Soft-Output Decoding Algorithms in Iterative Decoding of Turbo Codes

S. Benedetto and G. Montorsi
Politecnico di Torino, Torino, Italy

D. Divsalar and F. Pollara
Communications Systems and Research Section

In this article, we present two versions of a simplified maximum a posteriori decoding algorithm. The algorithms work in a sliding window form, like the Viterbi algorithm, and can thus be used to decode continuously transmitted sequences obtained by parallel concatenated codes, without requiring code trellis termination. A heuristic explanation is also given of how to embed the maximum a posteriori algorithms into the iterative decoding of parallel concatenated codes (turbo codes). The performances of the two algorithms are compared on the basis of a powerful rate 1/3 parallel concatenated code. Basic circuits to implement the simplified a posteriori decoding algorithm using lookup tables, and two further approximations (linear and threshold), with a very small penalty, to eliminate the need for lookup tables are proposed.

I. Introduction and Motivations

The broad framework of this analysis encompasses digital transmission systems where the received signal is a sequence of wave forms whose correlation extends well beyond T, the signaling period. There can be many reasons for this correlation, such as coding, intersymbol interference, or correlated fading. It is well known [1] that the optimum receiver in such situations cannot perform its decisions on a symbol-by-symbol basis, so that deciding on a particular information symbol u_k involves processing a portion of the received signal T_d seconds long, with $T_d > T$. The decision rule can be either optimum with respect to a sequence of symbols, $u_k^n \triangleq (u_k, u_{k+1}, \cdots, u_{k+n-1})$, or with respect to the individual symbol, u_k.

The most widely applied algorithm for the first kind of decision rule is the Viterbi algorithm. In its optimum formulation, it would require waiting for decisions until the whole sequence has been received. In practical implementations, this drawback is overcome by anticipating decisions (single or in batches) on a regular basis with a fixed delay, D. A choice of D five to six times the memory of the received data is widely recognized as a good compromise between performance, complexity, and decision delay.

Optimum symbol decision algorithms must base their decisions on the maximum a posteriori (MAP) probability. They have been known since the early seventies [2,3], although much less popular than the Viterbi algorithm and almost never applied in practical systems. There is a very good reason for this neglect in that they yield performance in terms of symbol error probability only slightly superior to the Viterbi algorithm, yet they present a much higher complexity. Only recently, the interest in these

algorithms has seen a revival in connection with the problem of decoding concatenated coding schemes. Concatenated coding schemes (a class in which we include product codes, multilevel codes, generalized concatenated codes, and serial and parallel concatenated codes) were first proposed by Forney [4] as a means of achieving large coding gains by combining two or more relatively simple "constituent" codes. The resulting concatenated coding scheme is a powerful code endowed with a structure that permits an easy decoding, like "stage decoding" [5] or "iterated stage decoding" [6].

To work properly, all these decoding algorithms cannot limit themselves to passing the symbols decoded by the inner decoder to the outer decoder. They need to exchange some kind of soft information. Actually, as proved by Forney [4], the optimum output of the inner decoder should be in the form of the sequence of the probability distributions over the inner code alphabet conditioned on the received signal, the a posteriori probability (APP) distribution. There have been several attempts to achieve, or at least to approach, this goal. Some of them are based on modifications of the Viterbi algorithm so as to obtain, at the decoder output, in addition to the "hard"-decoded symbols, some reliability information. This has led to the concept of "augmented-output," or the list-decoding Viterbi algorithm [7], and to the soft-output Viterbi algorithm (SOVA) [8]. These solutions are clearly suboptimal, as they are unable to supply the required APP. A different approach consisted in revisiting the original symbol MAP decoding algorithms [2,3] with the aim of simplifying them to a form suitable for implementation [9–12].

In this article, we are interested in soft-decoding algorithms as the main building block of iterative stage decoding of parallel concatenated codes. This has become a "hot" topic for research after the successful proposal of the so-called turbo codes [6]. They are (see Fig. 1) parallel concatenated convolutional codes (PCCC) whose encoder is formed by two (or more) constituent systematic encoders joined through an interleaver. The input information bits feed the first encoder and, after having been interleaved by the interleaver, enter the second encoder. The codeword of the parallel concatenated code consists of the input bits to the first encoder followed by the parity check bits of both encoders. Generalizations to more than one interleaver are possible and fruitful [13].

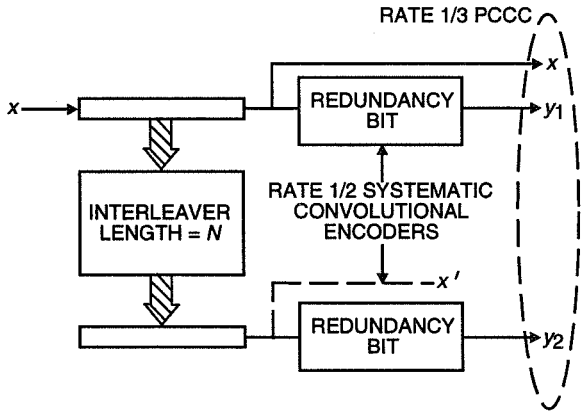

Fig. 1. Parallel concatenated convolutional code.

The suboptimal iterative decoder is modular and consists of a number of equal component blocks formed by concatenating soft decoders of the constituent codes (CC) separated by the interleavers used at the encoder side. By increasing the number of decoding modules and, thus, the number of decoding iterations, bit-error probabilities as low as 10^{-5} at $E_b/N_0 = 0.0$ dB for rate 1/4 PCCC have been shown by simulation [13]. A version of turbo codes employing two eight-state convolutional codes as constituent codes, an interleaver of 32×32 bits, and an iterative decoder performing two and one-half iterations with a complexity of the order of five times the maximum-likelihood (ML) Viterbi decoding of each constituent code is presently available on a chip yielding a measured bit-error probability of 0.9×10^{-6} at $E_b/N_0 = 3$ dB [14].

In recent articles [15,17], upper bounds to the ML bit-error probability of PCCCs have been proposed. As a by-product, it has been shown by simulation that iterative decoding can approach quite closely the ML performance. The iterative decoding algorithm was a simplification of the algorithm proposed in [3], whose regular steps and limited complexity seem quite suitable to very large-scale integration (VLSI) implementation. Simplified versions of the algorithm [3] have been proposed and analyzed in [12] in the context of a block decoding strategy that requires trellis termination after each block of bits. Similar simplification also was used in [16] for hardware implememtation of the MAP algorithm.

In this article, we will describe two versions of a simplified MAP decoding algorithm that can be used as building blocks of the iterative decoder to decode PCCCs. A distinctive feature of the algorithms is that they work in a "sliding window" form, like the Viterbi algorithm, and thus can be used to decode "continuously transmitted" PCCCs, without requiring trellis termination and a block-equivalent structure of the code. The simplest among the two algorithms will be compared with the optimum block-decoding algorithm proposed in [3]. The comparison will be given in terms of bit-error probability when the algorithms are embedded into iterative decoding schemes for PCCCs. We will choose, for comparison, a very powerful PCCC scheme suitable for deep-space applications [18–20] and, thus, working at a very low signal-to-noise ratio.

II. System Context and Notations

As previously outlined, our final aim is to find suitable soft-output decoding algorithms for iterated staged decoding of parallel concatenated codes employed in a continuous transmission. The core of such algorithms is a procedure to derive the sequence of probability distributions over the information symbols' alphabet based on the received signal and constrained on the code structure. Thus, we will start by this procedure and only later will we extend the description to the more general setting.

Readers acquainted with the literature on soft-output decoding algorithms know that one burden in understanding and comparing the different algorithms is the spread and, sometimes, mess of notations involved. For this reason, we will carefully define the system and notations and then stick consistently to them for the description of all algorithms.

For the first part of the article, we will refer to the system of Fig. 2. The information sequence $\mathbf{u}$, composed of symbols drawn from an alphabet $U = \{u_1, \cdots, u_I\}$ and emitted by the source, enter an encoder that generates code sequences $\mathbf{c}$. Both source and code sequences are defined over a time index set K (a finite or infinite set of integers). Denoting the code alphabet $C = \{c_1, \cdots, c_M\}$, the code $\mathcal{C}$ can be written as a subset of the Cartesian product of C by itself K times, i.e.,

$$\mathcal{C} \subseteq C^K$$

The code symbols c_k (the index k will always refer to time throughout the article) enter the modulator, which performs a one-to-one mapping of them with its signals, or channel input symbols x_k, belonging to the set $X = \{x_1, \cdots, x_M\}$.[1]

The channel symbols x_k are transmitted over a stationary memoryless channel with output symbols y_k. The channel is characterized by the transitions probability distribution (discrete or continuous, according to the channel model) $P(y|x)$. The channel output sequence is fed to the symbol-by-symbol soft-output demodulator, which produces a sequence of probability distributions $\gamma_k(c)$ over C conditioned on the received signal, according to the memoryless transformation

[1] For simplicity of notation, we have assumed that the cardinality of the modulator equals that of the code alphabet. In general, each coded symbol can be mapped in more than one channel symbol, as in the case of multilevel codes or trellis codes with parallel transitions. The extension is straightforward.

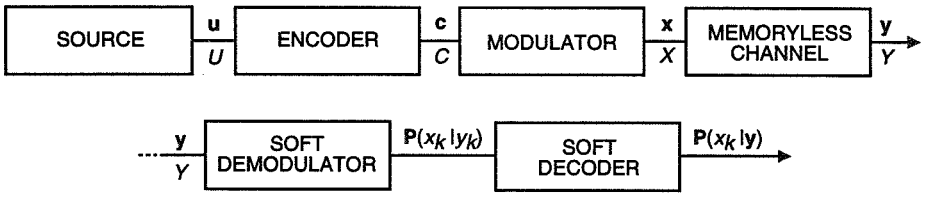

Fig. 2. The transmission system.

$$\gamma_k(c) \triangleq P(x_k = x(c), y_k) = P(y_k|x_k = x(c))P_k(c) \triangleq \gamma_k(x) \tag{1}$$

where we have assumed to know the sequence of the a priori probability distributions of the channel input symbols $(P_k(x) : k \in K)$ and made use of the one-to-one mapping $C \rightarrow X$.

The sequence of probability distributions $\gamma_k(c)$ obtained by the modulator on a symbol-by-symbol basis is then supplied to the soft-output symbol decoder, which processes the distributions in order to obtain the probability distributions $P_k(u|\mathbf{y})$. They are defined as

$$P_k(u|\mathbf{y}) \triangleq P(u_k = u|\mathbf{y}) \tag{2}$$

The probability distributions $P_k(u|\mathbf{y})$ are referred to in the literature as symbol-by-symbol a posteriori probabilities (APP) and represent the optimum symbol-by-symbol soft output.

From here on, we will limit ourselves to the case of time-invariant convolutional codes with N states, use the following notations with reference to Fig. 3, and assume that the (integer) time instant we are interested in is the kth:

(1) S_i is the generic state at time k, belonging to the set $\mathcal{S} = \{S_1, \cdots, S_N\}$

(2) $S_i^-(u')$ is one of the precursors of S_i, and precisely the one defined by the information symbol u' emitted during the transition $S_i^-(u') \rightarrow S_i$.[2]

(3) $S_i^+(u)$ is one of the successors of S_i, and precisely the one defined by the information symbol u emitted during the transition $S_i \rightarrow S_i^+(u)$.

(4) To each transition in the trellis, a signal x is associated, which depends on the state from which the transition originates and on the information symbol u determining that transition. When necessary, we will make this dependence explicit by writing $x(u', S_i)$ when the transition ends in S_i and $x(S_i, u)$ when the transition originates from S_i.

III. The BCJR Algorithm

In this section, we will restate in our new notations, without derivation, the algorithm described in [3], which is the optimum algorithm to produce the sequence of APP. We will call this algorithm the

[2] The state S_i and the symbol u' uniquely specify the precursor $S_i^-(u')$ in the case of the class of recursive convolutional encoders, like the ones we are interested in (when the largest degree of feedback polynomial represents the memory of a convolutional encoder). The extension to the case of feed-forward encoders and other nonconventional recursive convolutional encoders is straightforward.

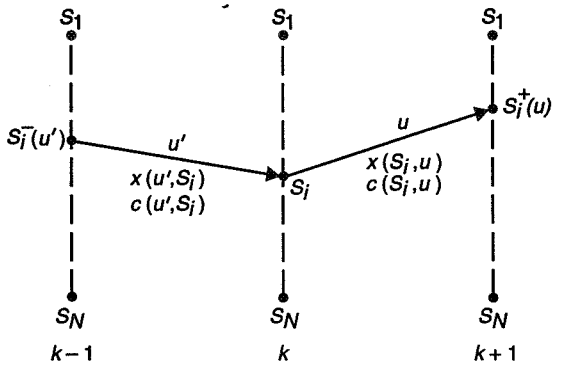

Fig. 3. The meaning of notations.

BCJR algorithm from the authors' initials.[3] We consider first the original version of the algorithm, which applies to the case of a finite index set $K = \{1, \cdots, n\}$ and requires the knowledge of the whole received sequence $\mathbf{y} = (y_1, \cdots, y_n)$ to work. In the following, the notations $\mathbf{u}, \mathbf{c}, \mathbf{x}$, and $\mathbf{y}$ will refer to sequences n-symbols long, and the integer time variable k will assume the values $1, \cdots, n$. As for the previous assumption, the encoder admits a trellis representation with N states, so that the code sequences $\mathbf{c}$ (and the corresponding transmitted signal sequences $\mathbf{x}$) can be represented as paths in the trellis and uniquely associated with a state sequence $\mathbf{s} = (s_0, \cdots, s_n)$ whose first and last states, s_0 and s_n, are assumed to be known by the decoder.[4]

Defining the a posteriori transition probabilities from state S_i at time k as

$$\sigma_k(S_i, u) \triangleq P(u_k = u, s_{k-1} = S_i|\mathbf{y}) \tag{3}$$

the APP $P(u|\mathbf{y})$ we want to compute can be obtained as

$$P_k(u|\mathbf{y}) = \sum_{S_i} \sigma_k(S_i, u) \tag{4}$$

Thus, the problem of evaluating the APP is equivalent to that of obtaining the a posteriori transition probabilities defined in Eq. (3). In [3], it was proven that the APP can be computed as

$$\sigma_k(S_i, u) = h_\sigma \alpha_{k-1}(S_i)\gamma_k(x(S_i, u))\beta_k(S_i^+(u)) \tag{5}$$

where

---

[3] The algorithm is usually referred to in the recent literature as the "Bahl algorithm"; we prefer to credit all the authors: L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv.

[4] Lower-case s_k denotes the states of a sequence at time k, whereas upper-case S_i represents one particular state belonging to the set $\mathcal{S}$.

- h_σ is such that

$$\sum_{S_i,u} \sigma_k(S_i,u) = 1$$

- $\gamma_k(x(S_i,u))$ are the joint probabilities already defined in Eq. (1), i.e.,

$$\gamma_k(x) \triangleq P(y_k, x_k = x) = P(y_k|x_k = x) \cdot P(x_k = x) \tag{6}$$

The γ's can be calculated from the knowledge of the a priori probabilities of the channel input symbols x and of the transition probabilities of the channel $P(y_k|x_k = x)$. For each time k, there are M different values of γ to be computed, which are then associated to the trellis transitions to form a sort of branch metrics. This information is supplied by the symbol-by-symbol soft-output demodulator.

- $\alpha_k(S_i)$ are the probabilities of the states of the trellis at time k conditioned on the past received signals, namely,

$$\alpha_k(S_i) \triangleq P(s_k = S_i|y_1^k) \tag{7}$$

where y_1^k denotes the sequence $y_1, y_2, \cdots, y_k$. They can be obtained by the forward recursion[5]

$$\alpha_k(S_i) = h_\alpha \sum_u \alpha_{k-1}(S_i^-(u))\gamma_k(x(u,S_i)) \tag{8}$$

with h_α a constant determined through the constraint

$$\sum_{S_i} \alpha_k(S_i) = 1$$

and where the recursion is initialized as

$$\alpha_0(S_i) = \begin{cases} 1 & \text{if } S_i = s_0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

- $\beta_k(S_i)$ are the probabilities of the trellis states at time k conditioned on the future received signals $P(s_k = S_i|y_{k+1}^n)$. They can be obtained by the backward recursion

$$\beta_k(S_i) = h_\beta \sum_u \beta_{k+1}(S_i^+(u))\gamma_{k+1}(x(S_i,u)) \tag{10}$$

[5] For feed-forward encoders and nonconventional recursive convolutional encoders like $G(D) = [1, (1 + D + D^2)/(1 + D)]$ in Eq. (8), the summation should be over all possible precursors $S_i^-(u)$ that lead to the state S_i, and $x(u, S_i)$ should be replaced by $x(S_i^-(u), u)$. Then such modifications are also required for Eqs. (18) and (26). In Eqs. (22), (29), and (32), the maximum should be over all $S_i^-(u)$ that lead to S_i. The $c(u, S_i)$ should be replaced by $c(S_i^-(u), u)$.

with h_β a constant obtainable through the constraint

$$\sum_{S_i} \beta_k(S_i) = 1$$

and where the recursion is initialized as

$$\beta_n(S_i) = \begin{cases} 1 & \text{if } S_i = s_n \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

We can now formulate the BCJR algorithm by the following steps:

(1) Initialize α_0 and β_n according to Eqs. (9) and (11).

(2) As soon as each term y_k of the sequence $\mathbf{y}$ is received, the demodulator supplies to the decoder the "branch metrics" γ_k of Eq. (6), and the decoder computes the probabilities α_k according to Eq. (8). The obtained values of $\alpha_k(S_i)$ as well as the γ_k are stored for all k, S_i, and x.

(3) When the entire sequence $\mathbf{y}$ has been received, the decoder recursively computes the probabilities β_k according to the recursion of Eq. (10) and uses them together with the stored α's and γ's to compute the a posteriori transition probabilities $\sigma_k(S_i, u)$ according to Eq. (5) and, finally, the APP $P_k(u|\mathbf{y})$ from Eq. (4).

A few comments on the computational complexity of the finite-sequence BCJR algorithm can be found in [3].

IV. The Sliding Window BCJR (SW-BCJR)

As previous description made clear, the BCJR algorithm requires that the whole sequence has been received before starting the decoding process. In this aspect, it is similar to the Viterbi algorithm in its optimum version. To apply it in a PCCC, we need to subdivide the information sequence into blocks,[6] decode them by terminating the trellises of both CCs,[7] and then decode the received sequence block by block. Beyond the rigidity, this solution also reduces the overall code rate.

A more flexible decoding strategy is offered by a modification of the BCJR algorithm in which the decoder operates on a fixed memory span, and decisions are forced with a given delay D. We call this new, and suboptimal, algorithm the sliding window BCJR (SW-BCJR) algorithm. We will describe two versions of the sliding window BCJR algorithm that differ in the way they overcome the problem of initializing the backward recursion without having to wait for the entire sequence. We will describe the two algorithms using the previous step description suitably modified. Of the previous assumptions, we retain only that of the knowledge of the initial state s_0, and thus assume the transmission of semi-infinite code sequences, where the time span K ranges from 1 to ∞.

---

[6] The presence of the interleaver naturally points toward a block length equal to the interleaver length.

[7] The termination of trellises in a PCCC has been considered a hard problem by several authors. As shown in [13], it is, indeed, quite an easy task.

A. The First Version of the Sliding Window BCJR Algorithm (SW1-BCJR)

Here are the steps:

(1) Initialize α_0 according to Eq. (9).

(2) Forward recursion at time k: Upon receiving y_k, the demodulator supplies to the decoder the M distinct branch metrics, and the decoder computes the probabilities $\alpha_k(S_i)$ according to Eqs. (6) and (8). The obtained values of $\alpha_k(S_i)$ are stored for all S_i, as well as the $\gamma_k(x)$.

(3) Initialization of the backward recursion ($k > D$):

$$\beta_k(S_j) = \alpha_k(S_j), \ \forall S_j \tag{12}$$

(4) Backward recursion: It is performed according to Eq. (10) from time $k-1$ back to time $k-D$.

(5) The a posteriori transition probabilities at time $k-D$ are computed according to

$$\sigma_{k-D}(S_i, u) = h_\sigma \cdot \alpha_{k-D-1}(S_i)\gamma_{k-D}(x(S_i, u))\beta_{k-D}(S_i^+(u)) \tag{13}$$

(6) The APP at time $k-D$ is computed as

$$P_{k-D}(u|\mathbf{y}) = \sum_{S_i} \sigma_{k-D}(S_i, u) \tag{14}$$

For a convolutional code with parameters (k_0, n_0), number of states N, and cardinality of the code alphabet $M = 2^{n_0}$, the SW1-BCJR algorithm requires storage of $N \times D$ values of α's and $M \times D$ values of the probabilities $\gamma_k(x)$ generated by the soft demodulator. Moreover, to update the α's and β's for each time instant, the algorithm needs to perform $M \times 2^{k_0}$ multiplications and N additions of 2^{k_0} numbers. To output the set of APP at each time instant, we need a D-times long backward recursion. Thus, the computational complexity requires overall

- $(D+1)M \times 2^{k_0}$ multiplications
- $(D+1)M$ additions of 2^{k_0} numbers each

As a comparison,[8] the Viterbi algorithm would require, in the same situation, $M \times 2^{k_0}$ additions and $M \times 2^{k_0}$-way comparisons, plus the trace-back operations, to get the decoded bits.

B. The Second, Simplified Version of the Sliding Window BCJR Algorithm (SW2-BCJR)

A simplification of the sliding window BCJR that significantly reduces the memory requirements consists of the following steps:

[8] Though, indeed, not fair, as the Viterbi algorithm does not provide the information we need.

(1) Initialize α_0 according to Eq. (9).

(2) Forward recursion ($k > D$): If $k > D$, the probabilities $\alpha_{k-D-1}(S_i)$ are computed according to Eq. (8).

(3) Initialization of the backward recursion ($k > D$):

$$\beta_k(S_j) = \frac{1}{N}, \ \forall S_j \tag{15}$$

(4) Backward recursion ($k > D$): It is performed according to Eq. (10) from time $k-1$ back to time $k-D$.

(5) The a posteriori transition probabilities at time $k-D$ are computed according to

$$\sigma_{k-D}(S_i, u) = h_\sigma \cdot \alpha_{k-D-1}(S_i)\gamma_{k-D}(x(S_i, u))\beta_{k-D}(S_i^+(u)) \tag{16}$$

(6) The APP at time $k-D$ is computed as

$$P_{k-D}(u|\mathbf{y}) = \sum_{S_i} \sigma_{k-D}(S_i, u) \tag{17}$$

This version of the sliding window BCJR algorithm does not require storage of the $N \times D$ values of α's as they are updated with a delay of D steps. As a consequence, only N values of α's and $M \times D$ values of the probabilities $\gamma_k(x)$ generated by the soft demodulator must be stored. The computational complexity is the same as the previous version of the algorithm. However, since the initialization of the β recursion is less accurate, a larger value of D should be set in order to obtain the same accuracy on the output values $P_{k-D}(u|\mathbf{y})$. This observation will receive quantitative evidence in the section devoted to simulation results.

V. Additive Algorithms

A. The Log-BCJR

The BCJR algorithm and its sliding window versions have been stated in multiplicative form. Owing to the monotonicity of the logarithm function, they can be converted into an additive form passing to the logarithms. Let us define the following logarithmic quantities:

$$\Gamma_k(x) \triangleq \log[\gamma(x)]$$

$$A_k(S_i) \triangleq \log[\alpha_k(S_i)]$$

$$B_k(S_i) \triangleq \log[\beta_k(S_i)]$$

$$\Sigma_k(S_i, u) \triangleq \log[\sigma_k(S_i, u)]$$

These definitions lead to the following A and B recursions, derived from Eqs. (8), (10), and (5):

$$A_k(S_i) = \log\left[\sum_u \exp\left\{A_{k-1}(S_i^-(u)) + \Gamma_k(x(u, S_i))\right\}\right] + H_A \tag{18}$$

$$B_k(S_i) = \log\left[\sum_u \exp\left\{\Gamma_{k+1}\left(x(S_i, u)\right) + B_{k+1}\left(S_i^+(u)\right)\right\}\right] + H_B \tag{19}$$

$$\Sigma_k(S_i, u) = A_{k-1}(S_i) + \Gamma_k(x(S_i, u)) + B_k(S_i^+(u)) + H_\Sigma \tag{20}$$

with the following initializations:

$$A_0(S_i) = \begin{cases} 0 & \text{if } S_i = s_0 \\ -\infty & \text{otherwise} \end{cases}$$

$$B_1(S_i) = \begin{cases} 0 & \text{if } S_i = s_n \\ -\infty & \text{otherwise} \end{cases}$$

B. Simplified Versions of the Log-BCJR

The problem in the recursions defined for the log-BCJR consists of the evaluation of the logarithm of a sum of exponentials:

$$\log\left[\sum_i \exp\{A_i\}\right]$$

An accurate estimate of this expression can be obtained by extracting the term with the highest exponential,

$$A_M = \max_i A_i$$

so that

$$\log\left[\sum_i \exp\{A_i\}\right] = A_M + \log\left(1 + \sum_{A_i \neq A_M} \exp\{A_i - A_M\}\right) \tag{21}$$

and by computing the second term of the right-hand side (RHS) of Eq. (21) using lookup tables. Further simplifications and the required circuits for implementation are discussed in the Appendix.

However, when $A_M \gg A_i$, the second term can be neglected. This approximation leads to the additive logarithmic-BCJR (AL-BCJR) algorithm:

$$A_k(S_i) = \max_u \left[A_{k-1}(S_i^-(u)) + \Gamma_k(x(u, S_i))\right] + H_A \tag{22}$$

$$B_k(S_i) = \max_u \left[B_{k+1}(S_i^+(u)) + \Gamma_{k+1}(x(S_i, u))\right] + H_B \tag{23}$$

$$\Sigma_k(S_i, u) = A_{k-1}(S_i) + \Gamma_k(x(S_i, u)) + B_k(S_i^+(u)) + H_\Sigma \tag{24}$$

with the same initialization of the log-BCJR.

Both versions of the SW-BCJR algorithm described in the previous section can be used, with obvious modifications, to transform the block log-BCJR and the AL-BCJR into their sliding window versions, leading to the SW-log-BCJR and the SWAL1-BCJR and SWAL2-BCJR algorithms.

VI. Explicit Algorithms for Some Particular Cases

In this section, we will make explicit the quantities considered in the previous algorithms' descriptions by making assumptions on the code type, modulation format, and channel.

A. Rate 1/*n* Binary Systematic Convolutional Encoder

In this section, we particularize the previous equations in the case of a rate $1/n$ binary systematic encoder associated to n binary-pulse amplitude modulation (PAM) signals or binary phase shift keying (PSK) signals.

The channel symbols x and the output symbols from the encoder can be represented as vectors of n binary components:

$$\tilde{c} \triangleq [c_1, \cdots, c_n] c_i \in \{0, 1\}$$

$$\tilde{x} \triangleq [x_1, \cdots, x_n] x_i \in \{A, -A\}$$

$$\tilde{x}_k \triangleq [x_{k1}, \cdots, x_{kn}]$$

$$\tilde{y}_k \triangleq [y_{k1}, \cdots, y_{kn}]$$

where the notations have been modified to show the vector nature of the symbols. The joint probabilities $\gamma_k(\tilde{x})$, over a memoryless channel, can be split as

$$\gamma_k(\tilde{x}) = \prod_{m=1}^{n} P(y_{km} | x_{km} = x_m) P(x_{km} = x_m) \tag{25}$$

Since in this case the encoded symbols are n-tuple of binary symbols, it is useful to redefine the input probabilities, γ, in terms of the likelihood ratios:

$$\lambda_{km} \triangleq \frac{P(y_{km}|x_{km}=A)}{P(y_{km}|x_{km}=-A)}$$

$$\lambda_{km}^{A} \triangleq \frac{P(x_{km}=A)}{P(x_{km}=-A)}$$

so that, from Eq. (25),

$$\gamma_k(\tilde{x}) = \prod_{m=1}^{n} \frac{(\lambda_{km})^{c_m}}{1+\lambda_{km}} \frac{(\lambda_{km}^{A})^{c_m}}{1+\lambda_{km}^{A}} = h_\gamma \prod_{m=1}^{n} \left[\lambda_{km} \cdot \lambda_{km}^{A}\right]^{c_m}$$

where h_γ takes into account all terms independent of $\tilde{x}$.

The BCJR can be restated as follows:

$$\alpha_k(S_i) = h_\gamma h_\alpha \sum_u \alpha_{k-1}(S_i^-(u)) \prod_{m=1}^{n} \left[\lambda_{km} \cdot \lambda_{km}^{A}\right]^{c_m(u,S_i)} \tag{26}$$

$$\beta_k(S_i) = h_\gamma h_\beta \sum_u \beta_{k+1}(S_i^+(u)) \prod_{m=1}^{n} \left[\lambda_{(k+1)m} \cdot \lambda_{(k+1)m}^{A}\right]^{c_m(S_i,u)} \tag{27}$$

$$\sigma_k(S_i,u) = h_\gamma h_\sigma \alpha_{k-1}(S_i) \prod_{m=1}^{n} \left[\lambda_{km} \cdot \lambda_{km}^{A}\right]^{c_m(u,S_i)} \beta_k(S_i^+(u)) \tag{28}$$

whereas its simplification, the AL-BCJR algorithm, becomes

$$A_k(S_i) = \max_u \left\{ A_{k-1}(S_i^-(u)) + \sum_{m=1}^{n} c_m(u,S_i) \left(\Lambda_{km} + \Lambda_{km}^{A}\right) \right\} + H_A \tag{29}$$

$$B_k(S_i) = \max_u \left\{ B_{k+1}(S_i^+(u)) + \sum_{m=1}^{n} c_m(S_i,u) \left(\Lambda_{km} + \Lambda_{km}^{A}\right) \right\} + H_B \tag{30}$$

$$\Sigma_k(S_i,u) = A_{k-1}(S_i) + \sum_{m=1}^{n} c_m(S_i,u) \left(\Lambda_{km} + \Lambda_{km}^{A}\right) + B_k(S_i^+(u)) \tag{31}$$

where Λ stands for the logarithm of the corresponding quantity λ.

B. The Additive White Gaussian Noise Channel

When the channel is the additive white Gaussian noise (AWGN) channel, we obtain the explicit expression of the log–likelihood ratios Λ_{ki} as

$$\Lambda_{ki} \triangleq \log\left[\frac{P(y_{ki}|x_{kt}=A)}{P(y_{ki}|x_{ki}=-A)}\right]$$

$$=\log\left[\frac{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\{-\frac{1}{2\sigma^2}(y_{ki}-A)^2\}}{\frac{1}{\sqrt{2\pi\sigma^2}}\exp\{-\frac{1}{2\sigma^2}(y_{ki}+A)^2\}}\right]=\frac{2A}{\sigma^2}y_{ki}$$

Hence, the AL-BCJR algorithm assumes the following form:

$$A_k(S_i)=\max_u\left\{A_{k-1}(S_i^-(u))+\sum_{m=1}^{n}c_m(u,S_i)\left(\frac{2A}{\sigma^2}y_{km}+\Lambda_{km}^A\right)\right\}+H_A \tag{32}$$

$$B_k(S_i)=\max_u\left\{B_{k+1}(S_i^+(u))+\sum_{m=1}^{n}c_m(S_i,u)\left(\frac{2A}{\sigma^2}y_{km}+\Lambda_{km}^A\right)\right\}+H_B \tag{33}$$

$$\Sigma_k(S_i,u)=A_{k-1}(S_i)+\sum_{m=1}^{n}c_m(S_i,u)\left(\frac{2A}{\sigma^2}y_{km}+\Lambda_{km}^A\right)+B_k(S_i^+(u)) \tag{34}$$

In the examples presented in Section VIII, we will consider turbo codes with rate 1/2 component convolutional codes transmitted as binary PAM or binary PSK over an AWGN channel.

VII. Iterative Decoding of Parallel Concatenated Convolutional Codes

In this section, we will show how the MAP algorithms previously described can be embedded into the iterative decoding procedure of parallel concatenated codes. We will derive the iterative decoding algorithm through suitable approximations performed on maximum-likelihood decoding. The description will be based on the fairly general parallel concatenated code shown in Fig. 4, which employs three encoders and three interleavers (denoted by π in the figure).

Let u_k be the binary random variable taking values in $\{0,1\}$, representing the sequence of information bits $\mathbf{u}=(u_1,\cdots,u_n)$. The optimum decision algorithm on the kth bit u_k is based on the conditional log–likelihood ratio L_k:

$$L_k=\log\frac{P(u_k=1|\mathbf{y})}{P(u_k=0|\mathbf{y})}$$

$$=\log\frac{\sum_{\mathbf{u}:u_k=1}P(\mathbf{y}|\mathbf{u})\prod_{j\neq k}P(u_j)}{\sum_{\mathbf{u}:u_k=0}P(\mathbf{y}|\mathbf{u})\prod_{j\neq k}P(u_j)}+\log\frac{P(u_k=1)}{P(u_k=0)}$$

$$=\log\frac{\sum_{\mathbf{u}:u_k=1}P(\mathbf{y}|\mathbf{x}(\mathbf{u}))\prod_{j\neq k}P(u_j)}{\sum_{\mathbf{u}:u_k=0}P(\mathbf{y}|\mathbf{x}(\mathbf{u}))\prod_{j\neq k}P(u_j)}+\log\frac{P(u_k=1)}{P(u_k=0)} \tag{35}$$

where, in Eq. (35), $P(u_j)$ are the a priori probabilities.

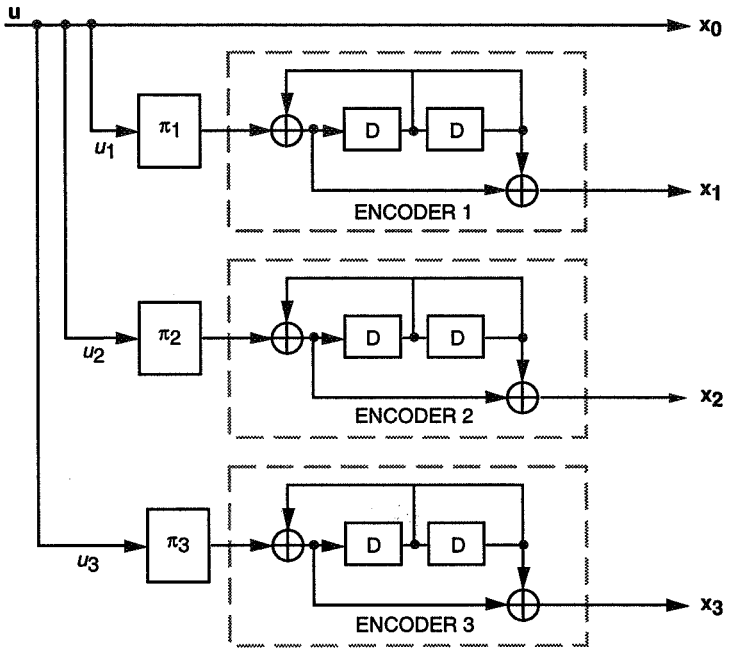

Fig. 4. Parallel concatenation of three convolutional codes.

If the rate k_o/n_o constituent code is not equivalent to a punctured rate $1/n_o'$ code or if turbo trellis-coded modulation is used, we can first use the symbol MAP algorithm as described in the previous sections to compute the log–likelihood ratio of a symbol $\mathbf{u} = u_1, u_2, \cdots, u_{k_o}$, given the observation $\mathbf{y}$ as

$$\lambda(\mathbf{u}) = \log \frac{P(\mathbf{u}|\mathbf{y})}{P(\mathbf{0}|\mathbf{y})}$$

where $\mathbf{0}$ corresponds to the all-zero symbol. Then we obtain the log–likelihood ratios of the jth bit within the symbol by

$$L(u_j) = \log \frac{\sum_{\mathbf{u}:u_j=1} e^{\lambda(\mathbf{u})}}{\sum_{\mathbf{u}:u_j=0} e^{\lambda(\mathbf{u})}}$$

In this way, the turbo decoder operates on bits, and bit, rather than symbol, interleaving is used.

To explain the basic decoding concept, we restrict ourselves to three codes, but extension to several codes is straightforward. In order to simplify the notation, consider the combination of the permuter (interleaver) and the constituent encoder connected to it as a block code with input $\mathbf{u}$ and outputs $\mathbf{x}_i$, $i = 0, 1, 2, 3 (\mathbf{x}_0 = \mathbf{u})$ and the corresponding received sequences as $\mathbf{y}_i$, $i = 0, 1, 2, 3$. The optimum bit decision metric on each bit is (for data with uniform a priori probabilities)

$$L_k = \log \frac{\sum_{\mathbf{u}:u_k=1} P(\mathbf{y}_0|\mathbf{u})P(\mathbf{y}_1|\mathbf{u})P(\mathbf{y}_2|\mathbf{u})P(\mathbf{y}_3|\mathbf{u})}{\sum_{\mathbf{u}:u_k=0} P(\mathbf{y}_0|\mathbf{u})P(\mathbf{y}_1|\mathbf{u})P(\mathbf{y}_2|\mathbf{u})P(\mathbf{y}_3|\mathbf{u})} \tag{36}$$

but, in practice, we cannot compute Eq. (36) for large n because the permutations π_2, π_3 imply that $\mathbf{y}_2$ and $\mathbf{y}_3$ are no longer simple convolutional encodings of $\mathbf{u}$. Suppose that we evaluate $P(\mathbf{y}_i|\mathbf{u})$, $i = 0, 2, 3$ in Eq. (36) using Bayes' rule and using the following approximation:

$$P(\mathbf{u}|\mathbf{y}_i) \approx \prod_{k=1}^{n} \tilde{P}_i(u_k) \tag{37}$$

Note that $P(\mathbf{u}|\mathbf{y}_i)$ is not separable in general. However, for $i = 0$, $P(\mathbf{u}|\mathbf{y}_0)$ is separable; hence, Eq. (37) holds with equality. So we need an algorithm that approximates a nonseparable distribution $P(\mathbf{u}|\mathbf{y}_i) \triangleq P$ with a separable distribution $\prod_{k=1}^{n} \tilde{P}_i(u_k) \triangleq Q$. The best approximation can be obtained using the Kullback cross-entropy minimizer, which minimizes the cross-entropy $H(Q,P) = E\{\log(Q/P)\}$ between the input P and the output Q.

The MAP algorithm approximates a nonseparable distribution with a separable one; however it is not clear how good it is compared with the Kullback cross-entropy minimizer. Here we use the MAP algorithm for such an approximation. In the iterative decoding, as the reliability of the $\{u_k\}$ improves, intuitively one expects that the cross-entropy between the input and the output of the MAP algorithm will decrease, so that the approximation will improve. If such an approximation, i.e., Eq. (37), can be obtained, we can use it in Eq. (36) for $i = 2$ and $i = 3$ (by Bayes' rule) to complete the algorithm.

Define $\tilde{L}_{ik}$ by

$$\tilde{P}_i(u_k) = \frac{e^{u_k \tilde{L}_{ik}}}{1 + e^{\tilde{L}_{ik}}} \tag{38}$$

where $u_k \in \{0,1\}$. To obtain $\{\tilde{P}_i\}$ or, equivalently, $\{\tilde{L}_{ik}\}$, we use Eqs. (37) and (38) for $i = 0,2,3$ (by Bayes' rule) to express Eq. (36) as

$$L_k = f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k) + \tilde{L}_{0k} + \tilde{L}_{2k} + \tilde{L}_{3k} \tag{39}$$

where $\tilde{L}_{0k} = 2A\mathbf{y}_{0k}/\sigma^2$ (for binary modulation) and

$$f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k) = \log \frac{\sum_{\mathbf{u}:u_k=1} P(\mathbf{y}_1|\mathbf{u}) \prod_{j \neq k} e^{u_j(\tilde{L}_{0j} + \tilde{L}_{2j} + \tilde{L}_{3j})}}{\sum_{\mathbf{u}:u_k=0} P(\mathbf{y}_1|\mathbf{u}) \prod_{j \neq k} e^{u_j(\tilde{L}_{0j} + \tilde{L}_{2j} + \tilde{L}_{3j})}} \tag{40}$$

We can use Eqs. (37) and (38) again, but this time for $i = 0,1,3$, to express Eq. (36) as

$$L_k = f(\mathbf{y}_2, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3, k) + \tilde{L}_{0k} + \tilde{L}_{1k} + \tilde{L}_{3k} \tag{41}$$

and similarly,

$$L_k = f(\mathbf{y}_3, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2, k) + \tilde{L}_{0k} + \tilde{L}_{1k} + \tilde{L}_{2k} \tag{42}$$

A solution to Eqs. (39), (41), and (42) is

$$\left.\begin{aligned} \tilde{L}_{1k} &= f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k) \\ \tilde{L}_{2k} &= f(\mathbf{y}_2, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3, k) \\ \tilde{L}_{3k} &= f(\mathbf{y}_3, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2, k) \end{aligned}\right\} \tag{43}$$

for $k = 1, 2, \cdots, n$, provided that a solution to Eq. (43) does indeed exist. The final decision is then based on

$$L_k = \tilde{L}_{0k} + \tilde{L}_{1k} + \tilde{L}_{2k} + \tilde{L}_{3k} \tag{44}$$

which is passed through a hard limiter with zero threshold. We attempted to solve the nonlinear equations in Eq. (43) for $\tilde{\mathbf{L}}_1$, $\tilde{\mathbf{L}}_2$, and $\tilde{\mathbf{L}}_3$ by using the iterative procedure

$$\tilde{L}_{1k}^{(m+1)} = \alpha_1^{(m)} f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2^{(m)}, \tilde{\mathbf{L}}_3^{(m)}, k) \tag{45}$$

for $k = 1, 2, \cdots, n$, iterating on m. Similar recursions hold for $\tilde{L}_{2k}^{(m)}$ and $\tilde{L}_{3k}^{(m)}$.

We start the recursion with the initial condition $\tilde{\mathbf{L}}_1^{(0)} = \tilde{\mathbf{L}}_2^{(0)} = \tilde{\mathbf{L}}_3^{(0)} = \tilde{\mathbf{L}}_0$. For the computation of $f(\cdot)$, we can use any MAP algorithm as described in the previous sections, with permuters (direct and inverse) where needed; call this the basic decoder D_i, $i = 1, 2, 3$. The $\tilde{L}_{ik}^{(m)}, i = 1, 2, 3$ represent the extrinsic information. The signal flow graph for extrinsic information is shown in Fig. 5 [13], which is a fully connected graph without self-loops. Parallel, serial, or hybrid implementations can be realized based on the signal flow graph of Fig. 5 (in this figure $\mathbf{y}_0$ is considered as part of $\mathbf{y}_1$). Based on our equations, each node's output is equal to internally generated reliability $\mathbf{L}$ minus the sum of all inputs to that node. The BCJR MAP algorithm always starts and ends at the all-zero state since we always terminate the trellis as described in [13]. We assumed $\pi_1 = I$ identity; however, any π_1 can be used.

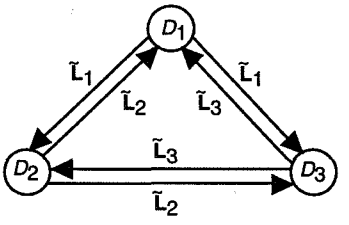

Fig. 5. Signal flow graph for extrinsic information.

The overall decoder is composed of block decoders D_i connected in parallel, as in Fig. 6 (when the switches are in position P), which can be implemented as a pipeline or by feedback. A serial implementation is also shown in Fig. 6 (when the switches are in position S). Based on [13, Fig. 5], a serial implementation was proposed in [21]. For those applications where the systematic bits are not transmitted or for parallel concatenated trellis codes with high-level modulation, we should set $\tilde{L}_0 = 0$. Even in the presence of systematic bits, if desired, one can set $\tilde{L}_0 = 0$ and consider $\mathbf{y}_0$ as part of $\mathbf{y}_1$. If the systematic bits are distributed among encoders, we use the same distribution for $\mathbf{y}_0$ among the received observations for MAP decoders.

At this point, further approximation for iterative decoding is possible if one term corresponding to a sequence $\mathbf{u}$ dominates other terms in the summation in the numerator and denominator of Eq. (40). Then the summations in Eq. (40) can be replaced by "maximum" operations with the same indices, i.e., replacing $\sum_{\mathbf{u}:u_k=i}$ with $\max_{\mathbf{u}:u_k=i}$ for $i = 0, 1$. A similar approximation can be used for $\tilde{L}_{2k}$ and $\tilde{L}_{3k}$ in Eq. (43). This suboptimal decoder then corresponds to an iterative decoder that uses AL-BCJR rather than BCJR decoders. As discussed, such approximations have been used by replacing $\sum$ with max in the log-BCJR algorithm to obtain AL-BCJR. Clearly, all versions of SW-BCJR can replace BCJR (MAP) decoders in Fig. 6.

For turbo codes with only two constituent codes, Eq. (45) reduces to

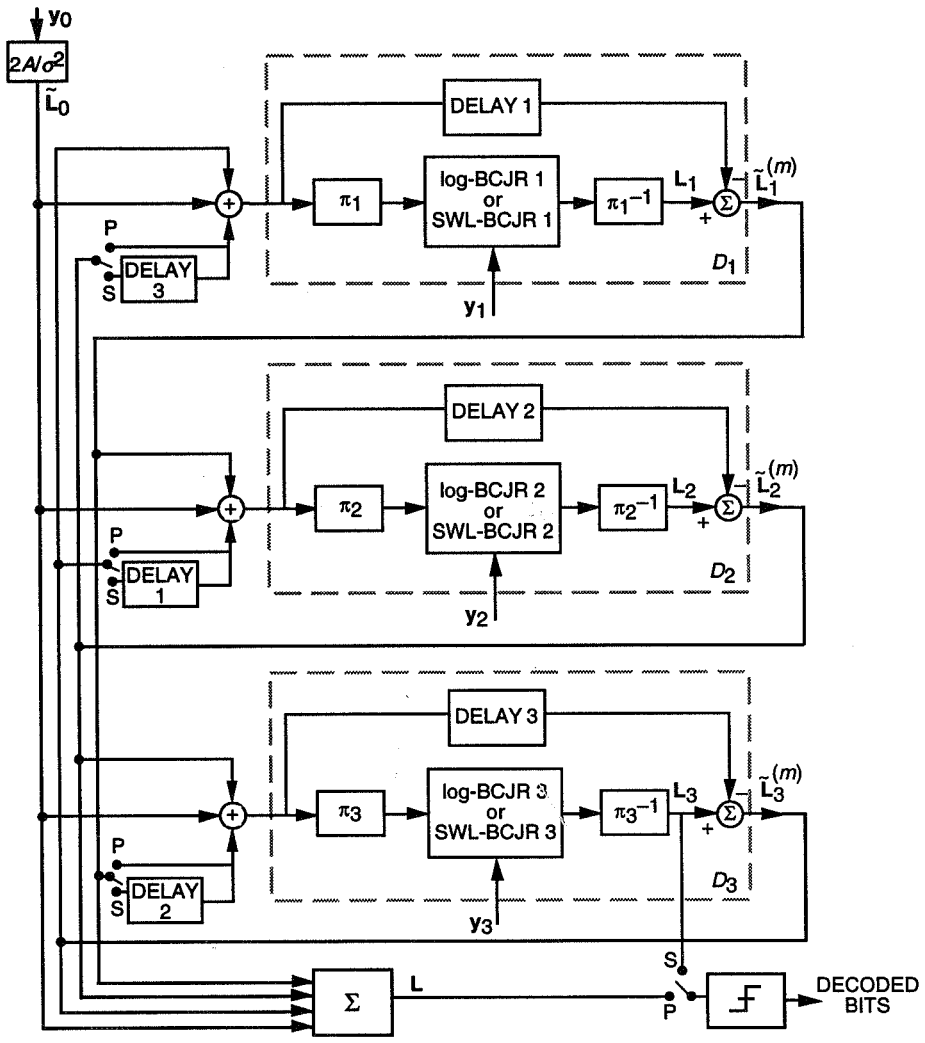

Fig. 6. Iterative decoder structure for three parallel concatenated codes.

$$\tilde{L}_{1k}^{(m+1)} = \alpha_1^{(m)} f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2^{(m)}, k)$$

$$\tilde{L}_{2k}^{(m+1)} = \alpha_2^{(m)} f(\mathbf{y}_2, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1^{(m)}, k)$$

for $k = 1, 2, \cdots, n$, and $m = 1, 2, \cdots$, where, for each iteration, $\alpha_1^{(m)}$ and $\alpha_2^{(m)}$ can be optimized (simulated annealing) or set to 1 for simplicity. The decoding configuration for two codes is shown in Fig. 7. In this special case, since the paths in Fig. 7 are disjointed, the decoder structure can be reduced to a serial mode structure if desired. If we optimize $\alpha_1^{(m)}$ and $\alpha_2^{(m)}$, our method for two codes is similar to the decoding method proposed in [6], which requires estimates of the variances of $\tilde{L}_{1k}$ and $\tilde{L}_{2k}$ for each iteration in the presence of errors. It is interesting to note that the concept of extrinsic information introduced in [6] was also presented as "partial factor" in [22]. However, the effectiveness of turbo codes lies in the use of recursive convolutional codes and random permutations. This results in time-shift-varying codes resembling random codes.

In the results presented in the next section, we will use a parallel concatenated code with only two constituent codes.

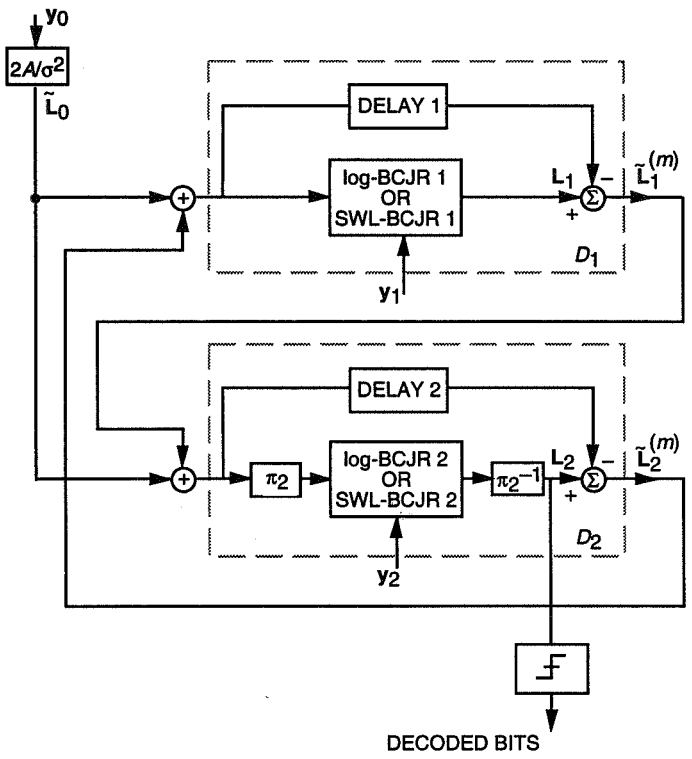

Fig. 7. Iterative decoder structure for two parallel concatenated codes.

VIII. Simulation Results

In this section, we will present some simulation results obtained applying the iterative decoding algorithm described in Section VII, which, in turn, uses the optimum BCJR and the suboptimal, but simpler, SWAL2-BCJR as embedded MAP algorithms. All simulations refer to a rate 1/3 PCCC with two equal, recursive convolutional constituent codes with 16 states and generator matrix

$$G(D) = \left[1, \frac{1 + D + D^3 + D^4}{1 + D^3 + D^4}\right]$$

and an interleaver of length 16,384 designed according to the procedure described in [13], using an S-random permutation with $S = 40$. Each simulation run examined at least 25,000,000 bits.

In Fig. 8, we plot the bit-error probabilities as a function of the number of iterations of the decoding procedure using the optimum block BCJR algorithm for various values of the signal-to-noise ratio. It can be seen that the decoding algorithm converges down to BER $= 10^{-5}$ at signal-to-noise ratios of 0.2 dB with nine iterations. The same curves are plotted in Fig. 9 for the case of the suboptimum SWAL2-BCJR algorithm. In this case, 0.75 dB of signal-to-noise ratio is required for convergence to the same BER and with the same number of iterations.

In Fig. 10, the bit-error probability versus the signal-to-noise ratio is plotted for a fixed number (5) of iterations of the decoding algorithm and for both optimum BCJR and SWAL2-BCJR MAP decoding algorithms. It can be seen that the penalty incurred by the suboptimum algorithm amounts to about 0.5 dB. This figure is in agreement with a similar result obtained in [12], where all MAP

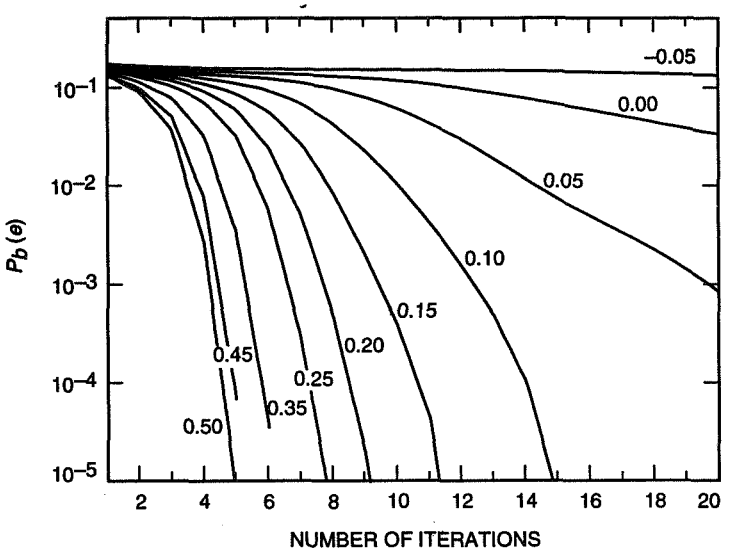

Fig. 8. Convergence of turbo coding: bit-error probability versus number of iterations for various E_b/N_0 using the SW2-BCJR algorithm.

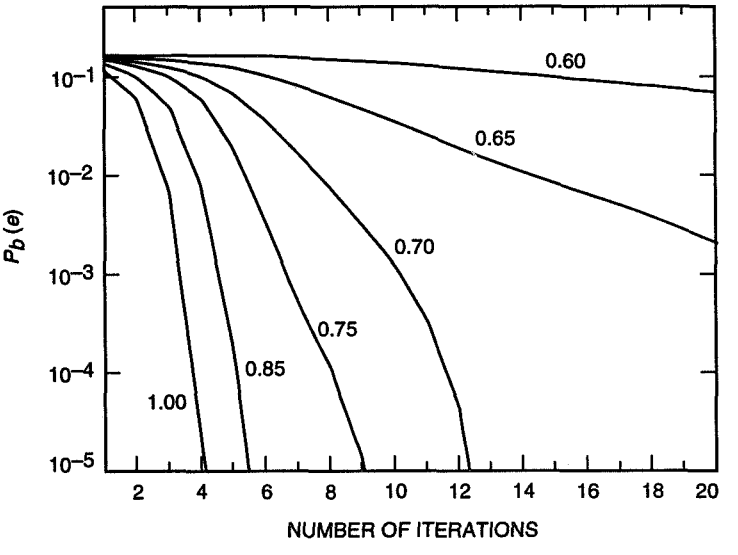

Fig. 9. Convergence of turbo coding: bit-error probability versus number of iterations for various E_b/N_0 using the SWAL2-BCJR algorithm.

algorithms were of the block type. The penalty is completely attributable to the approximation of the sum of exponentials described in Section V.B. To verify this, we have used a SW2-BCJR and compared its results with the optimum block BCJR, obtaining the same results.

Finally, in Figs. 11 and 12, we plot the number of iterations needed to obtain a given bit-error probability versus the bit signal-to-noise ratio, for the two algorithms. These curves provide information on the delay incurred to obtain a given reliability as a function of the bit signal-to-noise ratio.

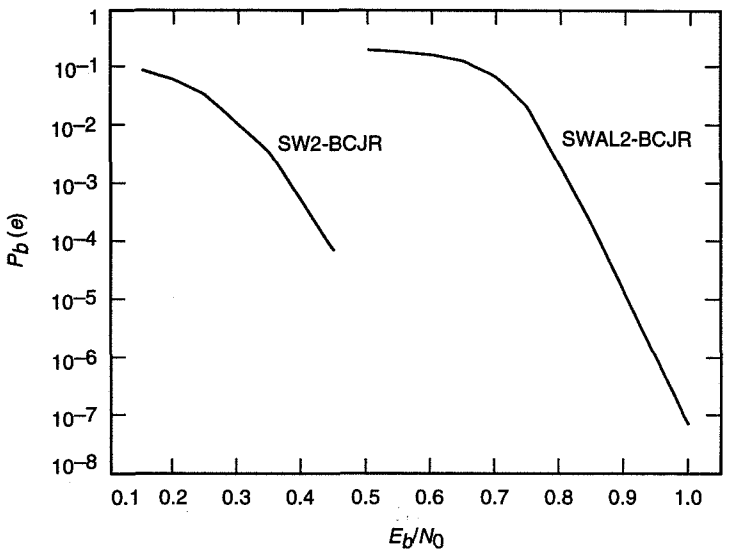

Fig. 10. Bit-error probability as a function of the bit signal-to-noise ratio using the SW2-BCJR and SWAL2-BCJR algorithms with five iterations.

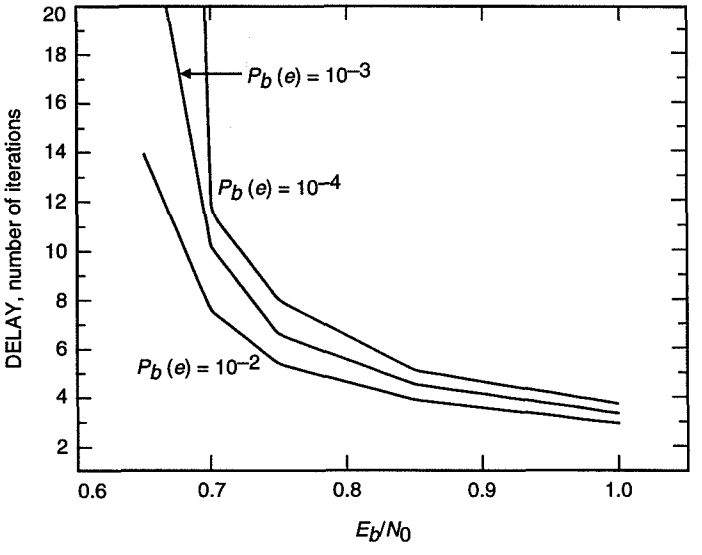

Fig. 11. Number of iterations to achieve several bit-error probabilities as a function of the bit signal-to-noise ratio using the SWAL2-BCJR algorithm.

IX. Conclusions

We have described two versions of a simplified maximum a posteriori decoding algorithm working in a sliding window form, like the Viterbi algorithm. The algorithms can be used as a building block to decode continuously transmitted sequences obtained by parallel concatenated codes, without requiring code trellis termination. A heuristic explanation of how to embed the maximum a posteriori algorithms into the iterative decoding of parallel concatenated codes was also presented. Finally, the performances of the two algorithms were compared on the basis of a powerful rate 1/3 parallel concatenated code.

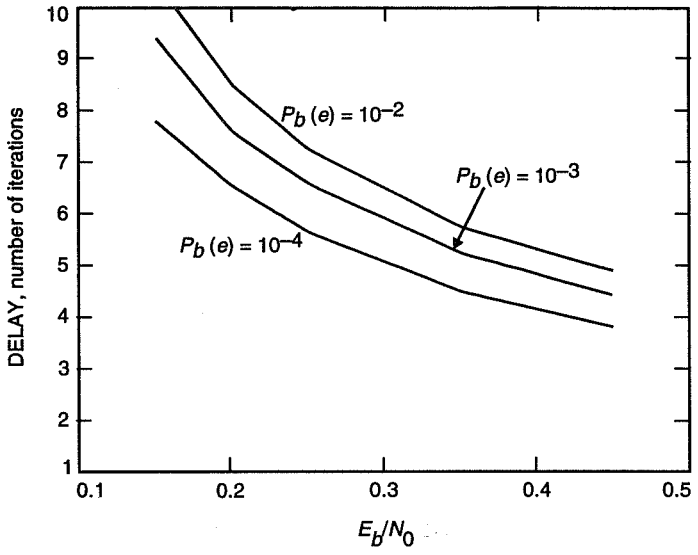

Fig. 12. Number ot iterations to achieve several bit-error probabilities as a function of the bit signal-to-noise ratio using the SW2-BCJR algorithm.

Acknowledgment

The research in this article was partially carried out at the Politecnico di Torino, Italy, under NATO Research Grant CRG 951208.

References

[1] S. Benedetto, E. Biglieri, and V. Castellani, *Digital Transmission Theory*, New York: Prentice-Hall, 1987.

[2] K. Abend and B. D. Fritchman, "Statistical Detection for Communication Channels With Intersymbol Interference," *Proceedings of IEEE*, vol. 58, no. 5, pp. 779–785, May 1970.

[3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, pp. 284–287, March 1974.

[4] G. D. Forney, Jr., *Concatenated Codes*, Cambridge, Massachusetts: Massachusetts Institute of Technology, 1966.

[5] V. V. Ginzburg, "Multidimensional Signals for a Continuous Channel," *Probl. Peredachi Inform.*, vol. 20, no. 1, pp. 28–46, January 1984.

[6] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes," *Proceedings of ICC'93*, Geneva, Switzerland, pp. 1064–1070, May 1993.

[7] N. Seshadri and C.-E. W. Sundberg, "Generalized Viterbi Algorithms for Error Detection With Convolutional Codes," *Proceedings of GLOBECOM'89*, vol. 3, Dallas, Texas, pp. 43.3.1–43.3.5, November 1989.

[8] 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.

[9] Y. Li, B. Vucetic, and Y. Sato, "Optimum Soft-Output Detection for Channels With Intersymbol Interference," *Trans. on Information Theory*, vol. 41, no. 3, pp. 704–713, May 1995.

[10] S. S. Pietrobon and A. S. Barbulescu, "A Simplification of the Modified Bahl Algorithm for Systematic Convolutional Codes," *Proceedings of ISITA'94*, Sydney, Australia, pp. 1073–1077, November 1994.

[11] U. Hansson and T. Aulin, "Theoretical Performance Evaluation of Different Soft-Output Algorithms," *Proceedings of ISITA'94*, Sydney, Australia, pp. 875–880, November 1994.

[12] P. Robertson, E. Villebrun, and P. Hoeher, "A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain," *Proceedings of ICC'95*, Seattle, Washington, pp. 1009–1013, June 1995.

[13] D. Divsalar and F. Pollara, "Turbo Codes for PCS Applications," *Proceedings of ICC'95*, Seattle, Washington, pp. 54–59, June 1995.

[14] *CAS 5093 Turbo-Code Codec*, 3.7 ed., data sheet, Chateaubourg, France: Comatlas, August 1994.

[15] S. Benedetto and G. Montorsi, "Performance of Turbo Codes," *Electronics Letters*, vol. 31, no. 3, pp. 163–165, February 1995.

[16] S. S. Pietrobon, "Implementation and Performance of a Serial MAP Decoder for Use in an Iterative Turbo Decoder," *Proceedings of ISIT'95*, Whistler, British Columbia, Canada, pp. 471, September 1995.
Also http://audrey.levels.unisa.edu.au/itr-users/steven/turbo/ISIT95ovh2.ps.gz

[17] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Transfer Function Bounds on the Performance of Turbo Codes," *The Telecommunications and Data Acquisition Progress Report 42-122, April–June 1995*, Jet Propulsion Laboratory, Pasadena, California, pp. 44–55, August 15, 1995.
http://edms-www.jpl.nasa.gov/tda/progress_report/42-122/122A.pdf

[18] S. Benedetto and G. Montorsi, " Design of Parallel Concatenated Convolutional Codes," to be published in *IEEE Transactions on Communications*, 1996.

[19] D. Divsalar and F. Pollara, "Multiple Turbo Codes," *Proceedings of IEEE MILCOM95*, San Diego, California, November 5–8, 1995.

[20] D. Divsalar and F. Pollara, "On the Design of Turbo Codes," *The Telecommunications and Data Acquisition Progress Report 42-123, July–September 1995*, Jet Propulsion Laboratory, Pasadena, California, pp. 99–121, November 15, 1995.
http://edms-www.jpl.nasa.gov/tda/progress_report/42-123/123D.pdf

[21] S. A. Barbulescu, "Iterative Decoding of Turbo Codes and Other Concatenated Codes," Ph.D. Dissertation, University of South Australia, August 1995.

[22] J. Lodge, R. Young, P. Hoeher, and J. Hagenauer, "Separable MAP 'Filters' for the Decoding of Product and Concatenated Codes," *Proceedings of ICC'93*, Geneva, Switzerland, pp. 1740–1745, May 1993.

Appendix

Circuits to Implement the MAP Algorithm for Decoding Rate 1/*n* Component Codes of a Turbo Code

In this appendix, we show the basic circuits required to implement a serial additive MAP algorithm for both block log-BCJR and SW-log-BCJR. Extension to a parallel implementation is straightforward. Figure A-1 shows the implementation[9] of Eq. (18) for the forward recursion using a lookup table for evaluation of $\log(1+e^{-x})$, and subtraction of $\max_j\{A_k(S_j)\}$ from $A_k(S_i)$ is used for normalization to prevent buffer overflow.[10] The circuit for maximization can be implemented simply by using a comparator and selector with feedback operation. Figure A-2 shows the implementation of Eq. (19) for the backward recursion, which is similar to Fig. A-1. A circuit for computation of $\log(P_k(u|\mathbf{y}))$ from Eq. (4) using Eq. (20) for final computation of bit reliability is shown in Fig. A-3. In this figure, switch 1 is in position 1 and switch 2 is open at the start of operation. The circuit accepts $\Sigma_k(S_i,u)$ for $i=1$, then switch 1 moves to position 2 for feedback operation. The circuit performs the operations for $i=1,2,\cdots,N$. When the circuit accepts $\Sigma_k(S_i,u)$ for $i=N$, switch 1 goes to position 1 and switch 2 is closed. This operation is done for $u=1$ and $u=0$. The difference between $\log(P_k(1|\mathbf{y}))$ and $\log(P_k(0|\mathbf{y}))$ represents the reliability value required for turbo decoding, i.e., the value of L_k in Eq. (35).

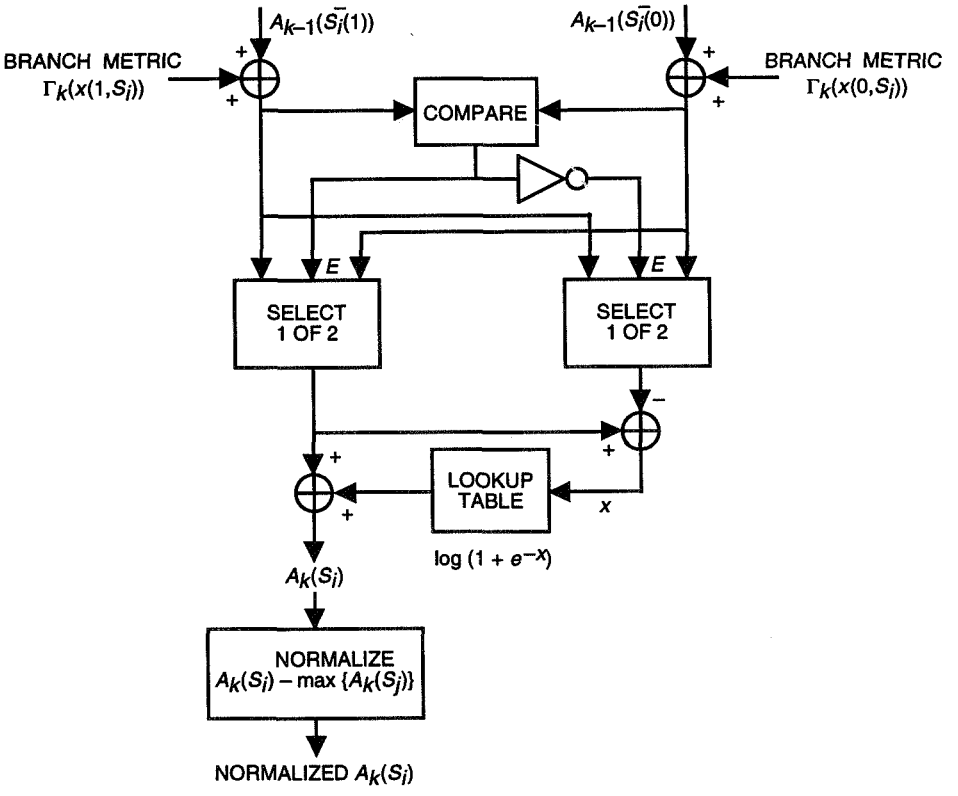

Fig. A-1. Basic structure for forward computation in the log-BCJR MAP algorithm.

[9] For feed-forward and nonconventional recursive convolutional codes, the notations in Fig. A-1 should be changed according to Footnotes 2 and 5.

[10] Simpler normalization can be achieved by monitoring the two most significant bits. When both of them are one, then we reset all the most significant bits to zero. This method increases the bit representation by an additional 2 bits.

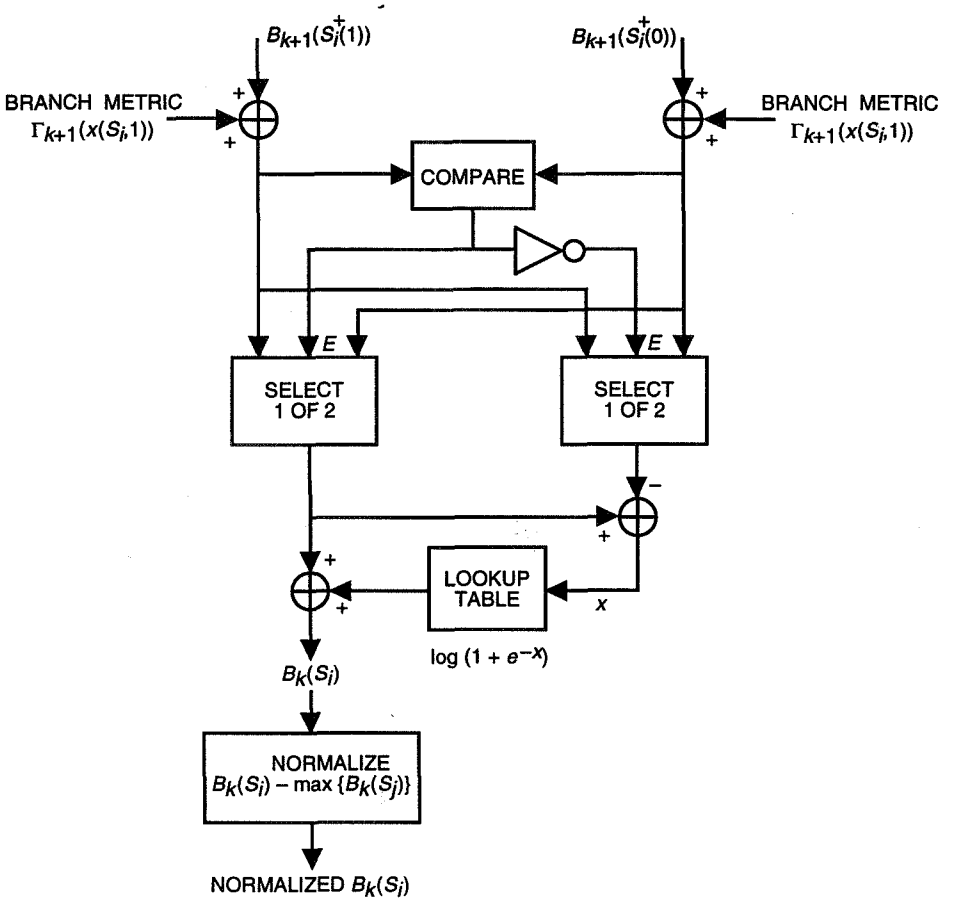

Fig. A-2. Basic structure for backward computation in the log-BCJR MAP algorithm.

We propose two simplifications to be used for computation of $\log(1 + e^{-x})$ without using a lookup table.

Approximation 1: We used the approximation $\log(1 + e^{-x}) \approx -ax + b$, $0 < x < b/a$ where $b = \log(2)$, and we selected $a = 0.3$ for the simulation. We observed about a 0.1-dB degradation compared with the full MAP algorithm for the code described in Section VIII. The parameter a should be optimized, and it may not necessarily be the same for the computation of Eq. (18), Eq. (19), and $\log(P_k(u|\mathbf{y}))$ from Eq. (4) using Eq. (20). We call this "linear" approximation.

Approximation 2: We take

$$\log(1 + e^{-x}) \approx \begin{cases} 0 & \text{if } x > \eta \\ c & \text{if } x < \eta \end{cases}$$

We selected $c = \log(2)$ and the threshold $\eta = 1.0$ for our simulation. We observed about a 0.2-dB degradation compared with the full MAP algorithm for the code described in Section VIII. This threshold should be optimized for a given SNR, and it may not necessarily be the same for the computation of Eq. (18), Eq. (19), and $\log(P_k(u|\mathbf{y}))$ from Eq. (4) using Eq. (20). If we use this approximation, the log-BCJR algorithm can be built based on addition, comparison, and selection operations without requiring a lookup table, which is similar to a Viterbi algorithm implementation. We call this "threshold" approximation. At most, 8- to 10-bit representation suffices for all operations (see also [12] and [16]).

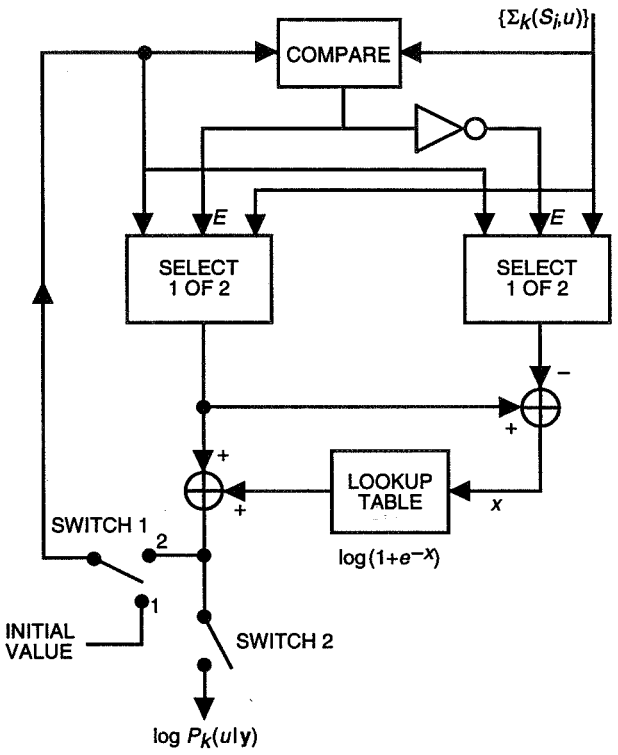

Fig. A-3. Basic structure for bit reliability computation in the log-BCJR MAP algorithm.