A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes

S. Benedetto,[a] D. Divsalar,[b] G. Montorsi,[a] and F. Pollara[b]

Concatenated coding schemes with interleavers consist of a combination of two simple constituent encoders and an interleaver. The parallel concatenation known as "turbo code" has been shown to yield remarkable coding gains close to theoretical limits, yet admitting a relatively simple iterative decoding technique. The recently proposed serial concatenation of interleaved codes may offer performance superior to that of turbo codes. In both coding schemes, the core of the iterative decoding structure is a soft-input soft-output (SISO) module. In this article, we describe the SISO module in a form that continuously updates the maximum a posteriori (MAP) probabilities of input and output code symbols and show how to embed it into iterative decoders for parallel and serially concatenated codes. Results are focused on codes yielding very high coding gain for space applications.

I. Introduction

Concatenated coding schemes have been studied by Forney [1] as a class of codes whose probability of error decreased exponentially at rates less than capacity, while decoding complexity increased only algebraically. Initially motivated only by theoretical research interests, concatenated codes have since then evolved as a standard for those applications where very high coding gains are needed, such as (deep-)space applications.

The recent proposal of "turbo codes" [2], with their astonishing performance close to the theoretical Shannon capacity limits, has once again shown the great potential of coding schemes formed by two or more codes working in a concurrent way. Turbo codes are parallel concatenated convolutional codes (PCCCs) in which the information bits are first encoded by a recursive systematic convolutional code and then, after passing through an interleaver, are encoded by a second systematic convolutional encoder. The code sequences are formed by the information bits, followed by the parity check bits generated by both encoders. Using the same ingredients, namely convolutional encoders and interleavers, serially concatenated convolutional codes (SCCCs) have been shown to yield performance comparable, and in some cases superior, to turbo codes [5].

[a] Politecnico di Torino, Torino, Italy.

[b] Communications Systems and Research Section.

Both concatenated coding schemes admit a suboptimum decoding process based on the iterations of the maximum a posteriori (MAP) algorithm [12] applied to each constituent code. The purpose of this article is to describe a soft-input soft-output module (denoted by SISO) that implements the MAP algorithm in its basic form, the extension of it to additive MAP (log-MAP), which is indeed a dual-generalized Viterbi algorithm with correction,[1] and finally extension to the continuous decoding of PCCC and SCCC. As examples of applications, we will show the results obtained by decoding two low-rate codes, with very high coding gain, aimed at deep-space applications.

II. Iterative Decoding of Parallel and Serial Concatenated Codes

In this section, we show the block diagram of parallel and serially concatenated codes, together with their iterative decoders. It is not within the scope of this article to describe and analyze the decoding algorithms. For them, the reader is directed to [2,4,6,7] (for PCCC) and [3,5,8] (for SCCC). Rather, we aim at showing that both iterative decoding algorithms need a particular module, named *soft-input, soft-output* (SISO), which implements operations strictly related to the MAP algorithm, and which will be analyzed in detail in the next section.

A. Parallel Concatenated Codes

The block diagram of a PCCC is shown in Fig. 1 (the same construction also applies to block codes). In the figure, a rate 1/3 PCCC is obtained using two rate 1/2 constituent codes (CCs) and an interleaver. For each input information bit, the codeword sent to the channel is formed by the input bit, followed by the parity check bits generated by the two encoders. In Fig. 1, the block diagram of the iterative decoder is also shown. It is based on two modules denoted by "SISO," one for each encoder, an interleaver, and a deinterleaver performing the inverse permutation with respect to the interleaver. The input and output reliabilities for each SISO module in Fig. 1 are described in Section V.

The SISO module is a four-port device, with two inputs and two outputs. A detailed description of its operations is deferred to the next section. Here, it suffices to say that it accepts as inputs the probability distributions of the information and code symbols labeling the edges of the code trellis, and forms as outputs an update of these distributions based upon the code constraints. It can be seen from Fig. 1 that the updated probabilities of the code symbols are never used by the decoding algorithm.

B. Serially Concatenated Codes

The block diagram of a SCCC is shown in Fig. 2 (the same construction also applies to block codes). In the figure, a rate 1/3 SCCC is obtained using as an *outer* encoder a rate 1/2 encoder, and as an inner encoder a rate 2/3 encoder. An interleaver permutes the output codewords of the outer code before passing them to the inner code. In Fig. 2, the block diagram of the iterative decoder is also shown. It is based on two modules denoted by "SISO," one for each encoder, an interleaver, and a deinterleaver.

The SISO module is the same as described before. In this case, though, both updated probabilities of the input and code symbols are used in the decoding procedure. The input and output reliabilities for each SISO module in Fig. 2 are described in Section V.

C. Soft-Output algorithms

The SISO module is based on MAP algorithms. MAP algorithms have been known since the early seventies [10–14]. The algorithms in [11–14] perform both forward and backward recursions and, thus, require that the whole sequence be received before starting the decoding operations. As a

[1] A. J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," submitted to the *JSAC* issue on "Concatenated Coding Techniques and Iterative Decoding: Sailing Toward Channel Capacity."

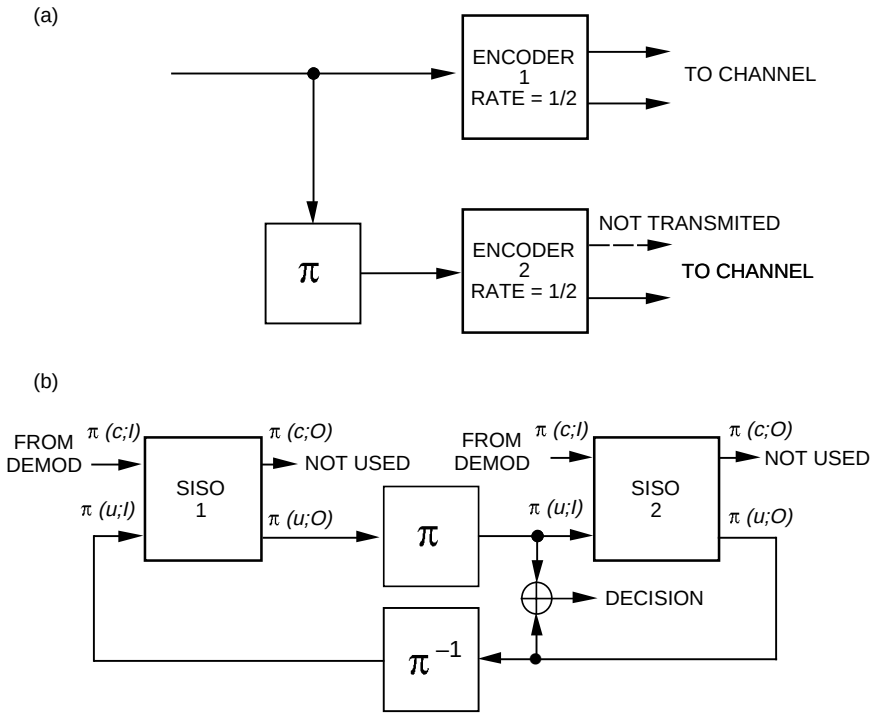

Fig. 1. Block diagram of a parallel concatenated convolutional code (PCCC): (a) a PCCC, rate = 1/3 and (b) iterative decoding of a PCCC.

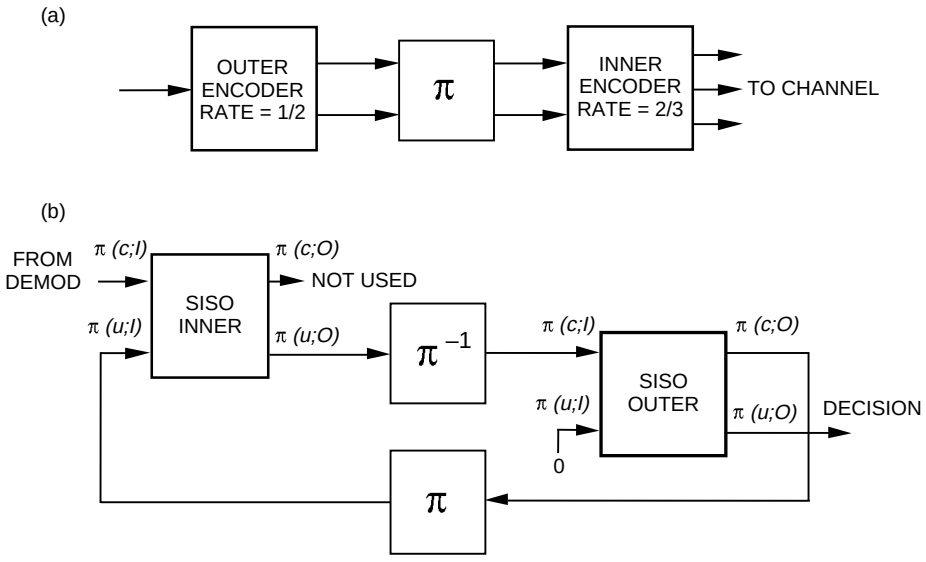

Fig. 2. Serially concatenated convolutional code (SCCC): (a) an SCCC, rate = 1/3 and (b) iterative decoding of an SCCC.

consequence, they can only be used in block-mode decoding. The memory requirement and computational complexity grow linearly with the sequence length.

The algorithm in [10] requires only a forward recursion, so that it can be used in continuous-mode decoding. However, its memory and computational complexity grow exponentially with the decoding delay. Recently, a MAP symbol-by-symbol decoding algorithm conjugating the positive aspects of previous algorithms, i.e., a fixed delay and linear memory and complexity growth with decoding delay, has been proposed in [15].

All previously described algorithms are truly MAP algorithms. To reduce the computational complexity, various forms of suboptimum soft-output algorithms have been proposed. Two approaches have been taken. The first approach tries to modify the Viterbi algorithm. Forney considered "augmented outputs" from the Viterbi algorithm [16]. These augmented outputs include the depth at which all paths are merged, the difference in length between the best and the next-best paths at the point of merging, and a given number of the most likely path sequences. The same concept of augmented output was later generalized for various applications [17–21]. A different approach to the modification of the Viterbi algorithm was followed in [22]. It consists of generating a reliability value for each bit of the hard-output signal and is called the soft-output Viterbi algorithm (SOVA). In the binary case, the degradation of SOVA with respect to MAP is small [23]; however, SOVA is not as effective in the nonbinary case. A comparison of several suboptimum soft-output algorithms can be found in [24]. The second approach consists of revisiting the original symbol MAP decoding algorithms [10,12] with the aim of simplifying them to a form suitable for implementation [15,25–30].

III. The SISO Module

A. The Encoder

The decoding algorithm underlying the behavior of SISO works for codes admitting a trellis representation. It can be a time-invariant or time-varying trellis, and, thus, the algorithm can be used for both block and convolutional codes. In the following, for simplicity of the exposition, we will refer to the case of time-invariant convolutional codes.

In Fig. 3, we show a trellis encoder, characterized by the following quantities:[2]

(1) $\mathbf{U} = (U_k)_{k \in K}$ is the sequences of input symbols, defined over a time index set K (finite or infinite) and drawn from the alphabet

$$\mathcal{U} = \{\tilde{u}_1, \ldots, \tilde{u}_{N_I}\}$$

To the sequence of input symbols, we associate the sequence of a priori probability distributions:

$$\mathbf{P}(\mathbf{u}; I) = (P_k(u_k; I))_{k \in K}$$

[2] In the following, capital letters U, C, S, E will denote random variables and lower-case letters u, c, s, e their realizations. The roman letter P[A] will denote the probability of the event A, whereas the letter $P(a)$ (italic) will denote a function of a. The subscript k will denote a discrete time, defined on the time index set K. Other subscripts, like i, will refer to elements of a finite set. Also, "()" will denote a time sequence, whereas "{}" will denote a finite set of elements.

where

$$P_k(u_k; I) \triangleq \mathrm{P}[U_k = u_k]$$

(2) $\mathbf{C} = (C_k)_{k \in K}$ is the sequences of output, or code, symbols, defined over the same time index set K, and drawn from the alphabet

$$\mathcal{C} = \{\tilde{c}_1, \ldots, \tilde{c}_{N_O}\}$$

To the sequence of output symbols, we associate the sequence of a priori probability distributions:

$$\mathbf{P}(\mathbf{c}; I) = (P_k(c_k; I))_{k \in K}$$

For simplicity of notation, we drop the dependency of u_k and c_k on k. Thus, $P_k(u_k; I)$ and $P_k(c_k; I)$ will be denoted simply by $P_k(u; I)$ and $P_k(c; I)$, respectively.

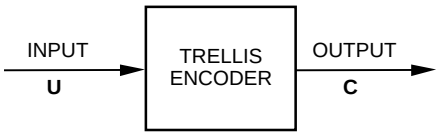

Fig. 3. The trellis encoder.

B. The Trellis Section

The dynamics of a time-invariant convolutional code are completely specified by a single trellis section, which describes the transitions (edges) between the states of the trellis at time instants k and $k+1$. A trellis section is characterized by the following:

(1) A set of N states $\mathcal{S} = \{s_1, \ldots, s_N\}$. The state of the trellis at time k is $S_k = s$, with $s \in \mathcal{S}$.

(2) A set of $N \times N_I$ edges obtained by the Cartesian product

$$\mathcal{E} = \mathcal{S} \times \mathcal{U} = \{e_1, \ldots, e_{N \times N_I}\}$$

which represents all possible transitions between the trellis states.

The following functions are associated with each edge $e \in \mathcal{E}$ (see Fig. 4):

(1) The starting state $s^S(e)$ (the projection of e onto $\mathcal{S}$).

(2) The ending state $s^E(e)$.

(3) The input symbol $u(e)$ (the projection of e onto $\mathcal{U}$).

(4) The output symbol $c(e)$.

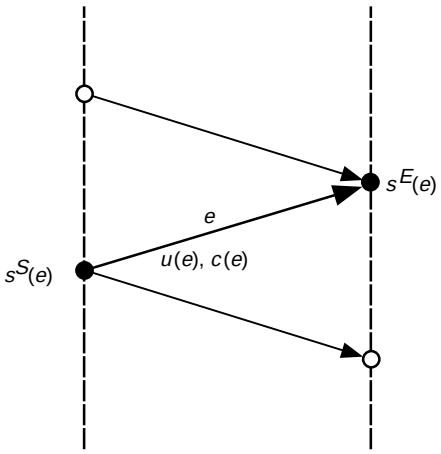

Fig. 4. An edge of the trellis section.

The relationship between these functions depends on the particular encoder. As an example, in the case of systematic encoders, $(s^E(e), c(e))$ also identifies the edge since $u(e)$ is uniquely determined by $c(e)$. In the following, we only assume that the pair $(s^S(e), u(e))$ uniquely identifies the ending state $s^E(e)$; this assumption is always verified, as it is equivalent to say that, given the initial trellis state, there is a one-to-one correspondence between input sequences and state sequences, a property required for the code to be uniquely decodable.

C. The SISO Algorithm

The SISO module is a four-port device that accepts at the input the sequences of probability distributions

$$\mathbf{P}(\mathbf{c}; I) \qquad \mathbf{P}(\mathbf{u}; I)$$

and outputs the sequences of probability distributions

$$\mathbf{P}(\mathbf{c}; O) \qquad \mathbf{P}(\mathbf{u}; O)$$

based on its inputs and on its knowledge of the trellis section (or code in general).

We assume first that the time index set K is finite, i.e., $K = \{1, \ldots, n\}$. The algorithm by which the SISO operates in evaluating the output distributions will be explained in two steps. In the first step, we consider the following algorithm:

(1) At time k, the output probability distributions are computed as

$$\tilde{P}_k(c; O) = \tilde{H}_c \sum_{e: c(e)=c} A_{k-1}[s^S(e)] P_k[u(e); I] P_k[c(e); I] B_k[s^E(e)] \tag{1}$$

$$\tilde{P}_k(u; O) = \tilde{H}_u \sum_{e: u(e)=u} A_{k-1}[s^S(e)] P_k[u(e); I] P_k[c(e); I] B_k[s^E(e)] \tag{2}$$

(2) The quantities $A_k(\cdot)$ and $B_k(\cdot)$ are obtained through the forward and backward recursions, respectively, as

$$A_k(s) = \sum_{e:s^E(e)=s} A_{k-1}[s^S(e)]P_k[u(e);I]P_k[c(e);I] \quad , k = 1, \ldots, n \tag{3}$$

$$B_k(s) = \sum_{e:s^S(e)=s} B_{k+1}[s^E(e)]P_{k+1}[u(e);I]P_{k+1}[c(e);I] \quad , k = n-1, \ldots, 0 \tag{4}$$

with initial values

$$A_0(s) = \begin{cases} 1 & s = S_0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$B_n(s) = \begin{cases} 1 & s = S_n \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The quantities $\tilde{H}_c, \tilde{H}_u$ are normalization constants defined as follows:

$$\tilde{H}_c \rightarrow \sum_c \tilde{P}_k(c;O) = 1$$

$$\tilde{H}_u \rightarrow \sum_u \tilde{P}_k(u;O) = 1$$

In the second step, from Eqs. (1) and (2), it is apparent that the quantities $P_k[c(e);I]$ in the first equation and $P_k[u(e);I]$ in the second do not depend on e, by definition of the summation indices, and thus can be extracted from the summations. Thus, defining the new quantities

$$P_k(c;O) \triangleq H_c \frac{\tilde{P}_k(c;O)}{P_k(c;I)}$$

$$P_k(u;O) \triangleq H_u \frac{\tilde{P}_k(u;O)}{P_k(u;I)}$$

where H_c and H_u are normalization constants such that

$$H_c \rightarrow \sum_c P_k(c;O) = 1$$

$$H_u \rightarrow \sum_u P_k(u;O) = 1$$

it can be easily verified that they can be obtained through the expressions

$$P_k(c;O) = H_c \tilde{H}_c \sum_{e:c(e)=c} A_{k-1}[s^S(e)] P_k[u(e);I] B_k[s^E(e)] \tag{7}$$

$$P_k(u;O) = H_u \tilde{H}_u \sum_{e:u(e)=u} A_{k-1}[s^S(e)] P_k[c(e);I] B_k[s^E(e)] \tag{8}$$

where the A's and B's satisfy the same recursions previously introduced in Eq. (3).

The new probability distributions $P_k(u;O)$ and $P_k(c;O)$ represent a smoothed version of the input distributions $P_k(c;I)$ and $P_k(u;I)$, based on the code constraints and obtained using the probability distributions of all symbols of the sequence but the kth ones, $P_k(c;I)$ and $P_k(u;I)$. In the literature of turbo decoding, $P_k(u;O)$ and $P_k(c;O)$ would be called extrinsic information. They represent the added value of the SISO module to the a priori distributions $P_k(u;I)$ and $P_k(c;I)$. Basing the SISO algorithm on $P_k(\cdot;O)$ instead of on $\tilde{P}_k(\cdot;O)$ simplifies the block diagrams, and related software and hardware, of the iterative schemes for decoding concatenated codes. For this reason, we will consider as an SISO algorithm the one expressed by Eq. (7). The SISO module is then represented as in Fig. 5.

Previously proposed algorithms were not in a form suitable for working with a general trellis code. Most of them assumed binary input symbols, some also assumed systematic codes, and none (not even the original Bahl–Cocke–Jelinek–Raviv (BCJR) algorithm) could cope with a trellis having parallel edges. As can be noticed from all summations involved in the equations that define the SISO algorithm, we work on trellis edges rather than on pairs of states, and this makes the algorithm completely general and capable of coping with parallel edges and also with encoders with rates greater than one, like those encountered in some concatenated schemes.

Fig. 5. The soft-input soft-output (SISO model).

D. Computation of Input and Output Bit Extrinsic Information

In this subsection, *bit* extrinsic information is derived from the *symbol* extrinsic information using Eqs. (7) and (8). Consider a rate k_o/n_o trellis encoder such that each input symbol U consists of k_o bits and each output symbol C consists of n_o bits. Assume

$$P_k(c;I) = \prod_{j=1}^{n_o} P_{k,j}(c^j;I) \tag{9}$$

$$P_k(u;I) = \prod_{j=1}^{k_o} P_{k,j}(u^j;I) \tag{10}$$

where $c^j \in \{0,1\}$ denotes the value of the jth bit C_k^j of the output symbol $C_k = c$; $j = 1,\ldots,n_o$, and $u^j \in \{0,1\}$ denotes the value of the jth bit U_k^j of the input symbol $U_k = u$; $j = 1,\ldots,k_o$. This assumption is valid in an iterative decoding when bit interleavers rather than symbol interleavers are used. One should be cautious when using $P_k(c;I)$ as a product for those encoders in a concatenated system

where the output C in Fig. 3 is connected to a channel. For such cases, if, for an example, a nonbinary input additive white Gaussian noise (AWGN) channel is used, this assumption usually is not needed (this will be discussed shortly), and $P_k(c;I) = P_k(c|y) = P_k(y|x(c))P(c)/P(y)$, where y is the complex received sample(s) and $x(c)$ is the transmitted nonbinary symbol(s). Then, for binary input memoryless channels, $P_k(y|x(c))$ can be written as a product. After obtaining symbol probability distributions $P_k(c;O)$ and $P_k(c;O)$ from Eqs. (7) and (8) by using Eqs. (2) and (4), it is easy then to show that the input and output bit extrinsic information can be obtained as

$$P_{k,j}(c^j;O) = H_{c^j} \sum_{c:C_k^j=c^j} P_k(c;O) \prod_{\substack{i=1 \\ i\neq j}}^{n_o} P_{k,j}(c^i;I) \tag{11}$$

$$P_{k,j}(u^j;O) = H_{u^j} \sum_{u:U_k^j=u^j} P_k(u;O) \prod_{\substack{i=1 \\ i\neq j}}^{k_o} P_{k,j}(u^i;I) \tag{12}$$

where H_{c^j} and H_{u^j} are normalization constants such that

$$H_{c^j} \rightarrow \sum_{c^j\in\{0,1\}} P_{k,j}(c^j;O) = 1$$

$$H_{u^j} \rightarrow \sum_{u^j\in\{0,1\}} P_{k,j}(u^j;O) = 1$$

Equation (11) is not used for those encoders in a concatenated coded system connected to a channel. To keep the expressions general, as is seen from Eqs. (3), (4), and (12), $P_k[c(e);I]$ is not represented as a product.

Direct computation of the probability distribution of bits without first obtaining the probability distribution of symbols is presented in [9]. In the following sections, for simplicity of notation, the probability distribution of symbols rather than of bits is considered. The extension of the results to probability distributions of bits based on the above derivations is straightforward.

IV. The Sliding-Window Soft-Input Soft-Output Module (SW-SISO)

As previous description should have made clear, the SISO algorithm requires that the whole sequence has been received before starting the smoothing process. The reason is due to the backward recursion that starts from the (supposed-known) final trellis state. As a consequence, its practical application is limited to the case when the duration of the transmission is short (n small) or, for n long, when the received sequence can be segmented into independent consecutive blocks, like for block codes or convolutional codes with trellis termination. It cannot be used for continuous decoding of convolutional codes. This constraint leads to a frame rigidity imposed on the system and also reduces the overall code rate.

A more flexible decoding strategy is offered by modifying the algorithm in such a way that the SISO module operates on a fixed memory span and outputs the smoothed probability distributions after a given delay, D. We call this new algorithm the sliding-window soft-input soft-output (SW-SISO) algorithm (and module). We propose two versions of the SW-SISO that differ in the way they overcome the problem of initializing the backward recursion without waiting for the entire sequence. From now on, we assume that the time index set K is semi-infinite, i.e., $K = \{1,\ldots,\infty\}$, and that the initial state s_0 is known.

A. The First Version of the Sliding-Window SISO Algorithm (SW-SISO1)

The SW-SISO1 algorithm consists of the following steps:

(1) Initialize A_0 according to Eq. (5).

(2) Forward recursion at time k: Compute the A_k through the forward recursion of Eq. (3).

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

$$B_k^{(0)}(s) = A_k(s) \ \ \forall s \tag{13}$$

(4) Backward recursion: It is performed according to Eq. (4) from iterations $i = 1$ to $i = D$ as

$$B_{k-i}^{(i)}(s) = \sum_{e:s^S(e)=s} B_{k-i+1}^{(i-1)}[s^E(e)] P_k[u(e);I] P_k[c(e);I] \tag{14}$$

and

$$B_{k-D}(s) = B_{k-D}^{(D)}(s) \ \ \forall s \tag{15}$$

(5) The probability distributions at time $k - D$ are computed as

$$P_{k-D}(c;O) = H_c \tilde{H}_c \sum_{e:c(e)=c} A_{k-D-1}[s^S(e)] P_{k-D}[u(e);I] B_{k-D}[s^E(e)] \tag{16}$$

$$P_{k-D}(u;O) = H_c \tilde{H}_c \sum_{e:u(e)=u} A_{k-D-1}[s^S(e)] P_{k-D}[c(e);I] B_{k-D}[s^E(e)] \tag{17}$$

B. The Second Simplified Version of the Sliding-Window SISO Algorithm (SW-SISO2)

A further simplification of the sliding-window SISO algorithm, which is similar to SW-SISO1 except for the backward initial condition, that significantly reduces the memory requirements consists of the following steps:

(1) Initialize A_0 according to Eq. (5).

(2) Forward recursion at time k, $k > D$: Compute the A_{k-D} through the forward recursion

$$A_{k-D}(s) = \sum_{e:s^E(e)=s} A_{k-D-1}[s^S(e)] P_{k-D}[u(e);I] P_{k-D}[c(e);I] \ \ , k > D \tag{18}$$

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

$$B_k^{(0)}(s) = \frac{1}{N} \ \ \forall s \tag{19}$$

(4) Backward recursion (time $k > D$): It is performed according to Eq. (14) as before.

(5) The probability distributions at time $k-D$ are computed according to Eqs. (16) and (17) as before.

C. Memory and Computational Complexity

1. Algorithm SW-SISO1. For a convolutional code with parameters (k_0, n_0) and number of states N, so that $N_I = 2^{k_0}$ and $N_O = 2^{n_0}$, the algorithm SW-SISO1 requires storage of $N \times D$ values of A's and $D(N_I + N_O)$ values of the input unconstrained probabilities $P_k(u; I)$ and $P_k(c; I)$. Moreover, to update the A's and B's for each time instant, it needs to perform $2 \times N \times N_I$ multiplications and N additions of N_I numbers. To output the set of probability distributions at each time instant, we need a D-times long backward recursion. Thus, overall the computational complexity requires the following:

(1) $2(D+1) \times N \times N_I$ multiplications.

(2) $(D+1) \times N \times (N_I - 1)$ additions.

2. Algorithm SW-SISO2. This simplified version of the sliding-window SISO algorithm does not require the storage of the $N \times D$ values of A's, as they are updated with a delay of D steps. As a consequence, only N values of A's and $D(N_I + N_O)$ values of the input unconstrained probabilities $P_k(u; I)$ and $P_k(c; I)$ need to be stored. The computational complexity is the same as that for the previous version of the algorithm. However, since the initialization of the B recursion is less accurate, a larger value of D may be necessary.

V. The Additive SISO Algorithm (A-SISO)

The sliding-window SISO algorithms solve the problems of continuously updating the probability distributions, without requiring trellis terminations. Their computational complexity, however, is still high when compared to other suboptimal algorithms like SOVA. This is due mainly to the fact that they are *multiplicative* algorithms. In this section, we overcome this drawback by proposing the additive version of the SISO algorithm. Clearly, the same procedure can be applied to its two sliding-window versions, SW-SISO1 and SW-SISO2.

To convert the previous SISO algorithm from multiplicative to additive form, we exploit the monotonicity of the logarithm function, and use for the quantities $P(u; \cdot)$, $P(c; \cdot)$, A, and B their natural logarithms, according to the following definitions:

$$\pi_k(c; I) \triangleq \log[P_k(c; I)]$$

$$\pi_k(u; I) \triangleq \log[P_k(u; I)]$$

$$\pi_k(c; O) \triangleq \log[P_k(c; O)]$$

$$\pi_k(u; O) \triangleq \log[P_k(c; O)]$$

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

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

With these definitions, the SISO algorithm defined by Eqs. (7) and (8) and Eqs. (3) and (4) becomes the following: At time k, the output probability distributions are computed as

$$\pi_k(c;O) = \log\left[\sum_{e:c(e)=c} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \beta_k[s^E(e)]\}\right] + h_c \tag{20}$$

$$\pi_k(u;O) = \log\left[\sum_{e:u(e)=u} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[c(e);I] + \beta_k[s^E(e)]\}\right] + h_u \tag{21}$$

where the quantities $\alpha_k(\cdot)$ and $\beta_k(\cdot)$ are obtained through the forward and backward recursions, respectively, as

$$\alpha_k(s) = \log\left[\sum_{e:s^E(e)=s} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \pi_k[c(e);I]\}\right], k = 1,\ldots,n \tag{22}$$

$$\beta_k(s) = \log\left[\sum_{e:s^S(e)=s} \exp\{\beta_{k+1}[s^E(e)] + \pi_{k+1}[u;I] + \pi_{k+1}[c(e);I]\}\right], k = n-1,\ldots,0 \tag{23}$$

with initial values

$$\alpha_0(s) = \begin{cases} 0 & s = S_0 \\ -\infty & \text{otherwise} \end{cases}$$

$$\beta_n(S_i) = \begin{cases} 0 & s = S_n \\ -\infty & \text{otherwise} \end{cases}$$

The quantities h_c and h_u are normalization constants needed to prevent excessive growth of the numerical values of the α's and β's.

The problem in the previous recursions consists in the evaluation of the logarithm of a sum of exponentials like[3]

$$a = \log\left[\sum_{i}^{L} \exp\{a_i\}\right] \tag{24}$$

To evaluate a in Eq. (24), we can use two approximations, with increasing accuracy (and complexity). The first approximation is

$$a = \log\left[\sum_{i}^{L} \exp\{a_i\}\right] \simeq a_M \tag{25}$$

[3] The notations in this part are modified for simplicity and do not coincide with the previous ones.

where we have defined

$$a_M \triangleq \max_i a_i\,, \quad i = 1, \ldots, L$$

This approximation assumes that

$$a_M >> a_i\,, \ \ \forall a_i \neq a_M$$

It is almost optimal for medium-high signal-to-noise ratios and leads to performance degradations of the order of 0.5 to 0.7 dB for very low signal-to-noise ratios.

Using Eq. (25), the recursions of Eqs. (22) and (23) become

$$\alpha_k(s) = \max_{e:s^E(e)=s} \left\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e); I] + \pi_k[c(e); I]\right\} \quad k = 1, \ldots, n \tag{26}$$

$$\beta_k(s) = \max_{e:s^S(e)=s} \left\{\beta_{k+1}[s^E(e)] + \pi_{k+1}[u(e); I] + \pi_{k+1}[c(e); I]\right\} \quad k = n-1, \ldots, 0 \tag{27}$$

and the π's of Eqs. (20) and (21) become

$$\pi_k(c; O) = \max_{e:c(e)=c} \left\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e); I] + \beta_k[s^E(e)]\right\} + h_c \tag{28}$$

$$\pi_k(u; O) = \max_{e:u(e)=u} \left\{\alpha_{k-1}[s^S(e)] + \pi_k[c(e); I] + \beta_k[s^E(e)]\right\} + h_u \tag{29}$$

When the accuracy of the previously proposed approximation is not sufficient, we can evaluate a in Eq. (24) using the following recursive algorithm (already proposed in [26,31]):

$$a^{(1)} = a_1$$

$$a^{(l)} = \max(a^{(l-1)}, a_l) + \log[1 + \exp(-|a^{(l-1)} - a_l|)]\,, \quad l = 2, \ldots, L$$

$$a \equiv a^{(L)}$$

To evaluate a, the algorithm needs to perform $(L-1)$ times two kinds of operations: a comparison between two numbers to find the maximum, and the computation of

$$\log[1 + \exp(-\Delta)] \ \ \Delta \geq 0$$

The second operation can be implemented using a single-entry look-up table up to the desired accuracy (in [26], eight values were shown to be enough to guarantee almost ideal performance). Therefore, a in Eq. (24) can be written as $a = a_M + \delta(a_1, a_2, \ldots, a_L) \triangleq \max_i{}^*\{a_i\}$. The second term, $\delta(a_1, a_2, \ldots, a_L)$, is called the correction term and can be computed using a look-up table, as discussed above. Now, if desired, max can be replaced by max* in Eqs. (26) through (29).

Clearly, the additive form of the SISO algorithm can be applied to both versions of the sliding-window SISO algorithms described in the previous section, with straightforward modifications. In the following section, dealing with examples of application, we will use the additive form of the second (simpler) sliding-window algorithm, called the additive sliding-window SISO (ASW-SISO). An example of the additive SISO algorithm working at bit level, which can also be used for punctured codes derived from a rate 1/2 code, is given in the Appendix.

VI. Applications of the ASW-SISO Module

In this section, we provide examples of applications of the ASW-SISO module embedded into the iterative decoding schemes of the PCCCs and SCCCs previously shown in Figs. 1 and 2.

Consider, as a first example, a parallel concatenated convolutional code (turbo code or PCCC) obtained using as constituent codes two equal rate 1/2 systematic, recursive, 16-state convolutional encoders with generating matrix

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

The interleaver length is N=16,384. The overall PCCC forms a very powerful code for possible use in applications requiring reliable operation at very low signal-to-noise ratios, such as those in deep-space communications systems.

The performance of the continuous iterative decoding algorithm applied to the concatenated code, obtained by simulation using the ASW-SISO and the look-up table algorithms, is shown in Fig. 6, where we plot the bit-error probability as a function of the number of iterations of the decoding algorithm for various values of the bit signal-to-noise ratio, E_b/N_0. It can be seen that the decoding algorithm converges down to an error probability of 10^{-5} for signal-to-noise ratios of 0.2 dB with nine iterations. Moreover, convergence is guaranteed also at signal-to-noise ratios as low as 0.05 dB, which is 0.55 dB from the Shannon capacity limit.

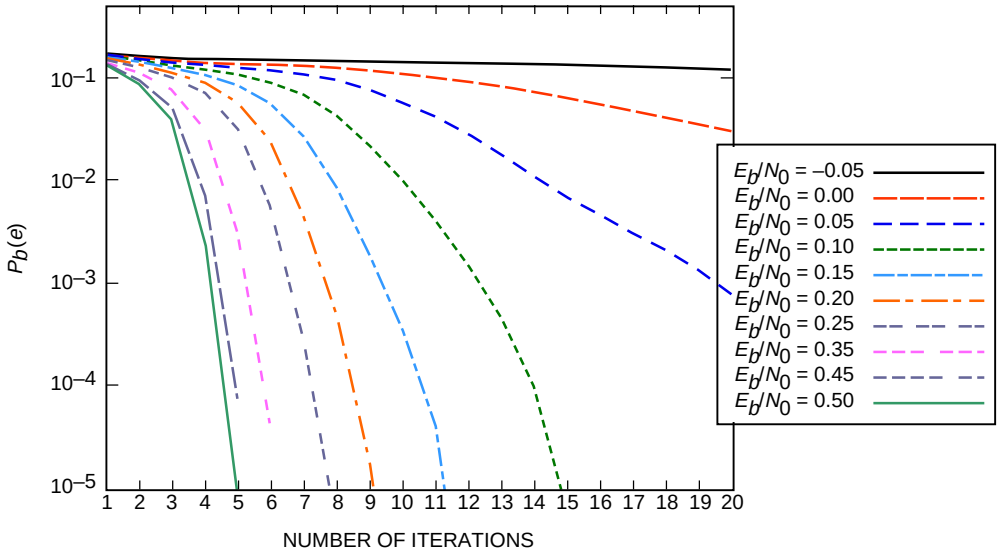

Fig. 6. The convergence of turbo-decoding: bit error probability versus the number of iterations using the ASW-SISO algorithm.

As a second example, we construct the serial concatenation of two convolutional codes (SCCCs) using as an outer code the rate 1/2, 8-state nonrecursive encoder with generating matrix

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

and, as an inner code, the rate 1/2, 8-state recursive encoder with generating matrix

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

The resulting SCCC has rate 1/4. The interleaver length has been chosen to ensure a decoding delay in terms of input information bits equal to 16,384.

The performance of the concatenated code, obtained by simulation as before, is shown in Fig. 7, where we plot the bit-error probability as a function of the number of iterations of the decoding algorithm for various values of the bit signal-to-noise ratio, E_b/N_0. It can be seen that the decoding algorithm converges down to an error probability of 10^{-5} for signal-to-noise ratios of 0.10 dB with nine iterations. Moreover, convergence also is guaranteed at signal-to-noise ratios as low as -0.10 dB, which is 0.71 dB from the capacity limit.

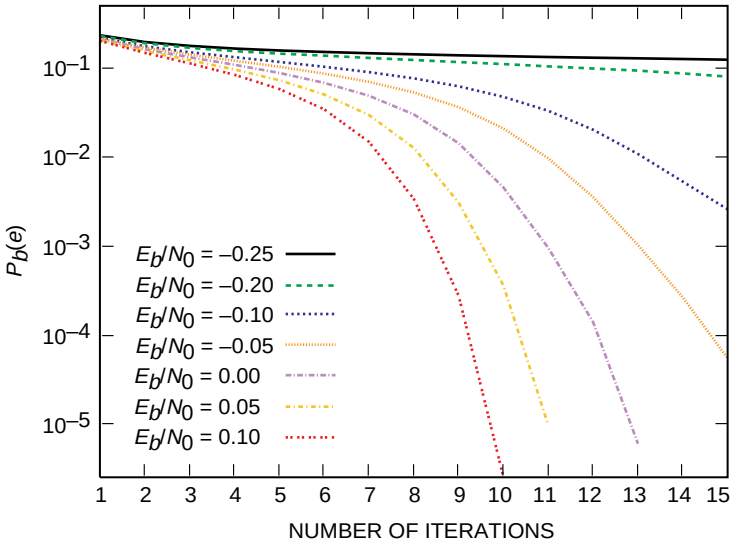

Fig. 7. Convergence of iterative decoding for a serial concatenated code: bit error rate probability versus number of iterations using the ASW-SISO algorithm.

As a third, and final, example, we compare the performance of a PCCC and an SCCC with the same rate and complexity. The concatenated code rate is 1/3, the CCs are four-state recursive encoders (rates 1/2 + 1/2 for the PCCCs and rates 1/2 + 2/3 for the SCCCs), and the decoding delays in terms of input bits are equal to 16,384. In Fig. 8, we report the bit-error probability versus the signal-to-noise ratio for six and nine decoding iterations. As the curves show, the PCCC outperforms the SCCC for high values of the bit-error probabilities. Below 10^{-5} (for nine iterations), the SCCC behaves significantly better and does not present the "floor" behavior typical of PCCCs. In particular, at 10^{-6}, the SCCC has an advantage of 0.5 dB with nine iterations.

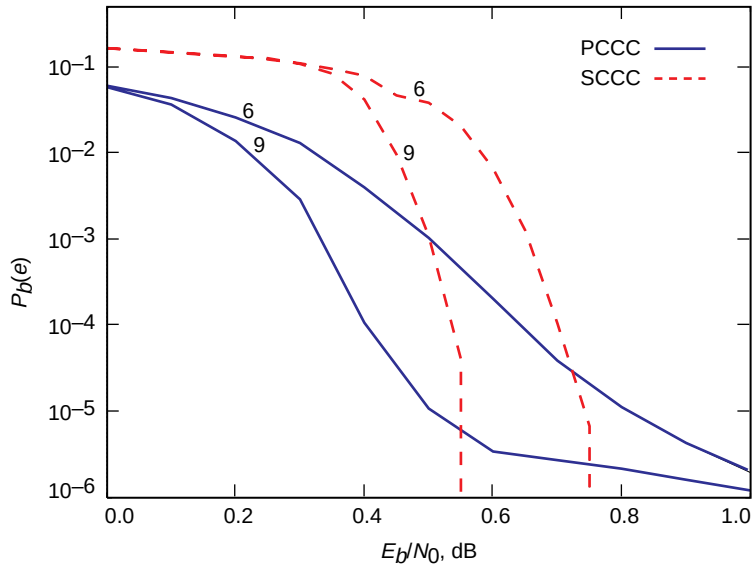

Fig. 8. Comparison of two rate 1/3 PCCC and SCCC. The curves refer to six and nine iterations of the decoding algorithm and to an equal input decoding delay of 16,384.

VII. Conclusions

Algorithms that estimate the maximum a posteriori (MAP) probabilities of the information sequence for trellis codes have been synthetically illustrated. Iterative decoding schemes for both parallel and serially concatenated codes need as a key component a module that implements the MAP algorithm. A very general module, called SISO, has been described, which works on the edges of the code trellis section and is able to cope with all possible encoders. While the optimum MAP algorithm is intrinsically block oriented, we have proposed a sliding-window modification of it that allows continuous decoding of the received stream. Some examples of application have been worked out concerning very powerful parallel and serially concatenated codes especially suitable for deep-space communications systems.

Acknowledgments

The research in this article has been partially supported by NATO under Research Grant CRG 951208. Sergio Benedetto and Guido Montorsi also acknowledge the support of the Italian National Research Council (CNR) under Progetto Finalizzato Trasporti (Prometheus) and of the Ministero dell'Universitá e della Ricerca Scientifica e Tecnologica (MURST) (Progetto 40% Comunicazioni con Mezzi Mobili).

References

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

[2] 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.

[3] S. Benedetto and G. Montorsi, "Iterative Decoding of Serially Concatenated Convolutional Codes," *Electronics Letters*, vol. 32, no. 13, pp. 1186–1188, June 1996.

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

[5] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design, and Iterative Decoding," *The Telecommunications and Data Acquisition Progress Report 42-126, April–June 1996*, Jet Propulsion Laboratory, Pasadena, California, pp. 1–26, August 15, 1996.
http://tda.jpl.nasa.gov/tda/progress_report/42-126/126D.pdf

[6] J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Transactions on Information Theory*, vol. 43, no. 2, pp. 429–445, March 1996.

[7] P. Robertson, "Illuminating the Structure of Decoders for Parallel Concatenated Recursive Systematic (Turbo) Codes," *Proceedings of Globecom'94*, San Francisco, California, pp. 1298–1303, December 1994.

[8] J. Y. Couleaud, "High Gain Coding Schemes for Space Communications," *ENSICA Final Year Report*, University of South Australia, The Levels, Australia, September 1995.
http://www.itr.unisa.edu.au/ steven/turbo/jyc.ps.gz

[9] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A Soft-Input Soft-Output MAP Module for Iterative Decoding of Concatenated Codes," to appear in *IEEE Communications Letters*, January 1997.

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

[11] R. W. Chang and J. C. Hancock, "On Receiver Structures for Channels Having Memory," *IEEE Transactions on Information Theory*, vol. IT-12, pp. 463–468, October 1966.

[12] 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*, vol. 1T-20, pp. 284–287, March 1974.

[13] P. L. McAdam, L. Welch, and C. Weber, "Map Bit Decoding of Convolutional Codes," *Abstracts of Papers, ISIT'72*, Asilomar, California, p. 91, January 1972.

[14] C. R. Hartmann and L. D. Rudolph, "An Optimum Symbol-by-Symbol Decoding Rule for Linear Codes," *IEEE Transactions on Information Theory*, vol. IT-22, pp. 514–517, September 1976.

[15] B. Vucetic and Y. Li, "A Survey of Soft-Output Algorithms," *Proceedings of ISITA'94*, Sydney, Australia, pp. 863–867 November 1994.

[16] G. D. Forney, Jr., "The Viterbi Algorithm," *IEEE Transactions on Information Theory*, vol. IT-61, no. 3, pp. 268–278, March 1973.

[17] H. Yamamoto and K. Itoh, "Viterbi Decoding Algorithm for Convolutional Codes With Repeat Request," *IEEE Transactions on Information Theory*, vol. IT-26, no. 5, pp. 540–547, September 1980.

[18] T. Hashimoto, "A List-Type Reduced-Constraint Generalization of the Viterbi Algorithm," *IEEE Transactions on Information Theory*, vol. IT-33, no. 6, pp. 866–876, November 1987.

[19] R. H. Deng and D. J. Costello, "High Rate Concatenated Coding Systems Using Bandwidth Efficient Trellis Inner Codes," *IEEE Transactions on Communications*, vol. COM-37, no. 5, pp. 420—427, May 1989.

[20] 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.

[21] T. Schaub and J. W. Modestino, "An Erasure Declaring Viterbi Decoder and Its Applications to Concatenated Coding Systems," *Proceedings of ICC'86*, Toronto, Canada, pp. 1612–1616, June 1986,.

[22] 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.

[23] P. Hoeher, "TCM on Frequency-Selective Fading Channels: A Comparison of Soft-Output Probabilistic Equalizers," *Proceedings of GLOBECOM'90*, San Diego, California, pp. 40l.4.l–40l.4.6, December 1990.

[24] U. Hansson, "Theoretical Treatment of ML Sequence Detection With a Concatenated Receiver," Technical Report 185L, School of Electrical and Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1994.

[25] 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.

[26] 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.

[27] P. Jung, "Novel Low Complexity Decoder for Turbo Codes," *Electronics Letters*, vol. 31, no. 2, pp. 86–87, January 1995.

[28] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Algorithm for Continuous Decoding of Turbo Codes," *Electronics Letters*, vol. 32, no. 4, pp. 314–315, February 1996.

[29] L. Papke, P. Robertson, and E. Villebrum, "Improved Decoding With the SOVA in a Parallel Concatenated (Turbo-Code) Scheme," *Proceedings of IEEE ICC'96*, Dallas, Texas, pp. 102–106, June 1996.

[30] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-Output Decoding Algorithms for Continuous Decoding of Parallel Concatenated Convolutional Codes," *Proceedings of IEEE ICC'96*, Dallas, Texas, pp. 112–117, June 1996.

[31] 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, October–December 1995*, Jet Propulsion Laboratory, Pasadena, California, pp. 63–87, February 15, 1996. http://tda.jpl.nasa.gov/tda/progress_report/42-124/124G.pdf

Appendix

An Example of the Additive SISO Algorithm Working at Bit Level

This appendix describes the SISO algorithm used in the example of serial concatenation of two rate 1/2 convolutional codes. Consider a rate 1/2 convolutional code. Let U_k be the input bit and $C_{1,k}$ and $C_{2,k}$ the output bits of the convolutional code at time k, taking values $\{0, 1\}$. Therefore, on the trellis edges at time k we have $u_k(e), c_{1,k}(e), c_{2,k}(e)$. In the following, for simplicity of notation, we drop the subscript k for the input and output bits. Define the reliability of a bit Z taking values $\{0, 1\}$ at time k as

$$\lambda_k[Z;\cdot] \triangleq \log \frac{P_k[Z=1;\cdot]}{P_k[Z=0;\cdot]} = \pi_k[Z=1;\cdot] - \pi_k[Z=0;\cdot]$$

The second argument in the brackets, shown by a dot, may represent I, the input, or O, the output, to the SISO. We use the following identity:

$$a = \log\left[\sum_{i=1}^{L} e^{a_i}\right] = \max_i\{a_i\} + \delta(a_1, \ldots, a_L) \triangleq \max_i{}^*\{a_i\}$$

where $\delta(a_1, \ldots, a_L)$ is the correction term, as discussed in Section V, that can be computed using a look-up table. We defined the "max*" operation as a maximization (compare–select) plus a correction term (look-up table). Using the results of Sections III.D, IV, and V, we obtain the *forward* and the *backward* recursions as

$$\alpha_k(s) = \max_{e:s^E(e)=s}{}^* \{\alpha_{k-1}\left[s^S(e)\right] + u(e)\lambda_k[U;I] + c_1(e)\lambda_k[C_1;I] + c_2(e)\lambda_k[C_2;I]\} + h_{\alpha_k}$$

$$\beta_k(s) = \max_{e:s^S(e)=s}{}^* \{\beta_{k+1}\left[s^E(e)\right] + u(e)\lambda_{k+1}[U;I] + c_1(e)\lambda_{k+1}[C_1;I] + c_2(e)\lambda_{k+1}[C_2;I]\} + h_{\beta_k}$$

with initial values $\alpha_0(s) = 0$ if $s = S_0$, and $\alpha_0(s) = -\infty$ otherwise, and $\beta_n(s) = 0$ if $s = S_n$, and $\beta_n(s) = -\infty$ otherwise, where h_{α_k} and h_{β_k} are normalization constants, which, for a hardware implementation of the SISO, are used to prevent buffer overflow. These operations are similar to those employed by the Viterbi algorithm when it is used in the forward and backward directions, except for a correction term that is added when compare–select operations are performed.

For the inner decoder, which is connected to the AWGN channel, we have $\lambda_k[C_1;I] = (2A/\sigma^2)r_{1,k}$ and $\lambda_k[C_2;I] = (2A/\sigma^2)r_{2,k}$, where $r_{i,k} = A(2c_i - 1) + n_{i,k}$, $i = 1, 2$, is the received samples at the output of the receiver matched filter, $c_i \in \{0, 1\}$, and $n_{i,k}$ is the zero-mean independent identically distributed (i.i.d.) Gaussian noise samples with variance σ^2.

The *extrinsic bit information* for U, C_1, and C_2 can be obtained as

$$\lambda_k(U;O) = \max^*_{e:u(e)=1}\{\alpha_{k-1}\left[s^S(e)\right] + c_1(e)\lambda_k[C_1;I] + c_2(e)\lambda_k[C_2;I] + \beta_k\left[s^E(e)\right]\}$$

$$- \max^*_{e:u(e)=0}\{\alpha_{k-1}\left[s^S(e)\right] + c_1(e)\lambda_k[C_1;I] + c_2(e)\lambda_k[C_2;I] + \beta_k\left[s^E(e)\right]\}$$

$$\lambda_k(C_1;O) = \max^*_{e:c_1(e)=1}\{\alpha_{k-1}\left[s^S(e)\right] + u(e)\lambda_k[U;I] + c_2(e)\lambda_k[C_2;I] + \beta_k\left[s^E(e)\right]\}$$

$$- \max^*_{e:c_1(e)=0}\{\alpha_{k-1}\left[s^S(e)\right] + u(e)\lambda_k[U;I] + c_2(e)\lambda_k[C_2;I] + \beta_k\left[s^E(e)\right]\}$$

$$\lambda_k(C_2;O) = \max^*_{e:c_2(e)=1}\{\alpha_{k-1}\left[s^S(e)\right] + u(e)\lambda_k[U;I] + c_1(e)\lambda_k[C_1;I] + \beta_k\left[s^E(e)\right]\}$$

$$- \max^*_{e:c_2(e)=0}\{\alpha_{k-1}\left[s^S(e)\right] + u(e)\lambda_k[U;I] + c_1(e)\lambda_k[C_1;I] + \beta_k\left[s^E(e)\right]\}$$

This example should also clarify the extension of the additive SISO, using bit reliabilities, to a convolutional code with code rate k_o/n_o. The circuits required to implement all of the above calculations are similar to those proposed in the Appendix of [31].