A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain

Patrick Robertson, Emmanuelle Villebrun and Peter Hoeher

Institute for Communications Technology, German Aerospace Research Establishment (DLR)
D-82230 Oberpfaffenhofen, Germany, Tel.: ++49 8153 28 2808; Email: Patrick.Robertson@dlr.de

Abstract

For estimating the states or outputs of a Markov process, the symbol-by-symbol MAP algorithm is optimal. However, this algorithm, even in its recursive form, poses technical difficulties because of numerical representation problems, the necessity of non-linear functions and a high number of additions and multiplications. MAP like algorithms operating in the logarithmic domain presented in the past solve the numerical problem and reduce the computational complexity, but are suboptimal especially at low SNR (a common example is the Max-Log-MAP because of its use of the **max** function). A further simplification yields the soft-output Viterbi algorithm (SOVA).

In this paper, we present a Log-MAP algorithm that avoids the approximations in the Max-Log-MAP algorithm and hence is equivalent to the true MAP, but without its major disadvantages. We compare the (Log-)MAP, Max-Log-MAP and SOVA from a theoretical point of view to illuminate their commonalities and differences. As a practical example forming the basis for simulations, we consider Turbo decoding, where recursive systematic convolutional component codes are decoded with the three algorithms, and we also demonstrate the practical suitability of the Log-MAP by including quantization effects. The SOVA is, at 10^{-4}, approximately 0.7 dB inferior to the (Log-)MAP, the Max-Log-MAP lying roughly in between. We also present some complexity comparisons and conclude that the three algorithms increase in complexity in the order of their optimality.

1 Introduction

We will consider trellis-based soft-output decoding algorithms delivering additional reliability information together with hard decisions. The *Bahl-Jelinek algorithm*, also known as the *symbol-by-symbol MAP algorithm* (MAP algorithm for short), is optimal for estimating the states or outputs of a Markov process observed in white noise [1]. However, this algorithm is perhaps too difficult in practice, basically because of the numerical representation of probabilities, non-linear functions and because of mixed multiplications and additions of these values.

Some approximations of the MAP algorithm have been derived, such as the *soft-output Viterbi algorithm (SOVA)* [2] and the *Max-Log-MAP algorithm* [3, 4, 5]. In both algorithms, processing is exclusively in the logarithmic domain; values and operations (addition and **max**-function) are easier to handle. However, both algorithms are suboptimal at low signal-to-noise ratios, where we use Turbo-Codes [6, 7, 8, 9], for example.

In this paper, we will modify the Max-Log-MAP algorithm through the use of a simple correction function at each **max**-operation [9]. This algorithm, to be called *Log-MAP algorithm*, is equivalent to the MAP algorithm in terms of performance, but without its problems of implementation. The correction needs just an additional one-dimensional table look-up and an addition per **max**-operation.

The organization of the paper is as follows: After reviewing the MAP and Max-Log-MAP algorithms, we will derive the Log-MAP algorithm in Section 2. In Section 3, we will compare these algorithms with the VA and the SOVA. Complexity comparisons and quantization issues are covered in Section 4. Finally, numerical results are presented in Section 5 by applying the addressed algorithms in a Turbo-Code system.

2 Definition of the MAP, Max-Log-MAP, and Log-MAP Algorithms

2.1 The Encoder and Notation

Since we will study the behaviour of the (Max-)Log-MAP algorithm applied to the decoding of convolutional codes (and in particular recursive systematic convolutional (RSC) codes), we will choose a notation that complies with such an encoder, for example one with four memory elements as shown in Fig. 1. Since the MAP algorithm is essentially block-oriented, we shall represent the binary input (information) data sequence by $\vec{d} = (d_1, ..., d_N)$.

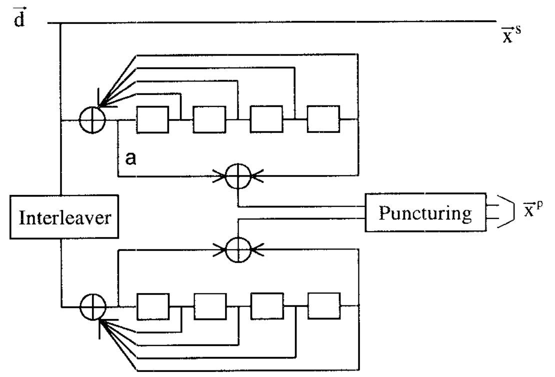


Figure 1: Two identical recursive systematic convolutional encoders employed in a Turbo coding scheme.

The encoder has M memory elements. In our example (the RSC code is of rate 1/2) it has two outputs, one is the sequence of information bits $\vec{x}^s = (x_1^s, ..., x_N^s) = \vec{d}$, since the encoder is systematic. The other is the 'parity information' sequence $\vec{x}^p = (x_1^p, ..., x_N^p)$, with $x_k^p = \sum_{i=0}^{M} g_i^f a_{k-i}$, where $(g_0^f, \ldots, g_M^f)$ is the feed-forward generator and $a_k = d_k \oplus \sum_{i=1}^{M} g_i^b a_{k-i}$; similarly, $(g_1^b, \ldots, g_M^b)$ is the feed-back generator of the encoder.

The sequences $\vec{x}^s$ and $\vec{x}^p$ may be punctured, are then modulated and transmitted over a channel. In this work, we have assumed BPSK modulation and an AWGN channel with one-sided noise power spectral density N_0. The corresponding received sequences be $\vec{y}^s$ and $\vec{y}^p$. For brevity, the sequence $\vec{y} = (\vec{y}^s, \vec{y}^p)$ will refer to the sequence of pairs of received systematic and parity symbols.

2.2 The MAP Algorithm

We will not repeat the derivation of the MAP algorithm, but only state the results. For more detail see [1, 6, 8]. Let the state of the encoder at time k be S_k, it can take on values between 0 and $2^M - 1$. The bit d_k is associated with the transition from step $k-1$ to step k. Goal of the MAP algorithm is to provide us with the logarithm of the ratio of the a posteriori probability (APP) of each information bit d_k being 1 to the APP of it being 0. We obtain:

$$\Lambda(d_k) = \ln \frac{\sum_{S_k}\sum_{S_{k-1}} \gamma_1(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1}) \cdot \beta_k(S_k)}{\sum_{S_k}\sum_{S_{k-1}} \gamma_0(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1}) \cdot \beta_k(S_k)}, \tag{1}$$

where the forward recursion of the MAP can be expressed as:

$$\begin{aligned} \alpha_k(S_k) &= \frac{\sum_{S_{k-1}}\sum_{i=0}^{1} \gamma_i(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1})}{\sum_{S_k}\sum_{S_{k-1}}\sum_{i=0}^{1} \gamma_i(y_k, S_{k-1}, S_k) \cdot \alpha_{k-1}(S_{k-1})}; \\ \alpha_0(S_0) &= \begin{cases} 1 & \text{for } S_0 = 0 \\ 0 & \text{else} \end{cases} \end{aligned} \tag{2}$$

and the backward recursion as:

$$\begin{aligned} \beta_k(S_k) &= \frac{\sum_{S_{k+1}}\sum_{i=0}^{1} \gamma_i(y_{k+1}, S_k, S_{k+1}) \cdot \beta_{k+1}(S_{k+1})}{\sum_{S_k}\sum_{S_{k+1}}\sum_{i=0}^{1} \gamma_i(y_{k+1}, S_k, S_{k+1}) \cdot \alpha_k(S_k)}; \\ \beta_N(S_N) &= \begin{cases} 1 & \text{for } S_N = 0 \\ 0 & \text{else} \end{cases} \quad . \end{aligned} \tag{3}$$

The branch transition probabilities are given by

$$\begin{aligned} \gamma_i((y_k^s, y_k^p), S_{k-1}, S_k) = q\,(d_k = i|S_k, S_{k-1}) \cdot \\ p(y_k^s|d_k = i)\; p(y_k^p|d_k = i, S_k, S_{k-1})\; Pr\{S_k|S_{k-1}\}. \end{aligned} \tag{4}$$

The value of $q\,(d_k = i|S_k, S_{k-1})$ is either one or zero depending on whether bit i is associated with the transition from state S_{k-1} to S_k or not. It is in the last component that we use a priori information for bit d_k [10]: In our case of no parallel transitions, $Pr\{S_k|S_{k-1}\} = Pr\{d_k = 1\}$ if $q\,(d_k = 1|S_k, S_{k-1}) = 1$; and $Pr\{S_k|S_{k-1}\} = Pr\{d_k = 0\}$ if $q\,(d_k = 0|S_k, S_{k-1}) = 1$.

2.3 The Max-Log-MAP Algorithm

As we already said in the introduction, the MAP algorithm is likely to be considered too complex for implementation in a real system [5]. To avoid the number of complicated operations and also number representation problems, one does not calculate $\gamma_i((y_k^s, y_k^p), S_{k-1}, S_k)$, $\alpha_k(S_k)$ and $\beta_k(S_k)$ any longer. One computes and works with the logarithms of these values instead [3]-[5].

By taking the logarithm of $\gamma_i((y_k^s, y_k^p), S_{k-1}, S_k)$ derived in (4) and inserting

$$p(y_k^p|d_k = i, S_k, S_{k-1}) = \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{1}{N_0}(y_k^p - x_k^p(i, S_k, S_{k-1}))^2}, \tag{5}$$

we obtain the following expression for $q(.) = 1$:

$$\begin{aligned} \ln \gamma_i((y_k^s, y_k^p), S_{k-1}, S_k) = \frac{2y_k^s x_k^s(i)}{N_0} + \\ \frac{2y_k^p x_k^p(i, S_k, S_{k-1})}{N_0} + \ln Pr\{S_k|S_{k-1}\} + K. \end{aligned} \tag{6}$$

We can ignore the constant K, indeed it cancels out in the calculation of $\ln \alpha_k(S_k)$ and $\ln \beta_k(S_k)$. We must remember that N_0 must be estimated to correctly weight the channel information with the a priori probability $Pr\{S_k|S_{k-1}\}$.

For $\ln \alpha_k(S_k)$, we get:

$$\begin{aligned} \ln \alpha_k(S_k) = \ln\Big(\sum_{S_{k-1}}\sum_{i=0}^{1} e^{\ln \gamma_i((y_k^s, y_k^p), S_{k-1}, S_k) + \ln \alpha_{k-1}(S_{k-1})}\Big) \\ - \ln\Big(\sum_{S_k}\sum_{S_{k-1}}\sum_{i=0}^{1} e^{\ln \gamma_i((y_k^s, y_k^p), S_{k-1}, S_k) + \ln \alpha_{k-1}(S_{k-1})}\Big). \end{aligned} \tag{7}$$

To obtain a simple solution, we use the following approximation:

$$\ln(e^{\delta_1} + ... + e^{\delta_n}) \approx \max_{i \in \{1..n\}} \delta_i; \tag{8}$$

$\max_{i \in \{1..n\}} \delta_i$ can be calculated by successively using $n-1$ maximum functions over only two values. From now on, we define that $\bar{\gamma}_i((y_k^s, y_k^p), S_{k-1}, S_k) = \ln \gamma_i((y_k^s, y_k^p), S_{k-1}, S_k)$, $\bar{\alpha}_k(S_k) = \ln \alpha_k(S_k)$ and $\bar{\beta}_k(S_k) = \ln \beta_k(S_k)$. We eventually obtain:

$$\begin{aligned} \bar{\alpha}_k(S_k) \approx \max_{(S_{k-1}, i)} (\bar{\gamma}_i((y_k^s, y_k^p), S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1})) \\ - \max_{(S_k, S_{k-1}, i)} (\bar{\gamma}_i((y_k^s, y_k^p), S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1})); \end{aligned} \tag{9}$$

and similarly,

$$\begin{aligned} \bar{\beta}_k(S_k) \approx \max_{(S_{k+1}, i)} (\bar{\gamma}_i((y_{k+1}^s, y_{k+1}^p), S_k, S_{k+1}) + \bar{\beta}_{k+1}(S_{k+1})) \\ - \max_{(S_k, S_{k+1}, i)} (\bar{\gamma}_i((y_{k+1}^s, y_{k+1}^p), S_k, S_{k+1}) + \bar{\alpha}_k(S_k)). \end{aligned} \tag{10}$$

The second terms are a consequence of the derivation from (2) and (3); they are needed for numerical reasons. Omitting them has no effect on the value of the output of the Max-Log-MAP algorithm, since these normalization terms will cancel out in (11).

In the same way we can give an approximation of the log-likelihood reliability of each bit d_k:

$$\Lambda(d_k) \approx \max_{(S_k,S_{k-1})}(\bar{\gamma}_1(y_k,S_{k-1},S_k)+\bar{\alpha}_{k-1}(S_{k-1})+\bar{\beta}_k(S_k)) - \max_{(S_k,S_{k-1})}(\bar{\gamma}_0(y_k,S_{k-1},S_k)+\bar{\alpha}_{k-1}(S_{k-1})+\bar{\beta}_k(S_k)). \quad (11)$$

To be used in a Turbo decoder, we want the output of the Max-Log-MAP algorithm, $\Lambda(d_k)$, to be split into three terms (extrinsic, a priori and systematic components) as shown in [6, 7, 8]; it can easily be shown that this is possible here.

The extrinsic component will be used as a priori information in the next decoding stage. This a priori log-likelihood ratio (LLR) for bit d_k is called $L(d_k)$ for short. We need to determine the a priori information in (6). If $q(d_k = 1|S_k, S_{k-1}) = 1$, then

$$L(d_k) = \ln\left(\frac{Pr\{d_k=1\}}{Pr\{d_k=0\}}\right) = \ln\left(\frac{Pr\{S_k|S_{k-1}\}}{1-Pr\{S_k|S_{k-1}\}}\right), \quad (12)$$

hence $\ln Pr\{S_k|S_{k-1}\} = L(d_k) - \ln(1+e^{L(d_k)})$. An approximation for $Pr\{S_k|S_{k-1}\}$ can be easily found using (8):

$$\ln Pr\{S_k|S_{k-1}\} \approx L(d_k) - \max(0, L(d_k)). \quad (13)$$

If $q(d_k = 0|S_k, S_{k-1}) = 1$ then

$$L(d_k) = \ln\left(\frac{Pr\{d_k=1\}}{Pr\{d_k=0\}}\right) = \ln\left(\frac{1-Pr\{S_k|S_{k-1}\}}{Pr\{S_k|S_{k-1}\}}\right), \quad (14)$$

hence $\ln Pr\{S_k|S_{k-1}\} = -\ln(1+e^{L(d_k)})$. Similarly we can approximate $\ln Pr\{S_k|S_{k-1}\} \approx -\max(0, L(d_k))$.

2.4 Correction of the Approximation: The Log-MAP Algorithm

Because of the approximation (8) we applied, the Max-Log-MAP algorithm is suboptimal and yields an inferior soft-output than the MAP algorithm. The problem is to exactly calculate $\ln(\exp\delta_1 + ... + \exp\delta_n)$. This problem can be solved by using the Jacobian logarithm [3, 4]:

$$\begin{aligned}\ln(e^{\delta_1}+e^{\delta_2}) &= \max(\delta_1,\delta_2)+\ln(1+e^{-|\delta_2-\delta_1|}) \\ &= \max(\delta_1,\delta_2)+f_c(|\delta_1-\delta_2|), \quad (15)\end{aligned}$$

where $f_c(.)$ is a correction function. Let us now prove recursively that the expression $\ln(\exp\delta_1 + ... + \exp\delta_n)$ can be computed exactly. The recursion is initialized with (15). Suppose that $\delta = \ln(e^{\delta_1} + ... + e^{\delta_{n-1}})$ is known. Hence,

$$\begin{aligned}&\ln(e^{\delta_1}+...+e^{\delta_n}) \\ &= \ln(\Delta + e^{\delta_n}) \quad \text{with } \Delta = e^{\delta_1}+...+e^{\delta_{n-1}} = e^{\delta} \\ &= \max(\ln\Delta, \delta_n) + f_c(|\ln\Delta - \delta_n|) \\ &= \max(\delta,\delta_n) + f_c(|\delta-\delta_n|) \quad \text{q.e.d.} \quad (16)\end{aligned}$$

When deriving the Log-MAP algorithm, we now augment all maximizations over two values with the correction function. As a consequence, by correcting at each step the approximation made by the Max-Log-MAP, we have preserved the original MAP algorithm.

By calculating $f_c(.)$, we lose some of the lower complexity of the Max-Log-MAP algorithm. That is why we approximate $f_c(.)$ by a pre-computed table. Since the correction only depends on $|\delta_2 - \delta_1|$, this table is one dimensional. We shall see that only very few values need to be stored.

3 Comparison of the (Max-)Log-MAP and Soft-Output Viterbi Algorithms

3.1 Hard Decisions

In [5], it was claimed that the hard decision of the Max-Log-MAP provides exactly the same hard decision as the Viterbi algorithm. We now present a mathematical proof of this result for our example of rate 1/2 RSC codes; extensions are rudimentary. We assume for simplicity that the definitions of $\bar{\alpha}_k(S_k)$ and $\bar{\beta}_k(S_k)$ do not include the normalization term. Remember that the Viterbi algorithm selects that path with the largest metric:

$$M = \max_{\forall \text{ paths}}\left\{\sum_{k=1}^{N}\left[-\frac{1}{N_0}(y_k^s - x_k^s(\text{path}))^2 - \frac{1}{N_0}(y_k^p - x_k^p(\text{path}))^2 + \ln(Pr\{S_k|S_{k-1}\})\right]\right\}. \quad (17)$$

The Max-Log-MAP output (11) for the last bit d_N can be written as $\Lambda(d_N) = M_1(d_N) - M_0(d_N)$, where

$$M_i(d_N) = \max_{S_{N-1}}\{\bar{\gamma}_i(y_N,S_{N-1},S_N=0)+\bar{\alpha}_{N-1}(S_{N-1})\}. \quad (18)$$

By applying the recursive definition of $\bar{\alpha}_{N-1}(S_{N-1})$ (without normalization) we obtain:

$$M_i(d_N) = \max_{S_{N-1}}\{\bar{\gamma}_i(y_N,S_{N-1},S_N=0) + \max_{(S_{N-2},j_{N-1})}(\bar{\gamma}_{j_{N-1}}(y_{N-1},S_{N-2},S_{N-1})+\bar{\alpha}_{N-2}(S_{N-2}))\}. \quad (19)$$

There exists an $S_{N-2,max}$ and an $j_{N-1,max}$ such that

$$M_i(d_N) = \max_{S_{N-1}}\{\bar{\gamma}_i(y_N,S_{N-1},S_N=0) + \bar{\gamma}_{j_{N-1,max}}(y_{N-1},S_{N-2,max},S_{N-1})+\bar{\alpha}_{N-2}(S_{N-2,max})\}. \quad (20)$$

This is repeated $N-2$ times, yielding:

$$M_i(d_N) = \max_{S_{N-1}}\{\bar{\gamma}_i(y_N,S_{N-1},S_N=0)\} + \sum_{k=1}^{N-1}\bar{\gamma}_{j_{k,max}}(y_N,S_{k-1,max},S_{k,max}), \quad (21)$$

since we assume $\bar{\alpha}_0(0) = 0$. By making a decision for the bit d_N, the Max-Log-MAP algorithm selects the largest $M_i(d_N)$. By inserting (6) into the maximum over i of (21) and comparing with (17), we can easily see that the Max-Log-MAP and Viterbi algorithm make the same decision for bit d_N, since $\max_{i\in\{0,1\}} M_i(d_N) = M+C = M(d_N)$, where C is a constant. To continue, let us define a metric for the Max-Log-MAP as:

$$M(d_k) = \max_{(S_k,S_{k-1},i)}(\bar{\gamma}_i(y_k,S_{k-1},S_k)+\bar{\alpha}_{k-1}(S_{k-1})+\bar{\beta}_k(S_k)). \quad (22)$$

The Max-Log-MAP will choose that bit d_k that maximizes (22). We now suppose that $M(d_k) = M + C$. Let us first prove that $M(d_k) = M(d_{k-1})$:

$$M(d_{k-1}) = \max_{S_{k-1}}\left\{\max_{(S_{k-2},i)}\{\bar{\gamma}_i(y_{k-1},S_{k-2},S_{k-1}) + \bar{\alpha}_{k-2}(S_{k-2})\} + \bar{\beta}_{k-1}(S_{k-1})\right\}. \quad (23)$$

We apply the recursive definitions of $\bar{\alpha}_{k-1}(S_{k-1})$ and $\bar{\beta}_{k-1}(S_{k-1})$, so that we obtain:

$$M(d_{k-1}) = \max_{(S_k, S_{k-1}, i)} \Big\{ \bar{\gamma}_i(y_k, S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k) \Big\} = M(d_k). \quad (24)$$

We can deduce from this recursion step and from the initial step where $k = N$ that $\forall k \in \{1, \dots, N\}$

$$M_{\text{Max-Log-MAP}} = M(d_N) = M(d_k) = M + C. \quad (25)$$

For each bit d_k, the Max-Log-MAP algorithm calculates two Viterbi metrics and takes the largest one. This proves that the Max-Log-MAP algorithm makes the same hard decisions as the Viterbi algorithm.

3.2 Soft Outputs

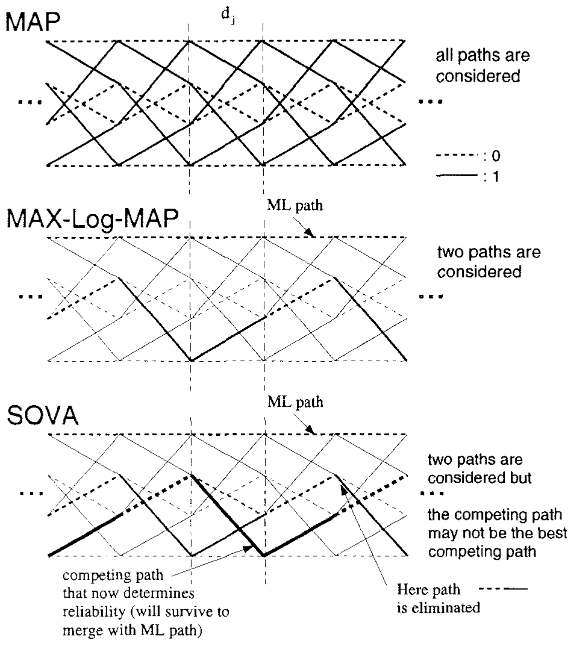
As we have already explained, the Max-Log-MAP algorithm and the SOVA work with the same metric. If we consider only the hard decisions, they are identical. But, they behave in different ways in computing the information returned about the reliability of decoded bit d_k. The SOVA considers only one competing path per decoding step. That is to say, for each bit d_j it does not consider all the competing paths but only the survivors of the Viterbi algorithm. To be taken into account in the reliability estimation, a competing path must join the path chosen by the Viterbi algorithm without being eliminated.

The differences between the (Log)MAP, Max-Log-MAP and SOVA are illustrated in Fig. 2. The MAP takes all paths into its calculation, but splits them into two sets: those that have an information bit one at step j and those that have a zero; it returns the LLR of these two sets. All that changes from step to step, is the classification of the paths into the respective sets. Due to the Markov properties of the trellis, the computation can be done relatively easily. In contrast, the Max-Log-MAP looks at only two paths *per step*: the best with bit zero and the best with bit one at transition j; it then outputs the difference of the log-likelihoods. However, from step to step one of these paths can change, but one will always be the maximum-likelihood (ML) path. The SOVA will always correctly find one of these two paths (the ML path), but not necessarily the other, since it may have been eliminated before merging with the ML path. There is no bias on the SOVA output when compared to that of the Max-Log-MAP algorithm, only the former will be more noisy.

4 Complexity and Quantization

4.1 Complexity Comparisons

As mentioned earlier, the correction function in (15) used by the Log-MAP can be implemented using a look-up table. We found that excellent results can be obtained with 8 stored values and $|\delta_1 - \delta_2|$ ranging between 0 and 5. No improvement is achieved when using a finer representation. We now present the result of complexity analyses in the following table, taking into account the additional complexity of including a priori information:



MAP
d_j
all paths are considered
...
- - - - : 0
——— : 1
MAX-Log-MAP
ML path
two paths are considered
SOVA
ML path
two paths are considered but
the competing path may not be the best competing path
competing path that now determines reliability (will survive to merge with ML path)
Here path - - - - is eliminated

Figure 2: Comparison between (Log)MAP, Max-Log-MAP and SOVA. The MAP uses all paths in the trellis to optimally determine the reliability of bit d_j. The Max-Log-MAP makes its decision (and soft output) based on the best two paths with different d_j. The SOVA also takes two paths, but not necessarily both the same as for the Max-Log-MAP.

Operation	Max-Log-MAP	Log-MAP	SOVA
`max` ops	$5 \times 2^M - 2$	$5 \times 2^M - 2$	$3(M+1) + 2^M$
additions	$10 \times 2^M + 11$	$15 \times 2^M + 9$	$2 \times 2^M + 8$
mult. by ± 1	8	8	8
bit comps			$6(M+1)$
look-ups		$5 \times 2^M - 2$	

If we assume that one bit comparison costs as much as one addition, this table allows us to conclude that the Max-Log-MAP algorithm is more than twice as complex as the SOVA, for memory $M = 4$ and less than 2 times as complex for $M = 2$.

4.2 Quantization

We shall now present the quantization ranges that were used in simulations of the Log-MAP, they are based on observations of the distributions of the pertinent variables. We have attempted to take into account the large variations that are the result of the iterative decoding process in a Turbo decoder.

Variable	Number of levels	Limit values	
y^s, y^p	16	-1.83	2.16
$L(d_k)$	64	-15.5	16.0
$\bar{\gamma}_i(y_k, S_k, S_{k-1})$	256	-15.875	16.0
$\bar{\alpha}_k(S_k)$	256	-15.875	16.0
$\bar{\beta}_k(S_k)$	256	-127.0	128.0
$\Lambda(d_k)$	256	-31.75	32.0

5 Simulation Results for Application in a Turbo-Code System

Figure 3 shows the BER for decoding Turbo-Codes with the MAP, Log-MAP, Max-Log-MAP, and SOVA, respectively, taking no quantization effects into account. However, the Log-MAP is applied with an 8-values correction table; a loss due to the 3 bit quantization of the correction table is not visible. Results for the SOVA are taken from [7].

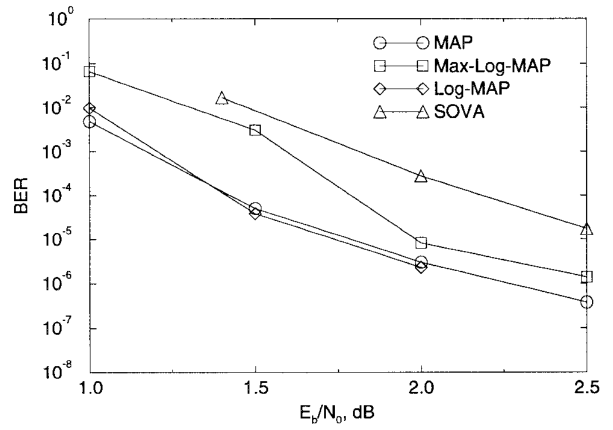


Figure 3: BER for Turbo-decoding with MAP, Log-MAP, Max-Log-MAP and SOVA (8 iterations). $N = 1024$, $M = 4$.

Finally, Figure 4 shows the corresponding BER for the Log-MAP with and without quantization. We set N_0 to 2, and used an 8 values table. The step-size of the correction table is equal to 1/8.

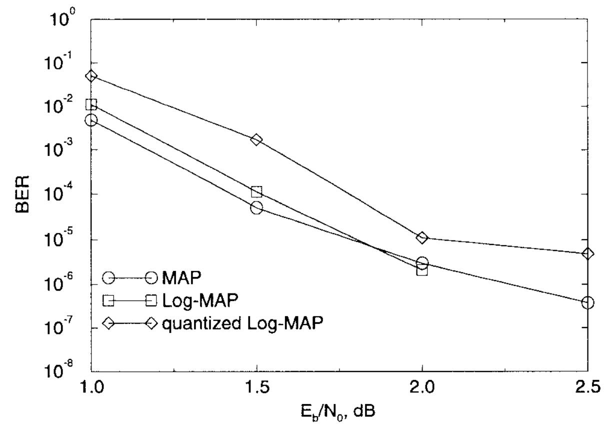


Figure 4: BER for Turbo-decoding with MAP, Log-MAP, and quantized Log-MAP (8 iterations).

6 Conclusions

We have demonstrated a Log-MAP algorithm that is equivalent to the (true) symbol-by-symbol MAP algorithm, i.e., is optimum for estimating the states or outputs of a Markov process. However, the novel implementation works exclusively in the logarithmic domain, thus avoiding the basic problems of the symbol-by-symbol MAP algorithm. The difference between our Log-MAP algorithm and the known Max-Log-MAP algorithm, which just approximates the MAP algorithm, is the substitution of logarithms by the Jacobian logarithm. This correction is simple to implement and quite insensitive to a quantization; a loss due to quantization of the correction function is not visible. Further implementation issues concerning continuous data ("real-time implementation") and simplifications valid for feed-forward trellises [5] are still applicable. We have compared the MAP, (Max-)Log-MAP and SOVA from a theoretical point of view to illuminate their commonalities and differences. As a practical example forming the basis for simulations, we considered Turbo-decoding, where recursive systematic convolutional (RSC) component codes have been decoded with the three algorithms. Quantization of the whole Log-MAP algorithm was also investigated: the loss is about 0.5 dB. This can probably be improved, but Turbo-Codes have a huge variation of variables' ranges as a result of iterative decoding. Finally, we have compared the complexity of the three algorithms. The number of operations of the Log-MAP is about twice the number of operations of the SOVA, however, the former is more suited to parallel processing [3].

We conclude that the Log-MAP is particularly suitable for decoding Turbo-Codes, since in this challenging application we have a very low signal-to-noise ratio but a small number of states, so that the additional complexity is less pronounced.

Acknowledgements

The authors would especially like to thank Dr. J. Hagenauer and Dr. J. Huber for valuable discussions and comments.

References

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, March 1974.

[2] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. GLOBECOM '89*, pp. 1680–1686, November 1989.

[3] J. A. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for isi channels," *IEEE Trans. Commun.*, vol. 42, pp. 1661–1671, February/March/April 1994.

[4] W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data disturbed by time-varying intersymbol interference," in *Proc. GLOBECOM '90*, pp. 1679–1684, December 1990.

[5] J. Petersen, "Implementierungsaspekte zur Symbol-by-Symbol MAP Decodierung von Faltungscodes," in *Proc. ITG Tagung, Codierung für Quelle, Kanal und Übertragung*, pp. 41–48, October 1994.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. ICC '93*, pp. 1064–1070, May 1993.

[7] J. Hagenauer, P. Robertson, and L. Papke, "Iterative ("Turbo") decoding of systematic convolutional codes with the MAP and SOVA algorithms," in *Proc. ITG Tagung, Codierung für Quelle, Kanal und Übertragung*, pp. 21–29, October 1994.

[8] P. Robertson, "Illuminating the structure of code and decoder for parallel concatenated recursive systematic (turbo) codes," in *Proc. GLOBECOM '94*, pp. 1298–1303, December 1994.

[9] E. Villebrun, "Turbo-decoding with close-to-optimal MAP algorithms." Diploma thesis, TU Munich, September 1994.

[10] G. D. Forney, "The Viterbi algorithm," *Proc. of the IEEE*, vol. 61, pp. 268–278, March 1973.