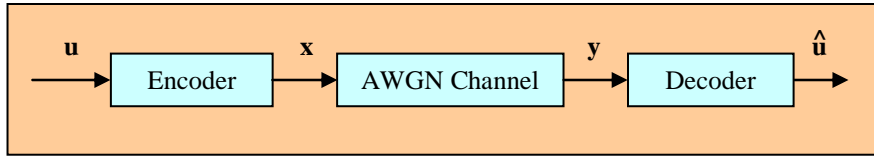


Fig. 1 A simplified block diagram of the system under consideration

It is convenient to work with trellises. Let us admit we have a rate $1/2$, $n = 2$, convolutional code, with $M = 4$ states $S = \{0, 1, 2, 3\}$ and a trellis section like the one presented in Fig. 2. Here a dashed line means that branch was generated by a -1 message bit and a solid line means the opposite. Each branch is labeled with the associated two bit codeword x_k , where, for simplicity, 0 and 1 correspond to -1 and $+1$, respectively.

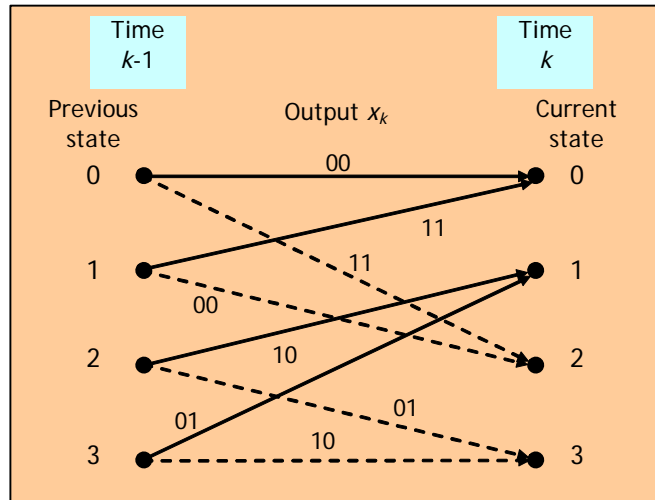


Fig. 2 The convolutional code trellis used in the example

Let us suppose we are at time k . The corresponding state is $S_k = s$, the previous state is $S_{k-1} = s'$ and the decoder received symbol is y_k . Before this time $k-1$ symbols have already been received and after it $N-k$ symbols will be received. That is, the complete sequence \mathbf{y} can be divided into three subsequences, one representing the past, another the present and another one the future:

$$\mathbf{y} = \underbrace{y_1 y_2 \dots y_{k-1}}_{\mathbf{y}_{<k}} y_k \underbrace{y_{k+1} \dots y_N}_{\mathbf{y}_{>k}} = \mathbf{y}_{<k} \mathbf{y}_k \mathbf{y}_{>k} \quad (3)$$

In the Appendix it is shown that the a posteriori LLR $L(u_k|\mathbf{y})$ is given by the expression

$$L(u_k | \mathbf{y}) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)} \quad (4)$$

$P(s', s, \mathbf{y})$ represents the joint probability of receiving the N -bit sequence \mathbf{y} and being in state s' at time $k-1$ and in state s at the current time k . In the numerator R_1 means the summation is computed over all the state transitions from s' to s that are due to message bits $u_k = +1$ (i. e. , dashed branches). Likewise, in the denominator R_0 is the set of all branches originated by message bits $u_k = -1$. The variables α , γ and β represent probabilities to be defined later.

2 The joint probability $P(s', s, \mathbf{y})$

This probability can be computed as the product of three other probabilities,

$$P(s', s, \mathbf{y}) = \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s), \quad (5)$$

They are defined as:

$$\alpha_{k-1}(s') = P(s', \mathbf{y}_{<k}) \quad (6)$$

$$\gamma_k(s', s) = P(\mathbf{y}_k, s | s') \quad (7)$$

$$\beta_k(s) = P(\mathbf{y}_{>k} | s) \quad (8)$$

At time k the probabilities α , γ and β are associated with the past, the present and the future of sequence \mathbf{y} , respectively. Let us see how to compute them by starting with γ .

2.1 Calculation of γ

The probability $\gamma_k(s', s) = P(\mathbf{y}_k, s | s')$ is the conditional probability that the received symbol is \mathbf{y}_k at time k and the current state is $S_k = s$, knowing that the state from which the connecting branch came was $S_{k-1} = s'$. It turns out that γ is given by

$$\gamma_k(s', s) = P(\mathbf{y}_k | x_k) P(u_k)$$

In the special case of an AWGN channel the previous expression becomes

$$\gamma_k(s', s) = C_k e^{u_k L(u_k)/2} \exp\left(\frac{L_c}{2} \sum_{l=1}^n x_{kl} y_{kl}\right), \quad (9)$$

where C_k is a quantity that will cancel out when we compute the conditional LLR $L(u_k | \mathbf{y})$, as it appears both in the numerator and the denominator of Eq. (4), and L_c , the channel reliability measure, or value, is equal to

$$L_c = 4a \frac{E_c}{N_0} = 4a R_c \frac{E_b}{N_0}$$

$N_0/2$ is the noise bilateral power spectral density, a is the *fading amplitude* (for nonfading channels it is $a = 1$), E_c and E_b are the transmitted energy per coded bit and message bit, respectively, and R_c is the code rate.

2.2 Recursive calculation of α and β

The probabilities α and β can (and should) be computed recursively. The respective recursive formulas are¹:

$$\alpha_k(s) = \sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s) \quad \text{Initial conditions: } \alpha_0(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \quad (10)$$

$$\beta_{k-1}(s') = \sum_s \beta_k(s) \gamma_k(s', s) \quad \text{Initial conditions: } \beta_N(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases} \quad (11)$$

Please note that:

- In both cases we need the same quantity, $\gamma_k(s', s)$. It will have to be computed first.
- In the $\alpha_k(s)$ case the summation is over all previous states $S_{k-1} = s'$ linked to state s by converging branches, while in the $\beta_{k-1}(s')$ case the summation is over all next states $S_k = s$ reached with the branches stemming from s' . These summations contain only two elements with binary codes.
- The probability α is being computed as the sequence \mathbf{y} is received. That is, when computing α we go forward from the beginning to the end of the trellis.
- The probability β can only be computed after we have received the whole sequence \mathbf{y} . That is, when computing β we come backward from the end to the beginning of the trellis.
- We will see that α and β are associated with the encoder states and that γ is associated with the branches or transitions between states.
- The initial values $\alpha_0(s)$ and $\beta_N(s)$ mean the trellis is terminated, that is, begins and ends in the all-zero state. Therefore it will be necessary to add some tail bits to the message in order that the trellis path is forced to return back to the initial state.

It is self-evident now why the BCJR algorithm is also known as the “*forward-backward algorithm*”.

3 Combatting numerical instability

Numerical problems associated with the BCJR algorithm are well known. In fact, the iterative nature of some computations may lead to undesired overflow or underflow situations. To circumvent them normalization countermeasures should be taken. Therefore, instead of using α and β directly from recursive equations (10) and (11) into Eq. (5) those probabilities should previously be normalized by the sum of all α and β at each time, respectively. The same applies to the joint probability $P(s', s, \mathbf{y})$, as follows. Define the auxiliary unnormalized variables α' and β' at each time step k :

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

¹ These recursive equations are prone to numerical instability. We will see soon how to overcome it.

$$\beta'_{k-1}(s') = \sum_s \beta_k(s) \gamma_k(s', s)$$

After all M values of α' and β' have been computed sum them all: $\sum_s \alpha'_k(s)$ and $\sum_{s'} \beta'_{k-1}(s')$.

Then normalize α and β dividing them by these summations:

$$\alpha_k(s) = \frac{\alpha'_k(s)}{\sum_s \alpha'_k(s)} \quad (12)$$

$$\beta_{k-1}(s') = \frac{\beta'_{k-1}(s')}{\sum_{s'} \beta'_{k-1}(s')} \quad (13)$$

Likewise, after all $2M$ products $\alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)$ over all the trellis branches have been computed at time k their sum

$$\begin{aligned} \Sigma_{P_k} &= \sum_{R_o, R_1} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s) = \\ &= \sum_{R_o} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s) + \sum_{R_1} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s) \end{aligned}$$

will normalize $P(s', s, \mathbf{y})$:

$$P_{norm}(s', s, \mathbf{y}) = \frac{P(s', s, \mathbf{y})}{\Sigma_{P_k}}$$

This way we guarantee all α , β and $P_{norm}(s', s, \mathbf{y})$ always sum to 1 at each time step k . None of these normalization summations affect the final log-likelihood ratio $L(u_k | \mathbf{y})$ as all them appear both in its numerator and denominator:

$$L(u_k | \mathbf{y}) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} P_{norm}(s', s, \mathbf{y})}{\sum_{R_0} P_{norm}(s', s, \mathbf{y})} \quad (14)$$

4 Trellis-aided calculation of α and β

Fig. 3 shows the trellis of Fig. 2 but now with new labels. We recall that in our convention a dashed line results from a +1 input bit while a solid line results from a -1 input bit.

Let us do the following as computations are made:

- Label each trellis branch with the value of $\gamma_k(s', s)$ computed according to Eq. (9).
- In each state node write the value of $\alpha_k(s)$ computed according to Eqs. (10) or (12) from the initial conditions $\alpha_0(s)$.

- In each state node and below $\alpha_k(s)$ write the value of $\beta_k(s)$ computed according to Eqs. (11) or (13) from the initial conditions $\beta_N(s)$.

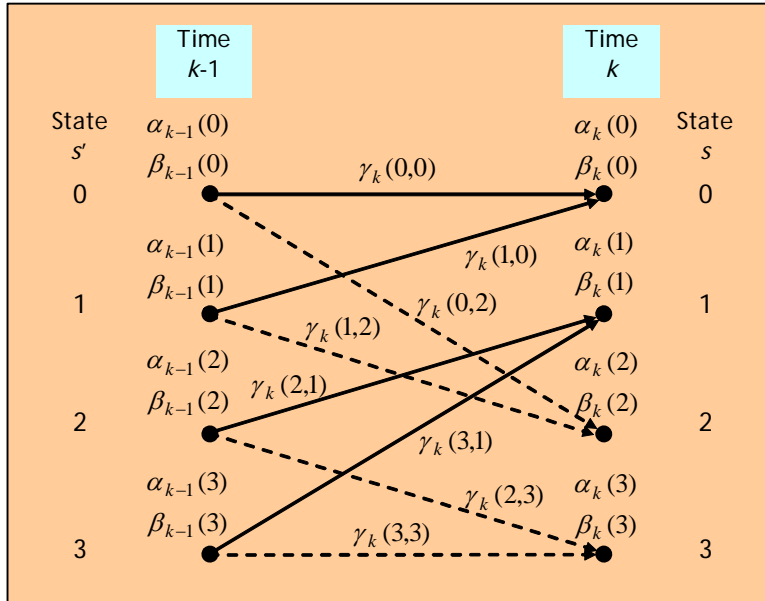


Fig. 3 α , β and γ as trellis labels

4.1 Calculation of α

Let us suppose then that we know $\alpha_{k-1}(s')$. The probability $\alpha'_k(s)$ (without normalization) is obtained by summing the products of $\alpha_{k-1}(s')$ and $\gamma_k(s', s)$ associated with the branches that converge into s . For example, we see in Fig. 3 that at time k two branches arrive at state $S_k = 2$, one coming from state 0 and the other one coming from state 1, as Fig. 4a shows. After all M $\alpha'_k(s)$ have been computed as explained they should be divided by their sum.

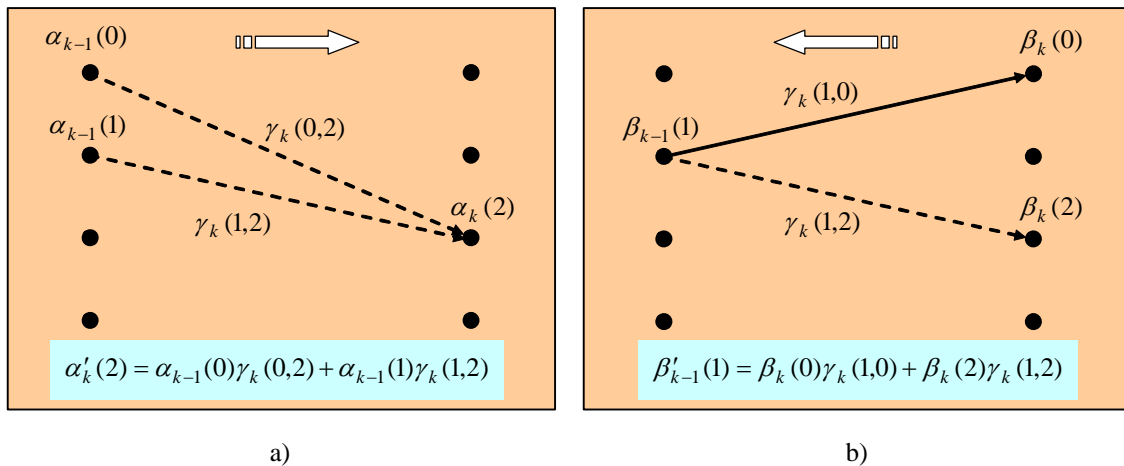


Fig. 4 Trellis-aided recursive computation of α and β . Expressions are not normalized.

The procedure is repeated until we reach the end of the received sequence and have computed $\alpha_N(0)$ (remember our trellis terminates at the all-zero state).

4.2 Calculation of β

The quantity β can be calculated recursively only after the complete sequence \mathbf{y} has been received. Knowing $\beta_k(s)$ the value of $\beta_{k-1}(s')$ is computed in a similar fashion as $\alpha_k(s)$ was: we look for the branches that leave state $S_{k-1} = s'$, sum the corresponding products $\gamma_k(s) \beta_k(s)$ and divide by the sum $\sum_{s'} \beta'_{k-1}(s')$. For example, in Fig. 3 we see that two branches leave the state $S_{k-1} = 1$, one directed to state $S_k = 0$ and the other directed to state $S_k = 2$, as Fig. 4b shows. The procedure is repeated until the calculation of $\beta_0(0)$.

4.3 Calculation of $P(s', s, \mathbf{y})$ and $L(u_k | \mathbf{y})$

With all the values of α , β and γ available we are ready to compute the joint probability $P_{norm}(s', s, \mathbf{y}) = \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s) / \sum_{P_k}$. For instance, what is the value of unnormalized $P(1, 2, \mathbf{y})$? As Fig. 5 shows, it is $\alpha_{k-1}(1) \gamma_k(1, 2) \beta_k(2)$.

We are left just with the a posteriori LLR $L(u_k | \mathbf{y})$. Well, let us observe the trellis of Fig. 3 again: we notice that a message bit +1 causes the following state transitions: $0 \rightarrow 2$, $1 \rightarrow 2$, $2 \rightarrow 3$ e $3 \rightarrow 3$. These are the R_1 transitions of Eq. (4). The remaining four state transitions, represented by a solid line, are caused, of course, by an input bit -1. These are the R_0 transitions. Therefore, the first four transitions are associated with the numerator of Eq. (4) and the remaining ones with the denominator:

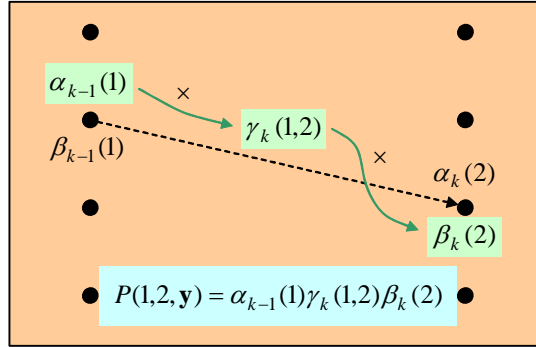


Fig. 5 The unnormalized probability $P(s', s, \mathbf{y})$ as the three-factor product $\alpha\gamma\beta$

$$\begin{aligned}
 L(u_k | \mathbf{y}) &= \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \\
 &= \ln \frac{P(0, 2, \mathbf{y}) + P(1, 2, \mathbf{y}) + P(2, 3, \mathbf{y}) + P(3, 3, \mathbf{y})}{P(0, 0, \mathbf{y}) + P(1, 0, \mathbf{y}) + P(2, 1, \mathbf{y}) + P(3, 1, \mathbf{y})}
 \end{aligned}$$

A summary of the unnormalized expressions needed to compute the conditional LLR is presented in Fig. 6.

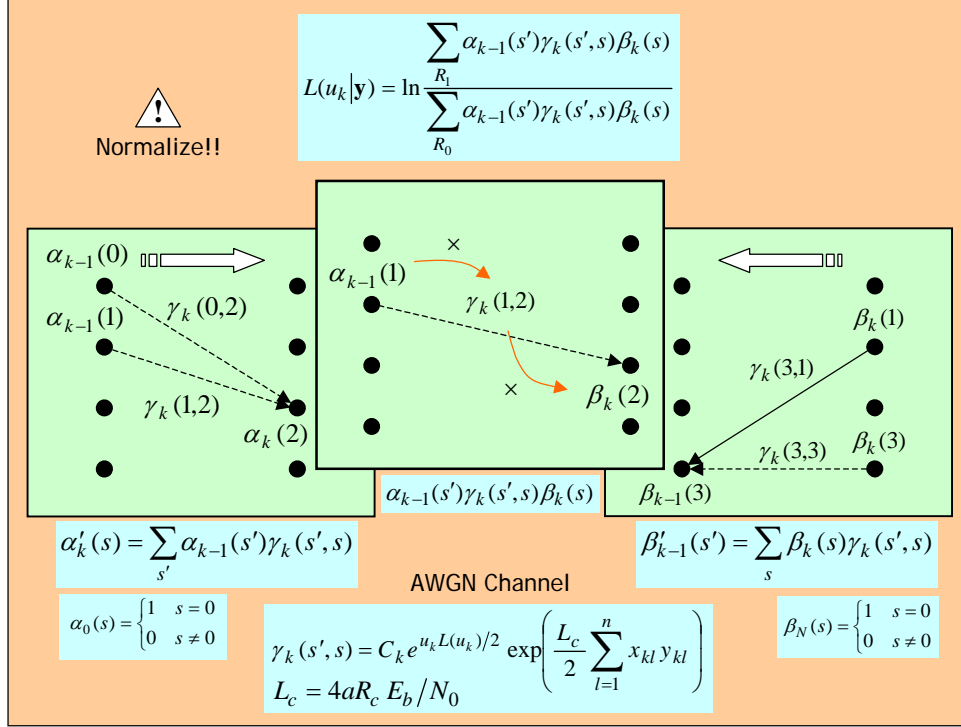


Fig. 6 Summary of expressions used in the BCJR algorithm

5 Simplified versions of the MAP algorithm

The BCJR, or MAP, algorithm suffers from an important disadvantage: it needs to perform many multiplications. In order to reduce this computational complexity several simplifying versions have been proposed, namely the Soft-Output Viterbi Algorithm (SOVA) in 1989 [2], the max-log-MAP algorithm in 1990-1994 [3] [4] and the log-MAP algorithm in 1995 [5]. In this text we will only address the log-MAP and the max-log-MAP algorithms. Both substitute additions for multiplications. Three new variables, A , B e Γ , are defined:

$$\begin{aligned} \Gamma_k(s', s) &= \ln \gamma_k(s', s) = \\ &= \ln C_k + \frac{u_k L(u_k)}{2} + \frac{L_c}{2} \sum_{l=1}^n x_{kl} y_{kl} \\ A_k(s) &= \ln \alpha_k(s) = \max_{s'} [A_{k-1}(s') + \Gamma_k(s', s)] & A_0(s) &= \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases} \\ B_{k-1}(s') &= \ln \beta_{k-1}(s') = \max_s [B_k(s) + \Gamma_k(s', s)] & B_N(s) &= \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases} \end{aligned}$$

where

$$\max^*(a, b) = \begin{cases} \max(a, b) + \ln(1 + e^{-|a-b|}) & \text{log - MAP algorithm} \\ \max(a, b) & \text{max - log - MAP algorithm} \end{cases} \quad (15)$$

The values of the function $\ln(1+e^{-|a-b|})$ are usually stored in a look-up table with just eight values of $|a-b|$ between 0 and 5 (it is enough, actually!).

The element $\ln C_k$ in the expression of $\Gamma_k(s', s)$ will not be used in the $L(u_k|\mathbf{y})$ computation. The LLR is given by the final expression

$$L(u_k|\mathbf{y}) = \max_{R_1}^* [A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)] - \max_{R_0}^* [A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)] \quad (16)$$

A summary of the expressions needed to compute Eq. (16) is presented in Fig. 7.

Eq. (16) is not computed immediately in the log-MAP algorithm. In fact, the function \max^* on that equation has more than the two variables present in Eq. (15). Fortunately it can be computed recursively, as shown in the Appendix.

The log-MAP algorithm uses exact formulas so its performance equals that of the BCJR algorithm although it is simpler, which means it is preferred in implementations. In turn, the max-log-MAP algorithm uses approximations, therefore its performance is slightly worse.

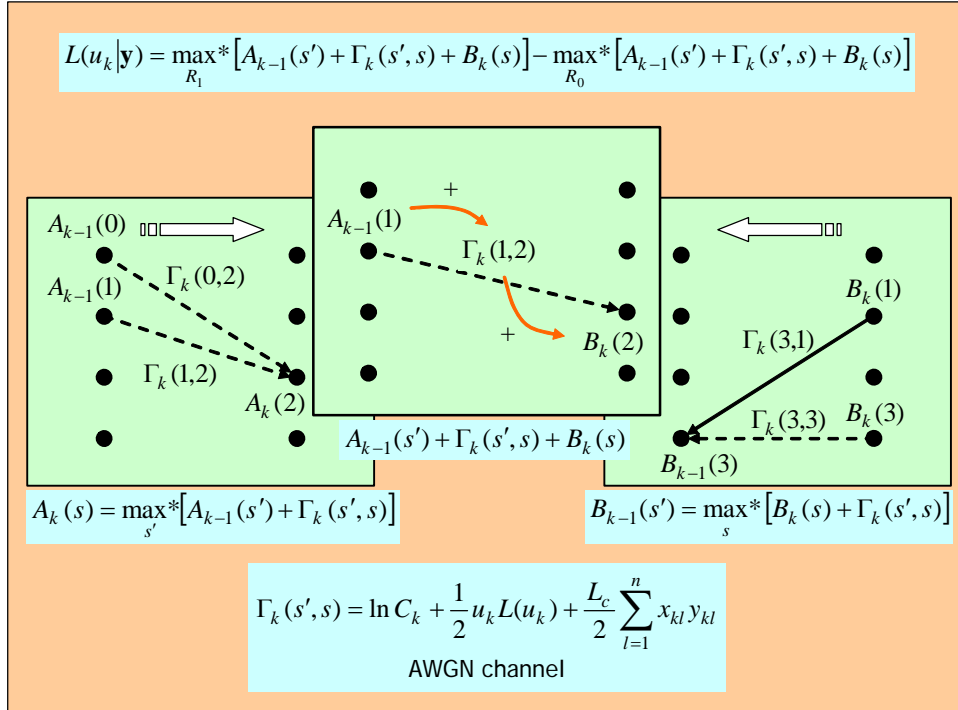


Fig. 7 Summary of expressions used in the simplified MAP algorithms

6 Application of the BCJR algorithm in iterative decoding

Let us consider a rate $1/n$ systematic convolutional encoder in which the first coded bit, x_{k1} , is equal to the information bit u_k . In that case the a posteriori log-likelihood ratio $L(u_k|\mathbf{y})$ can be decomposed into a sum of three elements (see the Appendix for details):

$$L(u_k|\mathbf{y}) = L(u_k) + L_c y_{k1} + L_e(u_k). \quad (17)$$

The first two terms in the right hand side are related with the information bit u_k . On the contrary, the third term, $L_e(u_k)$, depends only on the codeword parity bits. That is why $L_e(u_k)$ is called *extrinsic* information. This extrinsic information is an estimate of the a priori LLR $L(u_k)$. How? It is easy: we provide $L(u_k)$ and $L_{c,y_{k1}}$ as inputs to a MAP (or other) decoder and in turn we get $L(u_k | \mathbf{y})$ at its output. Then, by subtraction, we get the estimate of $L(u_k)$:

$$L_e(u_k) = L(u_k | \mathbf{y}) - L(u_k) - L_{c,y_{k1}} \quad (18)$$

This estimate of $L(u_k)$ is presumably a more accurate value of the unknown a priori LLR so it should replace the former value of $L(u_k)$. If we repeat the previous procedure in an iterative way providing $L_{c,y_{k1}}$ (again) and the new $L(u_k) = L_e(u_k)$ as inputs to another decoder we expect to get a more accurate $L(u_k | \mathbf{y})$ at its output. This fact is explored in turbo decoding, our next subject.

The turbo code inventors [6] worked with two parallel concatenated² and interleaved rate 1/2 recursive systematic convolutional codes decoded iteratively with two MAP decoders (see Figs. 8 and 9, where P and P^{-1} stand for interleaver and deinterleaver, respectively. P is also a row vector containing the permutation pattern). Both encoders in Fig. 8 are equal and their output parity bits $x_{kp}^{(1)}$ and $x_{kp}^{(2)}$ are often collected alternately through puncturing in order that an overall rate 1/2 code is obtained. The i -th element of the interleaved sequence, $u_i^{(P)}$, is merely equal to the P_i -th element of the original sequence, $u_i^{(P)} = u_{P_i}$.

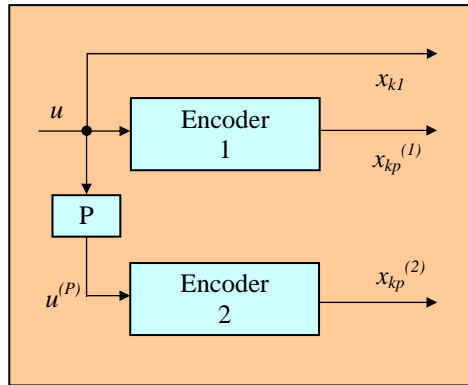


Fig. 8 The turbo encoder

Eq. (17) is the basis for iterative decoding. In the first iteration the a priori LLR $L(u_k)$ is zero if we consider equally likely input bits. The extrinsic information $L_e(u_k)$ that each decoder delivers will be used to update $L(u_k)$ from iteration to iteration and from that decoder to the other. This way the turbo decoder progressively gains more confidence on the ± 1 hard decisions the decision device will have to make in the end of the iterative process. Fig. 9 shows a simplified turbo decoder block diagram that helps illuminate the iterative procedures. We have just seen how to get an interleaved sequence $\mathbf{u}^{(P)}$ from \mathbf{u} (we compute $\mathbf{u}_i^{(P)} = \mathbf{u}_{P_i}$). To deinterleave the arbitrary sequence \mathbf{w} we simply compute $\mathbf{w}_{P_i}^{(P^{-1})} = \mathbf{w}_i$ (see the numerical example in Section 7.4).

² Serial concatenation is used as well, although not so often.

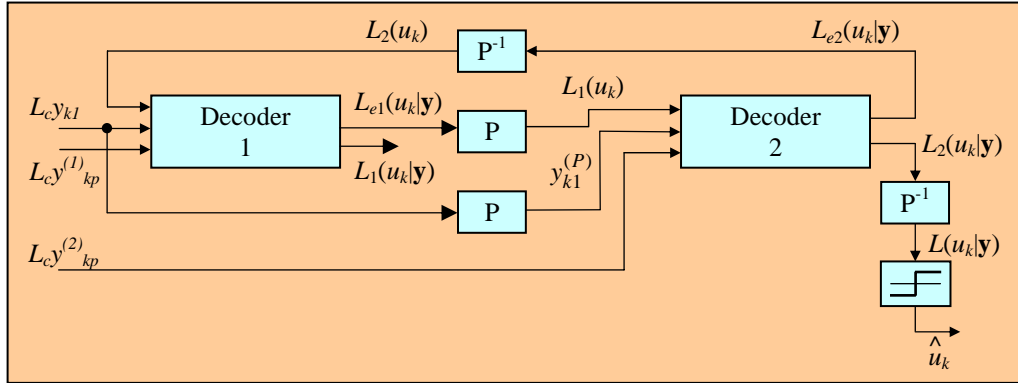


Fig. 9 A simplified block diagram of a turbo decoder

The iterative decoding proceeds as follows:

- in the first iteration we arbitrarily assume $L(u_k) = 0$, then decoder 1 outputs the extrinsic information $L_{e1}(u_k|\mathbf{y})$ on the systematic, or message, bit it gathered from the first parity bit (so note that actually decoder 2 does not need the LLR $L_1(u_k|\mathbf{y})!$);
- After appropriate interleaving the extrinsic information $L_{e1}(u_k|\mathbf{y})$ from decoder 1, computed from Eq. (18), is delivered to decoder 2 as $L_1(u_k)$, a new, more educated guess on $L(u_k)$. Then decoder 2 outputs $L_{e2}(u_k|\mathbf{y})$, its own extrinsic information on the systematic bit based on the other parity bit (note again we keep on “disdaining” the LLR!). After suitable deinterleaving, this information is delivered to decoder 1 as $L_2(u_k)$, a newer, even more educated guess on $L(u_k)$. A new iteration will then begin.
- After a prescribed number of iterations or when a stop criterium is reached the log-likelihood $L_2(u_k|\mathbf{y})$ at the output of decoder 2 is deinterleaved and delivered as $L(u_k|\mathbf{y})$ to the hard decision device, which in turn estimates the information bit based only on the sign of the deinterleaved LLR,

$$\hat{u}_k = \text{sign}[L(u_k|\mathbf{y})] = \text{sign}\{P^{-1}[L_2(u_k|\mathbf{y})]\}.$$

In iterative decoding we can use any simplifying algorithm instead of the BCJR algorithm.

7 Numerical examples

In the next examples of the BCJR algorithm and its simplifications we are going to use the same convolutional encoder we have used up until now. Suppose that upon the transmission of a sequence \mathbf{x} of six coded symbols over an AWGN channel the following sequence of twelve real numbers is received in the decoder.

$$\mathbf{y} = \underbrace{0.3 \quad 0.1}_{y_1} \quad \underbrace{-0.5 \quad 0.2}_{y_2} \quad \underbrace{0.8 \quad 0.5}_{y_3} \quad \underbrace{-0.5 \quad 0.3}_{y_4} \quad \underbrace{0.1 \quad -0.7}_{y_5} \quad \underbrace{1.5 \quad -0.4}_{y_6}$$

The encoder input bits, $u_k = \pm 1$, are equally likely and the trellis path associated with the coded sequence \mathbf{x} begins and ends in the all-zero state, for which two tail bits have to be added to the message.

The AWGN channel is such that $\frac{E_c}{N_0} = 1$ dB and $a = 1$. We know in advance the last two message (tail) bits. What is the MAP, log-MAP and max-log-MAP estimation of the other four?

7.1 Example with the BCJR algorithm

The channel reliability measure is $L_c = 4a \frac{E_c}{N_0} = 4 \times 1 \times 10^{0,1} = 5.0$. Since $P(u_k = \pm 1) = 1/2$ then

$L(u_k) = 0$ so $e^{u_k L(u_k)/2} = 1$ (cf. Eq. (9)), independently of the sign of u_k . In the same Eq. (9) we will consider $C_k = 1$ (why not? any value is good... but, attention! this way the “probabilities” $\gamma_k(s^*, s)$ we are going to compute actually may be higher than one! But, as what we want to compute is a ratio of probabilities, the final result will be the same³).

At time $k = 1$ we have the situation depicted in Fig. 10a. Thus, the values of γ are

$$\gamma_1(0,0) = C_k e^{u_k L(u_k)/2} \exp\left[\frac{L_c}{2} (x_{11} y_{11} + x_{12} y_{12})\right] = e^{[2.5 \times (-1 \times 0.3 - 1 \times 0.1)]} = e^{-1} = 0.37$$

$$\gamma_1(0,2) = e^{[2.5 \times (1 \times 0.3 + 1 \times 0.1)]} = e = 2.74$$

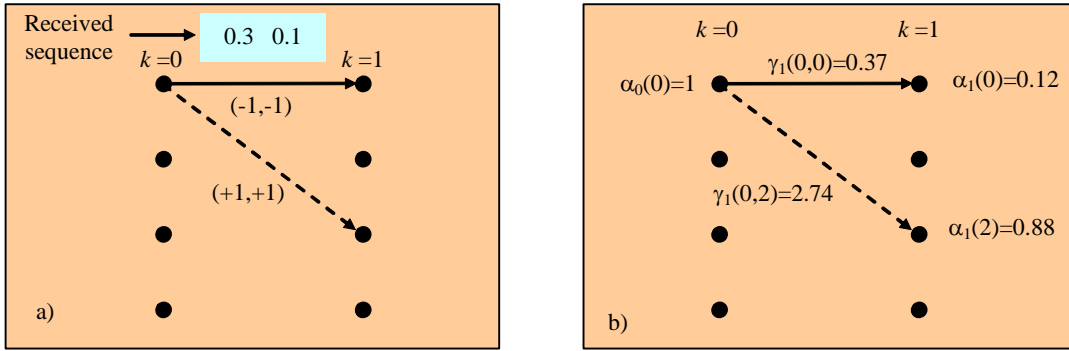


Fig. 10 Symbols received and sent at time $k = 1$ and associated probabilities α and γ .

and, therefore (see Fig. 10b),

$$\alpha_0(0)\gamma_1(0,0) = 1 \times 0.37 = 0.37 \quad \Rightarrow \quad \alpha_1(0) = \frac{0.37}{0.37 + 2.74} = 0.12$$

$$\alpha_0(0)\gamma_1(0,2) = 1 \times 2.74 = 2.74 \quad \Rightarrow \quad \alpha_1(2) = \frac{2.74}{0.37 + 2.74} = 0.88$$

The calculation of α and γ will progress in an identical way as the sequence is being received⁴. When it reaches the end it is the time to compute β backwards to the trellis beginning. The final situation with all the values of α , β and γ is portrayed in Fig. 11. Let us see a final example of how to compute α , β and γ at time $k = 3$:

$$\gamma_3(2,3) = e^{[2.5 \times (-1 \times (0.8) + 1 \times 0.5)]} = e^{-0.75} = 0.47$$

$$\alpha_3(3) = \frac{\alpha_2(2)\gamma_3(2,3) + \alpha_2(3)\gamma_3(3,3)}{4.31} = \frac{0.01 \times 0.47 + 0.92 \times 2.13}{4.31} = 0.45$$

$$\beta_2(2) = \frac{\beta_3(1)\gamma_3(2,1) + \beta_3(3)\gamma_3(2,3)}{9.96} = \frac{0.69 \times 2.13 + 0.001 \times 0.47}{9.96} = 0.15$$

³ However, care must be taken if overflow or underflow problems might occur. In that case it would be wiser to use the actual values of C_k .

⁴ There are at most $2^n = 4$ different values of γ at each time k since there are at most $2^n = 4$ different codewords.

$$\begin{aligned}\Sigma_{P_3} &= \sum_{R_0, R_1} \alpha_2(s') \gamma_3(s', s) \beta_3(s) = \\ &= \sum_{R_0} \alpha_2(s') \gamma_3(s', s) \beta_3(s) + \sum_{R_1} \alpha_2(s') \gamma_3(s', s) \beta_3(s) = 0.493 + 0.068 \approx 0.56\end{aligned}$$

The values of Σ_{P_k} , $\sum_{R_0 \text{ or } R_1} P(s', s, \mathbf{y})$ and $\sum_{R_0 \text{ or } R_1} P_{norm}(s', s, \mathbf{y})$ are to be found at Table 1.

Table 1 – Values of Σ_{P_k} , $\Sigma P(s', s, \mathbf{y})$ and $\Sigma P_{norm}(s', s, \mathbf{y})$

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
Unnormalized						
$\sum_{R_1} P(s', s, \mathbf{y})$	1.214	0.177	0.068	0.306	0.000	0.000
$\sum_{R_0} P(s', s, \mathbf{y})$	0.203	0.139	0.493	0.001	0.379	8.084
Σ_{P_k}	1.417	0.316	0.562	0.307	0.379	8.084
Normalized						
	↓	↓	↓	↓	↓	↓
$\sum_{R_1} P_{norm}(s', s, \mathbf{y})$	0.857	0.560	0.122	0.996	0.000	0.000
$\sum_{R_0} P_{norm}(s', s, \mathbf{y})$	0.143	0.440	0.878	0.004	1.000	1.000

We are on the brink of getting the wanted values of $L(u_k | \mathbf{y})$ given by Eq. (14): we collect the values of the last two rows of Table 1 and obtain

$$L(u_1 | \mathbf{y}) = \ln \frac{P_{norm}(0, 2, \mathbf{y})}{P_{norm}(0, 0, \mathbf{y})} = \ln \frac{0.857}{0.143} = 1.79$$

$$L(u_2 | \mathbf{y}) = \ln \frac{P_{norm}(0, 2, \mathbf{y}) + P_{norm}(2, 3, \mathbf{y})}{P_{norm}(0, 0, \mathbf{y}) + P_{norm}(2, 1, \mathbf{y})} = \ln \frac{0.560}{0.440} = 0.24$$

$$L(u_3 | \mathbf{y}) = \ln \frac{P_{norm}(0, 2, \mathbf{y}) + P_{norm}(1, 2, \mathbf{y}) + P_{norm}(2, 3, \mathbf{y}) + P_{norm}(3, 3, \mathbf{y})}{P_{norm}(0, 0, \mathbf{y}) + P_{norm}(1, 0, \mathbf{y}) + P_{norm}(2, 1, \mathbf{y}) + P_{norm}(3, 1, \mathbf{y})} = \ln \frac{0.122}{0.878} = -1.98$$

$$L(u_4 | \mathbf{y}) = \ln \frac{P_{norm}(0, 2, \mathbf{y}) + P_{norm}(1, 2, \mathbf{y}) + P_{norm}(2, 3, \mathbf{y}) + P_{norm}(3, 3, \mathbf{y})}{P_{norm}(0, 0, \mathbf{y}) + P_{norm}(1, 0, \mathbf{y}) + P_{norm}(2, 1, \mathbf{y}) + P_{norm}(3, 1, \mathbf{y})} = \ln \frac{0.996}{0.004} = 5.56$$

$$L(u_5 | \mathbf{y}) = \ln \frac{0}{P(0, 0, \mathbf{y}) + P(1, 0, \mathbf{y}) + P(2, 1, \mathbf{y}) + P(3, 1, \mathbf{y})} = -\infty$$

$$L(u_6 | \mathbf{y}) = \ln \frac{0}{P(0, 0, \mathbf{y}) + P(1, 0, \mathbf{y})} = -\infty$$

The last two LLR values result from the enforced trellis termination. In face of all the values obtained the decoder hard decision about the message sequence will be $\hat{\mathbf{u}} = +1 \ +1 \ -1 \ +1 \ -1 \ -1$ or, with $\{0, 1\}$ values,

$$1 \ 1 \ 0 \ 1 \ 0 \ 0.$$

The path estimated by the BCJR algorithm is enhanced in Fig. 13.

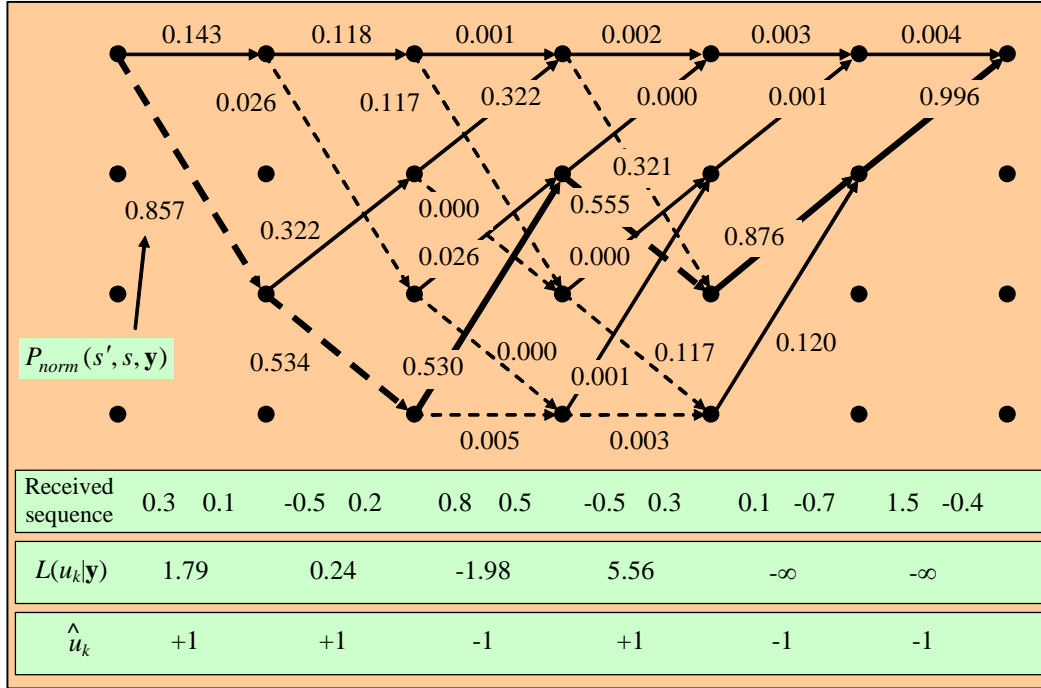


Fig. 13 Values of $P_{norm}(s', s, \mathbf{y})$ and $L(u_k|\mathbf{y})$, and estimate of u_k .

7.2 Example with the max-log-MAP algorithm

The values of A, B and Γ shown in Fig. 14 were obtained following the procedures previously described. Just to exemplify let us see how $A_4(1)$ and $B_3(2)$ in that figure were computed:

$$\begin{aligned} A_4(1) &= \max[A_3(2) + \Gamma_4(2,1), A_3(3) + \Gamma_4(3,1)] = \\ &= \max(2.01 - 2.01, 3.52 + 2.01) = 5.54 \end{aligned}$$

$$\begin{aligned} B_3(2) &= \max[B_4(1) + \Gamma_4(2,1), B_4(3) + \Gamma_4(2,3)] = \\ &= \max(-4.28 - 2.01, 0.76 + 2.01) = 2.77 \end{aligned}$$

The sums $A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)$ are computed according to Fig. 7. Now we just have to find the maximum value of those sums at each time k and for each set R_0 and R_1 . Subtracting the first value from the second we get the log-likelihood ratios of Table 2 and the corresponding u_k estimate.

Table 2 – Values obtained with the max-log-MAP algorithm

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
$\max_{R_1}()$	7.302	7.302	5.791	7.302	$-\infty$	$-\infty$
$\max_{R_0}()$	5.791	6.798	7.302	1.762	7.302	7.302
$L(u_k \mathbf{y})$	1.511	0.504	-1.511	5.539	$-\infty$	$-\infty$
u_k estimate	+1	+1	-1	+1	-1	-1

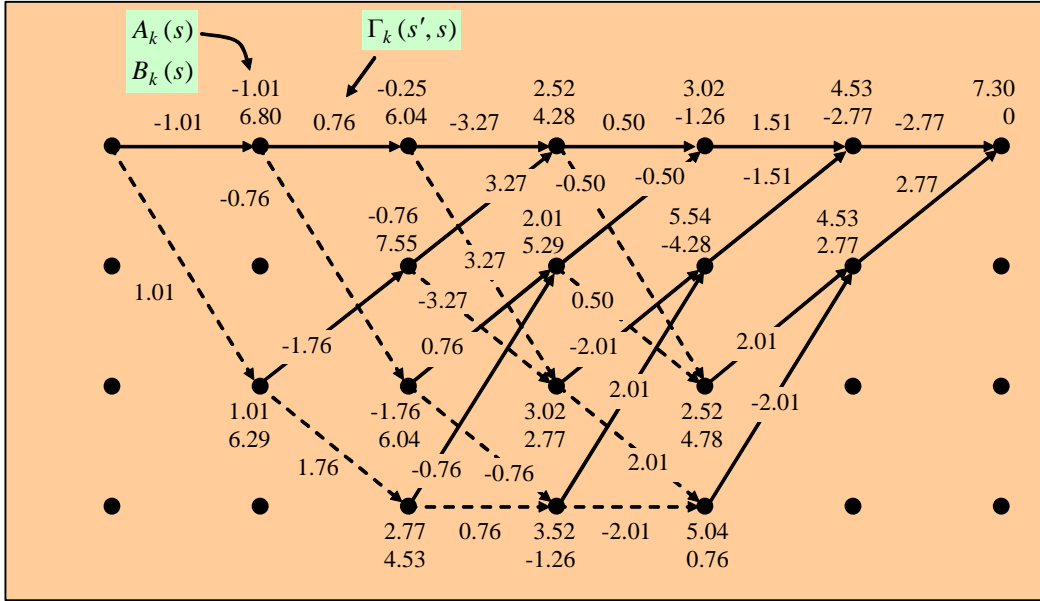


Fig. 14 Values of A , B and Γ after all the six received symbols have been processed.

Fig. 15 highlights the trellis branches that correspond to the maximum values in each set. The only uninterrupted path is the one corresponding to the estimated information bits.

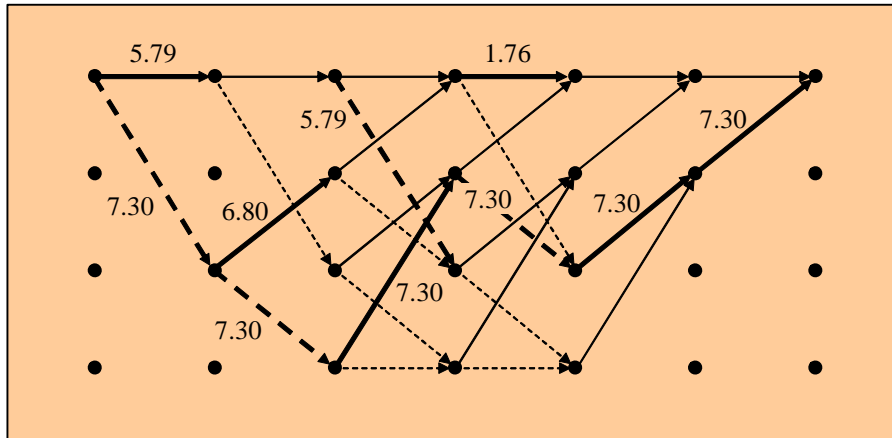


Fig. 15 The max-log-MAP algorithm: maximum values of “ $A + \Gamma + B$ ” in each set R_0 and R_1 .

7.3 Example with the log-MAP algorithm

In this case A and B are computed without any approximations. Table 3 is calculated from them and from Γ (which is equal in both simplified methods). As it should be expected, we have got the same values for $L(u_k|y)$ as we had before with the BCJR algorithm.

Table 3 – Values obtained with the log-MAP algorithm

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$
$\max_{R_1}()$	7.783	7.311	5.791	7.349	$-\infty$	$-\infty$
$\max_{R_0}()$	5.996	6.805	7.303	1.765	7.805	7.934
$L(u_k \mathbf{y})$	1.79	0.24	-1.98	5.56	$-\infty$	$-\infty$
u_k estimate	+1	+1	-1	+1	-1	-1

7.4 Example of turbo decoding

Now let us consider the following conditions:

- An all-zero 9-bit message sequence is applied to two equal recursive systematic convolutional encoders, each with generator matrix $\mathbf{G}(x) = [1 \ (1+x^2)/(1+x+x^2)]$. The output sequence is obtained through puncturing so the turbo encoder's code rate is 1/2. The RSC encoder trellis is shown in Fig. 16.
- The interleaver pattern is $\mathbf{P} = [1 \ 4 \ 7 \ 2 \ 5 \ 9 \ 3 \ 6 \ 8]$. Thus, for example, the third element of the interleaved version of arbitrary sequence $\mathbf{m} = [2 \ 4 \ 3 \ 1 \ 8 \ 5 \ 9 \ 6 \ 0]$ is $\mathbf{m}_3^{(P)} = \mathbf{m}_{P_3} = \mathbf{m}_7 = 9$. Therefore, $\mathbf{m}^{(P)} = [2 \ 1 \ 9 \ 4 \ 8 \ 0 \ 3 \ 5 \ 6]$.
- The AWGN channel is such that $E_c/N_0 = 0.25$ and $a = 1$. So, $L_c = 1$.
- The 18-element received sequence is

$$\mathbf{y} = 0.3 \ -4.0 \ -1.9 \ -2.0 \ -2.4 \ -1.3 \ 1.2 \ -1.1 \ 0.7 \ -2.0 \ -1.0 \ -2.1 \ -0.2 \ -1.4 \ -0.3 \ -0.1 \ -1.1 \ 0.3.$$

- The initial a priori log-likelihood ratio is assumed to be $L(u_k) = 0$.

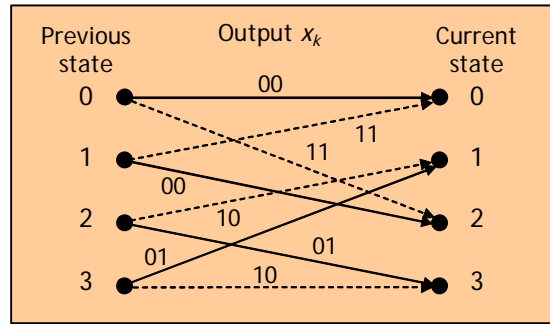


Fig. 16 The trellis of the RSC encoder characterized by $\mathbf{G}(x) = [1 \ (1+x^2)/(1+x+x^2)]$

Therefore, we will have the following input sequences in the turbo decoder of Fig. 9:

k	1	2	3	4	5	6	7	8	9
$L_c \mathbf{y}_{k1}$	0.3	-1.9	-2.4	1.2	0.7	-1.0	-0.2	-0.3	-1.1
$L_c \mathbf{y}_{k1}^{(P)}$	0.3	1.2	-0.2	-1.9	0.7	-1.1	-2.4	-1.0	-0.3
$L_c \mathbf{y}_{kp}^{(1)}$	-4.0	0	-1.3	0	-2.0	0	-1.4	0	0.3
$L_c \mathbf{y}_{kp}^{(2)}$	0	-2.0	0	-1.1	0	-2.1	0	-0.1	0

In the first decoding iteration we would get the following values of the a posteriori log-likelihood ratio $L_1(u_k|\mathbf{y})$ and of the extrinsic information $L_{e2}(u_k)$, both at the output of decoder 1, when we have $L(u_k)$ and $L_c \mathbf{y}_{k1}$ at its input:

$L_1(u_k \mathbf{y})$	-4.74	-3.20	-3.66	1.59	1.45	-0.74	0.04	0.04	-1.63
$L(u_k)$	0	0	0	0	0	0	0	0	0
$L_c \mathbf{y}_{k1}$	0.30	-1.90	-2.40	1.20	0.70	-1.00	-0.20	-0.30	-1.10
$L_{e1}(u_k)$	-5.04	-1.30	-1.26	0.39	0.75	0.26	0.24	0.34	-0.53
$L_1(u_k)$	-5.04	0.39	0.24	-1.30	0.75	-0.53	-1.26	0.26	0.34

The values of the extrinsic information (fourth row) were computed subtracting the second and third rows from the first (Eq. (18)). As we see, the soft information $L_1(u_k|\mathbf{y})$ provided by decoder 1 would result in four errored bits, those occurring when $L_1(u_k|\mathbf{y})$ is positive (at times $k = 4, 5, 7$ and 8). This same decoder would pass the extrinsic information $L_{e1}(u_k)$ to decoder 2 after convenient interleaving – that is, $L_1(u_k)$ in the last row. Note that after this half-iteration we have gained a strong confidence on the decision about the first bit due to the large negative values of $L_1(u_1|\mathbf{y})$ and $L_{e1}(u_1)$ – most probably $u_1 = -1$ (well, we already know that...). We are not so certain about the other bits, especially those with small absolute values of $L_1(u_1|\mathbf{y})$ and $L_{e1}(u_1)$.

Decoder 2 will now work with the interleaved systematic values $L_c \mathbf{y}_{k1}^{(P)}$, the parity values $\mathbf{y}_{kp}^{(2)}$ and the new a priori LLRs $L_1(u_k)$. The following table presents the decoding results according to Eq. (18), where sequence $L_2(u_k)$ represents the deinterleaved version of $L_{e2}(u_k)$ that will act as a more educated guess of the a priori LLR $L(u_k)$. So $L_2(u_k)$ will be used as one of the decoder 1 inputs in the next iteration.

$L_2(u_k \mathbf{y})$	-3.90	0.25	0.18	-3.04	1.23	-1.44	-3.65	-0.72	0.04
$L_1(u_k)$	-5.04	0.39	0.24	-1.30	0.75	-0.53	-1.26	0.26	0.34
$L_c \mathbf{y}_{k1}^{(P)}$	0.30	1.20	-0.20	-1.90	0.70	-1.10	-2.40	-1.00	-0.30
$L_{e2}(u_k)$	0.85	-1.34	0.14	0.16	-0.22	0.19	0.01	0.02	0.00
$L_2(u_k)$	0.85	0.16	0.01	-1.34	-0.22	0.02	0.14	0.00	0.19

How was $L_2(u_k)$ computed? As said before, the P_i -th element of $L_2(u_k)$ is the i -th element of the original sequence $L_{e2}(u_k)$. For example, the fourth element of $L_2(u_k)$ is equal to

$$[L_2(u_k)]_4 = [L_{e2}(u_k)]_{P_2} = [L_{e2}(u_k)]_2 = -1.34$$

because $4 = P_2$.

We can see again that we would still get four wrong bits if we made a hard decision on sequence $L_2(u_k|\mathbf{y})$ after it is reorganized in the correct order through deinterleaving. That sequence, -3.90 -3.04 -3.65 0.25 1.23 -0.72 0.18 0.04 -1.44, enforces errors in positions 4, 5, 7 and 8.

The previous procedures should be repeated iteration after iteration. For example, we would get Table 4 during five iterations. Shaded (positive) values would yield wrong hard decisions.

All four initially wrong bits have been corrected just after the third iteration. From then on it is a waste of time to proceed further. In fact, the values of the a posteriori log-likelihood ratios stabilize very quickly and we gain nothing if we go on with the decoding.

Table 4 – Outputs of turbo decoders during five iterations

Iteration	$k \rightarrow$	1	2	3	4	5	6	7	8	9
1	$L_1(u_k \mathbf{y})$	-4.74	-3.20	-3.66	1.59	1.45	-0.74	0.04	0.04	-1.63
2	$L_1(u_k \mathbf{y})$	-3.64	-2.84	-3.28	0.11	0.27	-0.95	-0.17	-0.25	-1.40
3	$L_1(u_k \mathbf{y})$	-3.65	-3.00	-3.35	-0.58	-0.34	-1.07	-0.61	-0.63	-1.53
4	$L_1(u_k \mathbf{y})$	-3.85	-3.21	-3.49	-1.02	-0.74	-1.20	-0.93	-0.90	-1.75
5	$L_1(u_k \mathbf{y})$	-4.08	-3.42	-3.64	-1.35	-1.05	-1.32	-1.18	-1.11	-1.95
1	$L(u_k \mathbf{y})$	-3.90	-3.04	-3.65	0.25	1.23	-0.72	0.18	0.04	-1.44
2	$L(u_k \mathbf{y})$	-3.61	-2.96	-3.29	-0.41	0.13	-0.97	-0.43	-0.25	-1.48
3	$L(u_k \mathbf{y})$	-3.75	-3.11	-3.35	-0.87	-0.45	-1.08	-0.80	-0.63	-1.66
4	$L(u_k \mathbf{y})$	-3.98	-3.32	-3.50	-1.22	-0.85	-1.21	-1.07	-0.90	-1.86
5	$L(u_k \mathbf{y})$	-4.21	-3.52	-3.65	-1.51	-1.15	-1.33	-1.28	-1.11	-2.06

Fig. 17 shows graphically the evolution of $L_1(u_k|\mathbf{y})$ and $L(u_k|\mathbf{y})$ with iterations, where we observe all sign changes have occurred in the beginning of decoding.

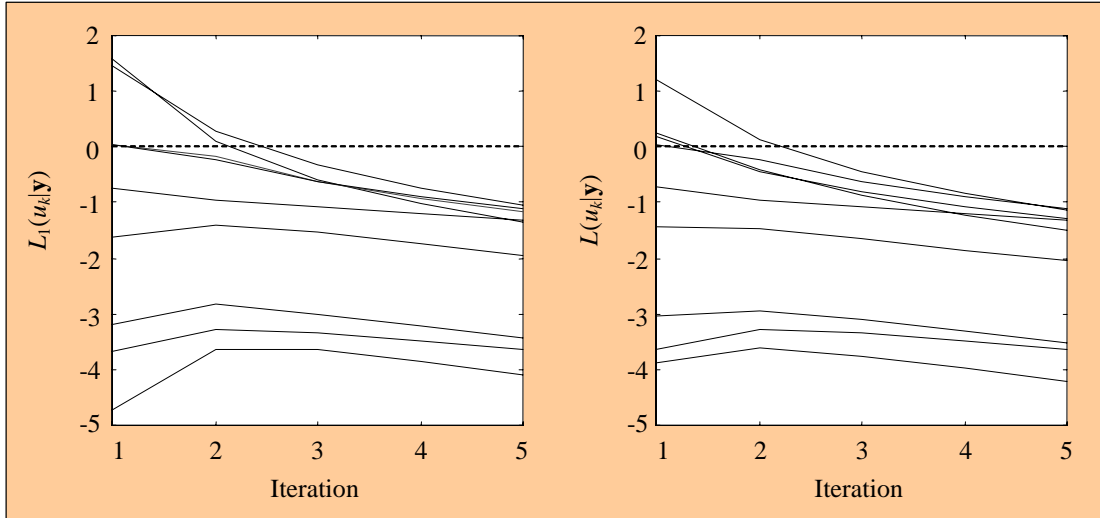


Fig. 17 The a posteriori log-likelihood ratios $L_1(u_k|\mathbf{y})$ (first decoder) and $L(u_k|\mathbf{y})$ (second decoder)

8 Appendix: the theory underlying the BCJR, the log-MAP and the max-log-MAP algorithms

In this Appendix the mathematical developments of the BCJR, log-MAP and max-log-MAP algorithms will be presented. All three estimate the encoder's input bits after time, in a bit-to-bit basis, contrarily to the Viterbi algorithm, which finds the maximum likelihood path through the trellis and from it estimates the transmitted sequence.

Our framework is the same as before: a rate $1/n$ convolutional code with M states, an N -bit message sequence \mathbf{u} , a codeword sequence \mathbf{x} transmitted over an AWGN channel and a received sequence \mathbf{y} . To illustrate the developments we will use $M = 4$ states.

We shall begin with a mathematical relation between conditional probabilities that we shall find to be useful:

$$P(A, B|C) = P(A|B, C)P(B|C) \quad (19)$$

This relation is proved applying Bayes' rule $P(A, B) = P(A|B)P(B)$ and defining the sets of events $D = \{A, B\}$ e $E = \{B, C\}$:

$$\begin{aligned}
P(A, B|C) &= P(D|C) = \frac{P(D, C)}{P(C)} = \\
&= \frac{P(A, B, C)}{P(C)} = \frac{P(A, E)}{P(C)} = \\
&= \frac{P(A|E)P(E)}{P(C)} = \frac{P(A|E)P(B, C)}{P(C)} = \\
&= \frac{P(A|E)P(B|C)P(C)}{P(C)} = P(A|B, C)P(B|C)
\end{aligned}$$

8.1 The BCJR, or MAP, algorithm

In the next sections we will show how to calculate theoretically the variables present in the algorithm. A final section will show how the algorithm can be applied in the iterative decoding of concatenated codes, of which turbo codes are the most noteworthy example.

8.1.1 The a posteriori log-likelihood ratio (LLR) $L(u_k|\mathbf{y})$

We want to calculate $L(u_k|\mathbf{y}) = \ln \frac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})}$. If $L(u_k|\mathbf{y}) > 0$ the decoder will estimate the bit

$u_k = +1$ was sent, otherwise it will estimate $u_k = -1$ was sent. Now let us observe Fig. 18. It shows a four state trellis section between times $k-1$ and k . There are $2M = 8$ transitions between states $S_{k-1} = s'$ and next states $S_k = s$: four result from a -1 input bit (those drawn with a thick line) and the other four result from a +1 input bit. Each one of these transitions is due unequivocally to a known bit (just notice in the trellis if the branch line is solid or dashed). Therefore, the probabilities that $u_k = -1$ or $u_k = +1$, given our knowledge of the sequence \mathbf{y} , are equal to the probabilities that the transitions (s', s) between states correspond to a solid or a dashed line, respectively. Since transitions are mutually exclusive – just one can occur at a given time – the probability that any one of them occurs is equal to the sum of the individual probabilities, that is,

$$P(u_k = -1|\mathbf{y}) = \sum_{R_0} P(s', s|\mathbf{y})$$

$$P(u_k = +1|\mathbf{y}) = \sum_{R_1} P(s', s|\mathbf{y}),$$

where R_0 and R_1 represent, respectively, the set of transitions from state $S_{k-1} = s'$ to state $S_k = s$ originated by $u_k = -1$ or $u_k = +1$, as Fig. 18 shows. Thus we get:

$$\begin{aligned}
L(u_k|\mathbf{y}) &= \ln \frac{P(u_k = +1|\mathbf{y})}{P(u_k = -1|\mathbf{y})} = \ln \frac{\sum_{R_1} P(s', s|\mathbf{y})}{\sum_{R_0} P(s', s|\mathbf{y})} = \\
&= \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})/P(\mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})/P(\mathbf{y})} = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})}
\end{aligned} \tag{20}$$

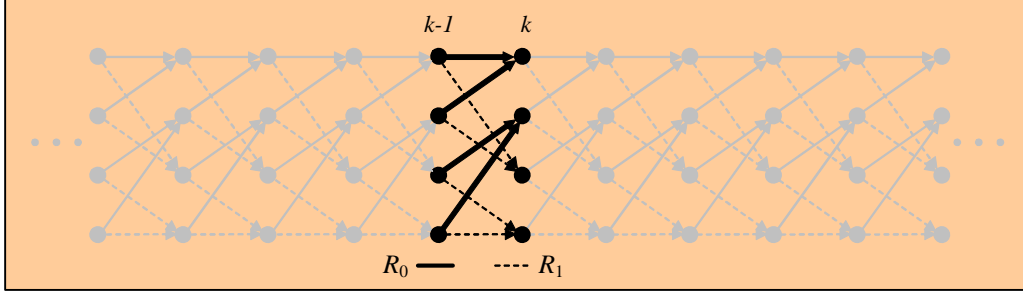


Fig. 18 Trellis section between times $k-1$ and k

8.1.2 $P(s', s, \mathbf{y})$ is a product $\alpha\gamma\beta$

The received N symbol sequence \mathbf{y} can be partitioned into three subsequences: a past sequence, $\mathbf{y}_{<k}$, a current sequence (just one symbol), \mathbf{y}_k , and a future sequence, $\mathbf{y}_{>k}$, as in Eq. (3). Writing $P(s', s, \mathbf{y}) = P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k, \mathbf{y}_{>k})$ and using Bayes' rule we have

$$\begin{aligned} P(s', s, \mathbf{y}) &= P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k, \mathbf{y}_{>k}) = \\ &= P(\mathbf{y}_{>k} | s', s, \mathbf{y}_{<k}, \mathbf{y}_k) P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k) \end{aligned}$$

It happens that if we know the current state $S_k = s$ the received sequence after time k , $\mathbf{y}_{>k}$, will depend neither on the previous state s' nor on the past or current sequence, $\mathbf{y}_{<k}$ e \mathbf{y}_k , since we are considering a memoryless channel. Therefore,

$$P(s', s, \mathbf{y}) = P(\mathbf{y}_{>k} | s) P(s', s, \mathbf{y}_{<k}, \mathbf{y}_k).$$

Developing the right hand side and noting again that in a memoryless channel if we know the previous state s' the next state s and the current symbol y_k do not depend on the past sequence $\mathbf{y}_{<k}$, we get

$$\begin{aligned} P(s', s, \mathbf{y}) &= P(\mathbf{y}_{>k} | s) P(\mathbf{y}_k, s | s', \mathbf{y}_{<k}) P(s', \mathbf{y}_{<k}) = \\ &= \underbrace{P(\mathbf{y}_{>k} | s)}_{\beta_k(s)} \underbrace{P(\mathbf{y}_k, s | s')}_{\gamma_k(s', s)} \underbrace{P(s', \mathbf{y}_{<k})}_{\alpha_{k-1}(s')} = \\ &= \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s) \end{aligned}$$

Probability $\alpha_{k-1}(s') = P(s', \mathbf{y}_{<k})$ represents the joint probability that at time $k-1$ the state is s' and the received sequence until then is $\mathbf{y}_{<k}$. Probability $\gamma_k(s', s) = P(\mathbf{y}_k, s | s')$ represents the probability that next state is s and the received symbol is y_k given the previous state is s' . Finally, probability $\beta_k(s) = P(\mathbf{y}_{>k} | s)$ is the conditional probability that, given the current state is s , the future sequence will be $\mathbf{y}_{>k}$.

Thus we will have

$$L(u_k | \mathbf{y}) = \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)}$$

As explained in the main text, it is convenient to normalize $P(s', s, \mathbf{y})$. So, instead of using the simple three-factor expression $\alpha\gamma\beta$ we should use the “safer” expression

$$P_{norm}(s', s, \mathbf{y}) = \frac{P(s', s, \mathbf{y})}{\sum_{R_0, R_1} P(s', s, \mathbf{y})} = \frac{P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y}) + \sum_{R_1} P(s', s, \mathbf{y})}$$

It would be easy to show that this will not have any effect on the final value of $L(u_k|\mathbf{y})$, as it should be expected.

8.1.3 The present: calculation of γ

With the help of the conditional probabilities relation of Eq. (19) we get for $\gamma_k(s', s)$

$$\begin{aligned} \gamma_k(s', s) &= P(\mathbf{y}_k, s | s') = \\ &= P(\mathbf{y}_k | s', s) P(s | s') = \end{aligned}$$

Let us understand the meaning of each of these two factors. If state s' is known the probability that at the next time one of the two possible states s is reached is equal to the probability that the input bit is $u_k = \pm 1$ (that is, the trellis branch is a solid or a dashed line). So, $P(s | s') = P(u_k)$. For instance, if $P(u_k = \pm 1) = 1/2$ and we are in a given state, it is equally likely to reach one of the next states or the other.

However, if $P(u_k = +1) = 3/5$ then $P(s' \xrightarrow[\text{line}]{\text{dashed}} s) = 3/5$. As far as the other factor, $P(\mathbf{y}_k | s', s)$, is concerned we should note that the joint occurrence of the consecutive states $S_{k-1} = s'$ e $S_k = s$ is equivalent to the occurrence of the corresponding coded symbol x_k , that is, $P(\mathbf{y}_k | s', s) = P(\mathbf{y}_k | x_k)$. Therefore,

$$\gamma_k(s', s) = P(\mathbf{y}_k | x_k) P(u_k) \quad (21)$$

Let us go back to $P(u_k)$. From the definition $L(u_k) = \ln \frac{P(u_k = +1)}{P(u_k = -1)} = \ln \frac{P(u_k = +1)}{1 - P(u_k = +1)}$ we get immediately

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

which can be conveniently expressed as

$$\begin{aligned} P(u_k = \pm 1) &= \frac{e^{L(u_k)/2}}{1 + e^{L(u_k)}} e^{u_k L(u_k)/2} = \\ &= C_{1k} e^{u_k L(u_k)/2} \end{aligned} \quad (22)$$

The fraction $C_{1k} = e^{L(u_k)/2} / [1 + e^{L(u_k)}]$ is equal to 1/2 if the bits $u_k = \pm 1$ are equally likely. Either way, C_{1k} does not depend on u_k being +1 or -1 and so, since it appears both in the numerator and denominator of the a posteriori LLR expression (Eq. (20)), it will cancel out in that computation. We will come back to the issue shortly.

The probability $P(\mathbf{y}_k | x_k)$ that n values $y_k = y_{k1} y_{k2} \dots y_{kn}$ are received given n values $x_k = x_{k1} x_{k2} \dots x_{kn}$ were transmitted will be equal to the product of the individual probabilities $P(y_{kl} | x_{kl})$, $l = 1, 2, \dots, n$. In fact, in a memoryless channel the successive transmissions are statistically independent:

$$P(\mathbf{y}_k | \mathbf{x}_k) = \prod_{l=1}^n P(y_{kl} | x_{kl}). \quad (23)$$

In the very usual case of $n = 2$ we simply have $P(\mathbf{y}_k | \mathbf{x}_k) = P(y_{k1} | x_{k1}) P(y_{k2} | x_{k2})$.

These probabilities depend on the channel and the modulation in use. With BPSK modulation the transmitted signals have amplitudes $x_{kl} E_c = \pm E_c$, where E_c is the energy transmitted per coded bit. Let us consider an AWGN channel with noise power spectral density $N_0/2$ and fading amplitude a . At the receiver's matched filter output the signal amplitude is now $y'_{kl} = ax_{kl} \sqrt{E_c} + n' = \pm a \sqrt{E_c} + n'$, where n' is a sample of gaussian noise with zero mean and variance $\sigma_{n'}^2 = N_0/2$. Normalizing amplitudes in the receiver we get

$$y_{kl} = \frac{y'_{kl}}{\sqrt{E_c}} = ax_{kl} + \frac{n'}{\sqrt{E_c}} = ax_{kl} + n,$$

where now the variance of noise $n = n'/\sqrt{E_c}$ is $\sigma_n^2 = \sigma_{n'}^2/E_c = N_0/2E_c$. We will have, finally,

$$P(y_{kl} | x_{kl}) = \frac{1}{\sqrt{\pi N_0/E_c}} e^{-\frac{E_c}{N_0} (y_{kl} - ax_{kl})^2}.$$

If we expand Eq. (23) we get

$$\begin{aligned} P(\mathbf{y}_k | \mathbf{x}_k) &= \left[\frac{1}{(\sqrt{\pi N_0/E_c})^n} \exp\left(-\frac{E_c}{N_0} \sum_{l=1}^n y_{kl}^2\right) \exp\left(-\frac{E_c}{N_0} a^2 \sum_{l=1}^n x_{kl}^2\right) \right] \exp\left(2a \frac{E_c}{N_0} \sum_{l=1}^n x_{kl} y_{kl}\right) = \\ &= C_{2k} \exp\left(2a \frac{E_c}{N_0} \sum_{l=1}^n x_{kl} y_{kl}\right) \end{aligned}$$

The product of factors $C_{2k} = \frac{1}{(\sqrt{\pi N_0/E_c})^n} \exp\left(-\frac{E_c}{N_0} \sum_{l=1}^n y_{kl}^2\right) \exp\left(-\frac{E_c}{N_0} a^2 \sum_{l=1}^n x_{kl}^2\right)$ does not depend

either on the u_k sign or the codeword x_k . In fact, the first factor of the product depends only on the channel,

the second one depends on the channel and the sequence y_k and the third one, in which $\sum_{l=1}^n x_{kl}^2 = n$,

depends on the channel and the fading amplitude. This means C_{2k} will appear both in the numerator and denominator of Eq. (20) and cancels out.

The summation $\sum_{l=1}^n x_{kl} y_{kl}$ represents the sum of the element-by-element product of transmitted (x_k) and received (y_k) words. If we see each of these words as an n -element vector the summation is just their inner product:

$$\begin{aligned} \mathbf{x}_k &= [x_{k1} x_{k2} \dots x_{kn}] \\ \mathbf{y}_k &= [y_{k1} y_{k2} \dots y_{kn}] \end{aligned} \quad \Rightarrow \quad \sum_{l=1}^n x_{kl} y_{kl} = \mathbf{x}_k \bullet \mathbf{y}_k$$

We had seen that $L_c = 4a \frac{E_c}{N_0}$. Thus,

$$\begin{aligned} P(\mathbf{y}_k | x_k) &= C_{2k} \exp\left(\frac{L_c}{2} \sum_{l=1}^n x_{kl} y_{kl}\right) = \\ &= C_{2k} \exp\left(\frac{L_c}{2} \mathbf{x}_k \bullet \mathbf{y}_k\right) \end{aligned}$$

Going back to Eq. (21) we can now write the final expression of $\gamma_k(s', s)$ as it appears in Eq. (9):

$$\begin{aligned} \gamma_k(s', s) &= P(\mathbf{y}_k | x_k) P(u_k) = \\ &= C_{2k} \exp\left(\frac{L_c}{2} \sum_{l=1}^n x_{kl} y_{kl}\right) C_{1k} e^{u_k L(u_k)/2} = \\ &= C_k e^{u_k L(u_k)/2} \exp\left(\frac{L_c}{2} \sum_{l=1}^n x_{kl} y_{kl}\right) \end{aligned}$$

where we have done $C_k = C_{1k} C_{2k}$. C_k cancels out in the end because C_{1k} and C_{2k} do.

We already have $\gamma_k(s', s)$ and we need α and β . Let us begin with α .

8.1.4 The past: calculation of α

By definition $\alpha_{k-1}(s') = P(s', \mathbf{y}_{<k})$, which we conveniently write as $\alpha_k(s) = P(s, \mathbf{y}_{<k+1})$. However

$$\begin{aligned} \alpha_k(s) &= P(s, \mathbf{y}_{<k+1}) = \\ &= P(s, \mathbf{y}_{<k}, \mathbf{y}_k) \end{aligned}$$

From Probability Theory we know that $P(A) = \sum_B P(A, B)$, where the summation adds over all possible values of B . Therefore,

$$\begin{aligned} \alpha_k(s) &= P(s, \mathbf{y}_{<k}, \mathbf{y}_k) = \\ &= \sum_{s'} P(s, s', \mathbf{y}_{<k}, \mathbf{y}_k) \end{aligned}$$

The same old memoryless channel argument allows us to write

$$\begin{aligned}\sum_{s'} P(s, s', \mathbf{y}_{<k}, \mathbf{y}_k) &= \sum_{s'} P(s, \mathbf{y}_k | s', \mathbf{y}_{<k}) P(s', \mathbf{y}_{<k}) = \\ &= \sum_{s'} P(s, \mathbf{y}_k | s') P(s', \mathbf{y}_{<k}) = \sum_{s'} \gamma_k(s', s) \alpha_{k-1}(s')\end{aligned}$$

However, in order to avoid overflow or underflow it is more convenient to normalize α to its sum over all state transitions, that is, computing

$$\alpha_k(s) = \frac{\sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s)}{\sum_s \sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s)} = \frac{\alpha'_k(s)}{\sum_s \alpha'_k(s)}$$

This normalization does not alter the final LLR result. Now we only have to start from the appropriate initial conditions. In an all-zero terminated trellis they are:

$$\alpha_0(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases}$$

8.1.5 The future: calculation of β

The recursive formula for β is obtained the same way. By definition $\beta_k(s) = P(\mathbf{y}_{>k} | s)$, which we conveniently write as $\beta_{k-1}(s') = P(\mathbf{y}_{>k-1} | s')$. The memoryless argument will lead us now to

$$\begin{aligned}\beta_{k-1}(s') &= P(\mathbf{y}_{>k-1} | s') = \\ &= \sum_s P(s, \mathbf{y}_{>k-1} | s') = \sum_s P(s, \mathbf{y}_k, \mathbf{y}_{>k} | s') = \\ &= \sum_s P(\mathbf{y}_{>k} | s', s, \mathbf{y}_k) P(s, \mathbf{y}_k | s') = \sum_s P(\mathbf{y}_{>k} | s) P(s, \mathbf{y}_k | s') = \\ &= \sum_s \beta_k(s) \gamma_k(s', s)\end{aligned}$$

Again it is convenient to normalize β to its sum over all state transitions so instead of using the preceding equation we should use

$$\beta_{k-1}(s') = \frac{\sum_s \beta_k(s) \gamma_k(s', s)}{\sum_{s'} \sum_s \beta_k(s) \gamma_k(s', s)} = \frac{\beta'_{k-1}(s')}{\sum_{s'} \beta'_{k-1}(s')}$$

From the moment we know $\beta_N(s)$ we can compute the other values of β . The appropriate initial conditions in an all-zero terminated trellis are

$$\beta_N(s) = \begin{cases} 1 & s = 0 \\ 0 & s \neq 0 \end{cases}$$

Several alternatives have been proposed to alleviate the disadvantage of having to wait for the end of sequence \mathbf{y} to begin calculating β [7].

8.1.6 Iterative decoding

We have already seen in Section 6 how the BCJR algorithm and its simplifications can be applied in the iterative decoding of turbo codes. Let us see now how to reach Eq. (17), the one that expresses $L(u_k|\mathbf{y})$ as a three-element sum when we work with systematic codes.

Suppose the first coded bit, x_{k1} , is equal to the information bit u_k . Then we can expand Eq. (9), repeated here, in order to single out that systematic bit:

$$\begin{aligned}\gamma_k(s', s) &= C_k e^{u_k L(u_k)/2} \exp\left(\frac{L_c}{2} \sum_{l=1}^n x_{kl} y_{kl}\right) = \\ &= C_k e^{u_k L(u_k)/2} \exp\left[\frac{L_c}{2} \left(x_{k1} y_{k1} + \sum_{l=2}^n x_{kl} y_{kl}\right)\right] = \\ &= C_k e^{\frac{u_k}{2}[L(u_k) + L_c y_{k1}]} \exp\left(\frac{L_c}{2} \sum_{l=2}^n x_{kl} y_{kl}\right)\end{aligned}$$

Let us define the second exponential as the new variable

$$\chi_k(s', s) = \exp\left(\frac{L_c}{2} \sum_{l=2}^n x_{kl} y_{kl}\right) \quad (24)$$

This variable is just $\chi_k(s', s) = e^{L_c x_{k2} y_{k2}/2}$ if $n = 2$. We then get for $\gamma_k(s', s)$:

$$\gamma_k(s', s) = C_k e^{\frac{u_k}{2}[L(u_k) + L_c y_{k1}]} \chi_k(s', s, \mathbf{y}) . \quad (25)$$

Introducing $\chi_k(s', s)$ in Eq. (4) we will have

$$\begin{aligned}L(u_k|\mathbf{y}) &= \ln \frac{\sum_{R_1} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \gamma_k(s', s) \beta_k(s)} = \\ &= \ln \frac{\sum_{R_1} C_k e^{\frac{u_k}{2}[L(u_k) + L_c y_{k1}]} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)}{\sum_{R_0} C_k e^{\frac{u_k}{2}[L(u_k) + L_c y_{k1}]} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)}\end{aligned}$$

Sets R_1 e R_0 are associated with input bits +1 e -1, respectively, as we know. Thus, we may substitute these same values of u_k in the numerator and denominator. We will get then

$$\begin{aligned}
L(u_k | \mathbf{y}) &= \ln \left\{ e^{L(u_k)} e^{L_c y_{k1}} \frac{\sum_{R_1} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)} \right\} = \\
&= L(u_k) + L_c y_{k1} + \ln \frac{\sum_{R_1} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)}
\end{aligned}$$

If we now define

$$L_e(u_k) = \ln \frac{\sum_{R_1} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)}{\sum_{R_0} \alpha_{k-1}(s') \chi_k(s', s) \beta_k(s)} \quad (26)$$

we will finally get

$$L(u_k | \mathbf{y}) = L(u_k) + L_c y_{k1} + L_e(u_k), \quad (27)$$

or, alternately,

$$L_e(u_k) = L(u_k | \mathbf{y}) - L(u_k) - L_c y_{k1}. \quad \text{q.e.d.} \quad (28)$$

Recall that $L_e(u_k)$ is the extrinsic information a decoder conveys to the next one in iterative decoding (see Fig. 9 again), $L(u_k)$ is the a priori LLR of information bits and L_c is the channel reliability value.

We should point out that actually we do not need to calculate $\chi_k(s', s)$ to get the extrinsic information, the only quantity each decoder conveys to the other. Let us say there are two ways of calculating $L_e(u_k)$:

- We compute $\chi_k(s', s)$ by Eq. (24), $\gamma_k(s', s)$ by Eq. (25), $\alpha_{k-1}(s')$ and $\beta_k(s)$ by recursive equations (10) and (11) (or even better, by Eqs. (12) and (13)) and $L_e(u_k)$ by Eq. (26) – if necessary we compute $L(u_k | \mathbf{y})$ too (Eq. (27));
- We compute $\gamma_k(s', s)$, $\alpha_{k-1}(s')$, $\beta_k(s)$ and $L(u_k | \mathbf{y})$ just like in the BCJR algorithm (that is, using Eqs. (9), (10) (or (12)), (11) (or (13)) and (4), respectively) and, by subtraction, we compute $L_e(u_k)$ (Eq. (28)).

The iterative decoding proceeds as described in Section 6.

8.2 The log-MAP and max-log-MAP algorithms

In these algorithms additions substitute the BCJR algorithm multiplications with the aid of the jacobian logarithm

$$\ln(e^a + e^b) = \max(a, b) + \ln(1 + e^{-|a-b|}) \quad (29)$$

The jacobian logarithm is approximately equal to $\max(a, b)$. In fact, $\ln(1 + e^{-|a-b|})$ is essentially a correcting term. The difference between log-MAP and max-log-MAP algorithms lies on the way both treat

the jacobian logarithm. The former uses the exact formula of Eq. (29) and the latter uses the approximate expression $\ln(e^a + e^b) \approx \max(a, b)$. Let us unify notation through the function $\max^*(a, b)$:

$$\max^*(a, b) = \begin{cases} \ln(e^a + e^b) = \max(a, b) + \ln(1 + e^{-|a-b|}) & \text{log-MAP} \\ \max(a, b) & \text{max-log-MAP} \end{cases} \quad (30)$$

There is a degradation in performance with the max-log-MAP algorithm due to the approximation. That does not happen with the other, which performs exactly like the original BCJR algorithm. According to its proponents [5] the log-MAP correcting term $\ln(1 + e^{-x})$ can be stored in a simple eight-valued look-up table, for x between 0 and 5, because with more than eight values there is no noticeable improvement in performance.

New variables are defined:

$$\begin{aligned} A_k(s) &= \ln \alpha_k(s) \\ B_k(s) &= \ln \beta_k(s) \\ \Gamma_k(s', s) &= \ln \gamma_k(s', s) \end{aligned}$$

Applying natural logarithms to both sides of Eq. (9) we get

$$\Gamma_k(s', s) = \ln C_k + \frac{u_k L(u_k)}{2} + \frac{L_c}{2} \sum_{l=1}^n x_{kl} y_{kl}.$$

The term $\ln C_k$ is irrelevant when calculating $L(u_k|\mathbf{y})$. From the recursive equations of α and β we get

$$\begin{aligned} A_k(s) &= \ln \left[\sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s) \right] = \\ &= \ln \left\{ \sum_{s'} \exp[A_{k-1}(s') + \Gamma_k(s', s)] \right\} = \\ &= \max_{s'}^* [A_{k-1}(s') + \Gamma_k(s', s)] \end{aligned} \quad (31)$$

$$\begin{aligned} B_{k-1}(s') &= \ln \left[\sum_s \beta_k(s) \gamma_k(s', s) \right] = \\ &= \ln \left\{ \sum_s \exp[B_k(s) + \Gamma_k(s', s)] \right\} = \\ &= \max_s^* [B_k(s) + \Gamma_k(s', s)] \end{aligned} \quad (32)$$

Please note that with binary codes each summation in Eqs. (31) and (32) contains just two terms. If we analyse $A_k(s)$, for example, we see that the max-log-MAP algorithm chooses one of the two branches arriving at each state s : the one for which the sum $A_{k-1}(s') + \Gamma_k(s', s)$ is higher. That is, just like in the Viterbi algorithm there is a surviving branch and an eliminated branch. The same occurs with $B_{k-1}(s')$ but in the opposite direction.

The initial values of A and B in a terminated trellis are:

$$A_0(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases} \quad B_N(s) = \begin{cases} 0 & s = 0 \\ -\infty & s \neq 0 \end{cases}$$

$A_{k-1}(s) + \Gamma_k(s', s)$ and $B_k(s) + \Gamma_k(s', s)$ can be interpreted as a branch metric: in $A_k(s)$ case the metric is computed from the beginning to the end of the trellis, as in the Viterbi algorithm; in $B_k(s)$ case it is computed from the end to the beginning. So, in fact the max-log-MAP algorithm works as two simultaneous Viterbi algorithms, one traversing the trellis in one direction and the other traversing it the other way.

From Eq. (5) we immediately get

$$\begin{aligned} \ln P(s', s, \mathbf{y}) &= \ln[\alpha_{k-1}(s')\gamma_k(s', s)\beta_k(s)] = \\ &= A_{k-1}(s') + \Gamma_k(s', s) + B_k(s) \end{aligned}$$

The conditional LLR of Eq. (4) then becomes

$$\begin{aligned} L(u_k|\mathbf{y}) &= \ln \frac{\sum_{R_1} P(s', s, \mathbf{y})}{\sum_{R_0} P(s', s, \mathbf{y})} = \ln \frac{\sum_{R_1} \exp[A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)]}{\sum_{R_0} \exp[A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)]} = \\ &= \max_{R_1}^* [A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)] - \max_{R_0}^* [A_{k-1}(s') + \Gamma_k(s', s) + B_k(s)] \end{aligned} \quad (33)$$

Note again that the max-log-MAP algorithm uses only one of the branches of each set R_0 and R_1 .

The jacobian logarithm of Eq. (29) is a two-variable function. So, how to compute the log-likelihood ratio $L(u_k|\mathbf{y})$ of Eq. (33) with the log-MAP algorithm, where we have more than two variables? According to Robertson et al. [5] we can compute the function $\ln \left[\sum_i \exp(x_i) \right]$ recursively as follows: consider the function $z = \ln(e^{x_1} + e^{x_2} + \dots + e^{x_n})$ and proceed through the following steps:

$$\begin{aligned} 1) \quad z &= \ln \left(\frac{e^{x_1} + e^{x_2} + \dots + e^{x_n}}{e^{y_1}} \right) \Rightarrow e^{y_1} = e^{x_1} + e^{x_2} \Rightarrow y_1 = \ln(e^{x_1} + e^{x_2}) = \max(x_1, x_2) + \ln(1 + e^{-|x_1 - x_2|}) \\ 2) \quad z &= \ln \left(\frac{e^{y_1} + e^{x_3} + \dots + e^{x_n}}{e^{y_2}} \right) \Rightarrow e^{y_2} = e^{y_1} + e^{x_3} \Rightarrow y_2 = \ln(e^{y_1} + e^{x_3}) = \max(y_1, x_3) + \ln(1 + e^{-|y_1 - x_3|}) \\ &\vdots \\ n-1) \quad z &= \ln(e^{y_{n-2}} + e^{x_n}) = \max(y_{n-2}, x_n) + \ln(1 + e^{-|y_{n-2} - x_n|}) \end{aligned}$$

It would be easy to conclude from the above procedure that the function \max^* of more than two variables can be computed recursively as well [8]. For example, $\max^*(x_1, x_2, x_3) = \max^*[\max^*(x_1, x_2), x_3]$.

9 Concluding remarks and acknowledgments

This is another paper on MAP decoding. Although there is a myriad of papers on the subject too numerous to mention – a Google search should be enough to convince anyone – the author hopes his approach help make the issue a little bit easier to grasp. In fact, the concepts needed to understand the decoding algorithms are not that difficult to apprehend after all.

This work was done while on leave at the Information and Telecommunication Technology Center (ITTC) of the University of Kansas, Lawrence, USA. The author is grateful to ITTC and to Fundacao Calouste Gulbenkian and Fundacao para a Ciencia e Tecnologia (both from Lisbon, Portugal), without whose support his stay in Lawrence, KS would not have been possible.

10 References

- [1] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate", *IEEE Trans. on Information Theory*, pp. 284-287, March 1974.
- [2] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm With Soft-Decision Outputs and Its Applications", *Proceedings of GLOBECOM '89*, Dallas, Texas, pp. 47.1.1-47.1.7, November 1989.
- [3] W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data disturbed by time-varying inter-symbol interference," *Proceedings of IEEE Globecom*, pp. 1679-1684, December 1990.
- [4] J. A. Erfanian, S. Pasupathy and G. Gulak, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. Communications*, vol. 42, pp. 1661-1671, 1994.
- [5] P. Robertson, E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", *Proc. Intern. Conf. Communications (ICC)*, pp. 1009-1013, June 1995.
- [6] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes", *Proc. Intern. Conf. Communications (ICC)*, Geneva, Switzerland, pp. 1064-1070, May 1993.
- [7] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes", *The Telecommunications and Data Acquisition Progress Report 42-124*, Jet Propulsion Laboratory, Pasadena, California, pp. 63-87, February 15, 1996.
- [8] A. J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes", *IEEE Journal on Selected Areas in Communications*, Vol. 16, no. 2, pp. 260-264, February 1998.