Soft-input soft-output modules for the construction and distributed iterative decoding of code networks

Soft-input soft-output modules for the construction and distributed iterative decoding of code networks

Sergio Benedetto, Dariush Divsalar, Guido Montorsi, Fabrizio Pollara

Dipartimento di Elettronica, Politecnico di Torino

This work has been supported by NATO under Research Grant CRG 951208 and by Qualcomm, Inc.
The research described in this paper was partially carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with National Aeronautics and Space Administration (NASA).

Abstract

Soft-input soft-output building blocks (modules) are presented to construct and iteratively decode in a distributed fashion *code networks*, a new concept that includes, and generalizes, various forms of concatenated coding schemes. Among the modules, a central role is played by the SISO module (and the underlying algorithm): it consists of a four-port device performing a processing of the sequences of two input probability distributions by constraining them to the code trellis structure. The SISO and other soft-input soft-output modules are employed to construct and decode a variety of code networks, including "turbo codes" and serially concatenated codes with interleavers.

Keywords

Iterative decoding, turbo codes, serial concatenated codes, soft decoding algorithms.

I. Introduction

This paper concerns the construction and the distributed, iterative decoding of a conglomerate of codes that we call *code networks*, the name stemming from the complexity and richness of the possible structures of the coding schemes so obtained that make them look like communications networks. The connection of the various encoders is made through interleavers, and can assume various topologies like tree, star, ring etc. The individual encoders that form the network can work on any finite input and output alphabets, so that we can include in it binary codes as well as trellis-coded modulation schemes.

Particular cases of these code networks are the recently introduced, highly performing *turbo codes* [1] and serially concatenated codes with interleavers [2].

The key step in this new proposal is the definition of a number of building blocks that are employed in the code network construction; each encoder building block has a well-defined soft-input soft-output counterpart, which is used at the receiving side to realize a *distributed, iterative* decoding algorithm.

The concept of distributed decoding is here emphasized, in that we do not consider the code network as an overall, unique code, and thus do not deal with its optimum maximum-likelihood decoding and consequent suboptimality of the proposed decoder. Rather, we consider the decoder structure as a multiplicity of connected modules, which exchange soft information with the objective of improving their knowledge of the a-posteriori probabilities of the quantities that flow through them. From this perspective, the decoder structure comes as a direct, natural implication of the encoder network. The distributed decoding drastically reduces the decoder complexity, yielding very powerful codes endowed with a relatively simple decoding structure.

The iterative, distributed decoding algorithms work very well in all practical situations; however, two important theoretical questions are left unsolved, concerning the "if" and "where" of

the convergence of the distributed algorithms.

Among the decoder modules, a key role is played by a block we call SISO (Soft-Input Soft-Output), which implements a soft-output algorithm performing an update of the a-posteriori probabilities of both information and coded symbols based on the code constraint. A significant part of the paper is devoted to it.

Soft-output algorithms fall within the broad framework of digital transmission systems where the received signal is a sequence of waveforms whose correlation extends well beyond T, the signaling period. There can be many reasons for this correlation, such as coding, intersymbol interference, correlated fading. It is well known [3] that the optimum receiver in such situation 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-second long, with $T_d > T$. The decision rule can be either optimum with respect to a sequence of symbols $u_k^n \triangleq (u_k, u_{k+1}, \ldots, u_{k+n-1})$, or with respect to the individual symbols u_k.

The most widely applied algorithm that realizes the optimum maximum-likelihood sequence detection is the Viterbi algorithm. Optimum symbol decision algorithms must base their decisions on the maximization of the a posteriori probability (APP). They have been known since the early seventies [4], [5], [6], [7], [8]. The algorithms in [5], [6], [7], [8] present a memory requirement and computational complexity that grow linearly with the decoding delay, and require that the whole sequence had been received before starting the decoding operations. The algorithm in [4] can work with a fixed delay, thus not requiring the reception of the entire sequence. However, its memory and computational complexity grows exponentially with the decoding delay. Recently, an APP algorithm conjugating the nice aspects of previous algorithms, i.e. a fixed delay and linear complexity growth with decoding delay has been proposed in [9]. Various modification of the algorithm in [6] have also been proposed and verified by simulation [10], [11], [12], [13], [14], [15].

Symbol-by-symbol MAP decoding has been much less popular than the Viterbi algorithm and almost never applied in practical systems until recently. The reason is that, when used for decoding a single code or to cope with intersymbol interference, the performance improvement of symbol-by-symbol MAP decoding over the Viterbi algorithm is insignificant, and certainly insufficient to justify the increase in complexity.

The story is drastically different when we consider a system using more than a single source of memory, like two or more concatenated codes, or the cascade of a code with a channel with memory. Concatenated coding schemes (a class in which we include product codes, multilevel codes, generalized concatenated codes, serial and parallel concatenated codes) have been first proposed by Forney [16] as a means to achieve 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* [17] or *iterated stage decoding* [1].

In the above cases, the burden of decoding at the receiver side is split into two or more decoders. To work properly, the decoding algorithms cannot limit themselves to pass 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 [16], 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 and on the code, the *a posteriori probability* distribution.

The Viterbi algorithm cannot do the job, and different solutions are needed. 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 informations. This has led to the concept of "augmented-output", or list-decoding Viterbi algorithm [18], and to the soft-output Viterbi algorithm (SOVA) [19], [20]. These solutions are clearly suboptimal, as they are unable to supply the required APP's. A different approach consist in revisiting the original symbol APP algorithms [4], [6], with the aim of simplifying them to a form suitable for implementation [9], [10], [11], [12], [13], [14], [15]. SOVA has a significantly lower complexity than APP algorithms, paid in poorer performance: the degradation is small for binary symbols, but becomes significant in the nonbinary case [21], [22].

Recently, the parallel concatenation of two convolutional codes fed by information sequences obtained through the interposition of a long interleaver ("turbo codes", see [1]) has been shown to yield performance close to the Shannon capacity limit at a non zero value of the bit error probability, well above the channel cutoff rate. An alternative based on the serial concatenation of interleaved codes has also been analyzed, and proved to yield even superior performance [2]. The key to the unprecedented performance is the decoding algorithm, which iterates several times the cascade of the soft-input soft-output decoders of the two constituent codes. Although some interpretations of the iterative algorithms have been proposed [23], [24], a precise understanding of it, in terms of performance proximity and convergence to the ML or symbol-by-symbol MAP decoding of the whole concatenated code, is not known yet.

The aim of this paper is two-fold. In the first part. we present a versatile soft-input soft-output (SISO) building block for several applications, like symbol-by-symbol MAP decoding of a single code, and, more important, iterative decoding of multiple parallel and serial concatenated codes with interleavers. It is based on the BCJR algorithm[1] [6], but, unlike all the previously published modifications of the original BCJR algorithm the SISO algorithm is very general, in that it:

- allows continuous decoding of the required sequence, when used to decode the concatenation

[1]It is usually referenced as the "Bahl algorithm", from the name of the first author. We prefer to credit all the authors.

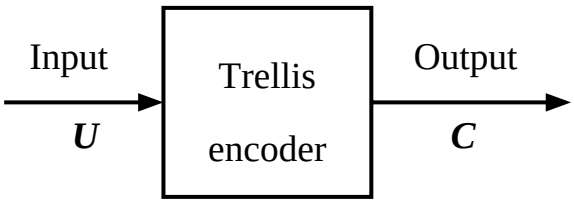


Fig. 1. The trellis encoder

of convolutional codes, without requiring the termination of the trellises of constituent codes;

- works with multilevel (not only binary) symbols;
- can be used with block and convolutional codes, both systematic and not systematic;
- can cope with codes having rates greater than one, like those encountered in some concatenated schemes;
- can accommodate parallel edges, i.e. trellises with more branches joining each pair of states, a common case for trellis coded modulation.

In the second part of the paper, we broaden the picture introducing the aforementioned building blocks for constructing and iteratively decoding encoder networks.

Each encoder building block will be shown to admit a soft-input soft-output counterpart, so that each encoder network implies by specular symmetry its iterative decoder, and the whole procedure yields a sort of "visual" justification of the decoding strategy.

Throughout the paper, several examples will be presented that refer to important practical cases; they will show that the tools presented here offer to the telecommunication engineer a great variety of design solutions with different performance/complexity trade-offs.

II. Notations and definitions

A. The encoder

The decoding algorithm underlying the behavior of SISO works for encoders represented in their trellis form. 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. 1 we show a *trellis encoder*, characterized by the following quantities[2] :

[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.

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

$$\mathcal{U} = \{u_1, \ldots, u_{N_I}\} \ .$$

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

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

where

$$P_k(u) \stackrel{\triangle}{=} \mathrm{P}[U_k = u]$$

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

$$\mathcal{C} = \{c_1, \ldots, c_{N_O}\} \ .$$

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

$$\mathbf{P}(c) = (P_k(c))_{k \in K} \ .$$

B. The trellis section

The dynamics of a time-invariant convolutional code is 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:

- 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 $s \in \mathcal{S}$.
- a set of $N \cdot N_I$ edges obtained by the Cartesian product

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

which represent all possible transitions between the trellis states.

To each edge $e \in \mathcal{E}$ the following functions are associated (see Fig. 2):

- the starting state $s^S(e)$ (the projection of e onto $\mathcal{S}$);
- the ending state $s^E(e)$;
- the input symbol $u(e)$ (the projection of e onto $\mathcal{U}$);
- the output symbol $c(e)$.

The relationship between these functions depend on the particular encoder. As an example, in the case of systematic encoders the pair $(s^S(e), c(e))$ 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.

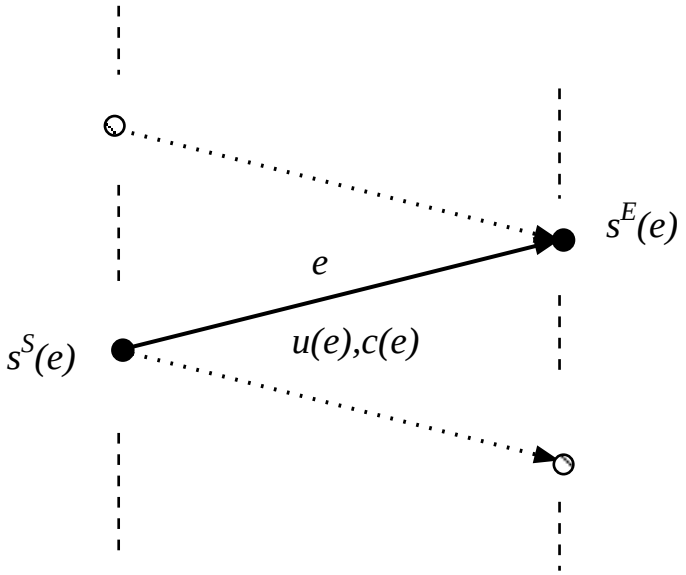


Fig. 2. An edge of the trellis section

Fig. 3. The Soft-Input Soft-Output (SISO) module

C. The SISO module

The Soft-Input Soft-Output (SISO) module is a four-port device that accepts at the input the sequences of probability distributions:

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

and outputs the sequences of probability distributions

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

computed according to its inputs and to its knowledge of the trellis section of the code.

III. The SISO algorithm

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.

First, we consider the following algorithm:

- At time k, the output probability distributions are computed as

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

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

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

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

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

with initial values:

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

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

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

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

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

From expressions (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, 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 (3).

The new probability distributions $P_k(u;O), P_k(c;O)$ represent an updated version of the input distributions $P_k(c;I), P_k(u;I)$, based on the code constraints and obtained using the probability distributions of all symbols of the sequence but the k-th ones $P_k(c;I), P_k(u;I)$. In the literature of "turbo decoding", $P_k(u;O), P_k(c;O)$ would be called *extrinsic informations*. They represent the "added value" of the SISO module to the "a priori" distributions $P_k(u;I), P_k(c;I)$. Basing the SISO algorithm on $P_k(\cdot;O)$ instead than on $\tilde{P}_k(\cdot;O)$ will simplify the block diagrams, and related software and hardware, of the iterative schemes for decoding concatenated codes. For this reason, we will consider as SISO algorithm the one expressed by (7). The SISO module is then represented as in Fig. 3.

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

A. 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 had 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 where the duration of the transmission is short (K small), or, for K 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 to 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).

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.

The SW-SISO algorithm consists of the following steps:

1. If $k = 1$ initialize A_0 according to (5).
2. Store the output and input probability distributions $P_k(c; I)$ and $P_k(u; I)$.
3. If $k < D$ skip the remaining steps.
4. Initialization of the backward recursion:

$$B_k(s) = \frac{1}{N} \quad \forall s \ . \tag{9}$$

5. Backward recursion: it is performed according to (4) for $i = k-1, \ldots, k-D+1$ as

$$B_i(s) = \sum_{e: s^S(e) = s} B_{i+1}[s^E(e)] P_{i+1}[u(e); I] P_{i+1}[c(e); I] \ \ , i = k-1, \ldots, k-D+1 \ . \tag{10}$$

6. The forward recursion and the probability distributions of the input and output symbols at time $k - D + 1$ are computed simultaneously

$$P_{k-D+1}(c; O) = \sum_{e: c(e) = c} A_{k-D}[s^S(e)] P_{k-D+1}[u(e); I] B_{k-D+1}[s^E(e)] \tag{11}$$

$$P_{k-D+1}(u; O) = \sum_{e: u(e) = u} A_{k-D}[s^S(e)] P_{k-D+1}[c(e); I] B_{k-D+1}[s^E(e)] \tag{12}$$

$$A_{k-D+1}(s) = \sum_{e: s^E(e) = s} A_{k-D}[s^S(e)] P_{k-D+1}[u(e); I] P_{k-D+1}[c(e); I] \tag{13}$$

7. Store the values of $A_{k-D+1}(s)$.

B. The sliding window SISO algorithm with grouped decisions (SWG-SISO)

In order to limit the number of recursions per decoded symbol, it may be convenient to perform the backward recursion only once every N_{bl} trellis steps, and to make a decision on a group of N_{bl} symbols at the same time. This gives rise to the SISO algorithm with grouped decisions (SWG-SISO), which consists of the following steps:

1. If $k = 1$ initialize A_0 according to (5).
2. Store the output and input probability distributions $P_k(c; I)$ and $P_k(u; I)$
3. If $k \neq D - 1 + mN_{bl}$ for some positive integer m skip the remaining steps.
4. Initialization of the backward recursion:

$$B_k(s) = \frac{1}{N} \quad \forall s \ . \tag{14}$$

5. The backward recursion from time $k-1$ to $(m-1)N_{bl}+1$ is performed according to (4) :

$$B_i(s) = \sum_{e: s^S(e) = s} B_{i+1}[s^E(e)] P_{i+1}[u(e); I] P_{i+1}[c(e); I] \ \ , , i = k-1, \ldots, m-1N_{bl}+1$$

6. From time mN_{bl} to $(m-1)N_{bl}+1$ store the obtained values of B's.
7. The forward recursion and the probability distributions of the m-th block of symbols are computed simultaneously from time $(m-1)N_{bl}+1$ to mN_{bl}:

$$P_i(u;O) = \sum_{e:u(e)=u} A_{i-1}[s^S(e)]P_i[c(e);I]B_i[s^E(e)] \quad (15)$$

$$P_i(c;O) = \sum_{e:c(e)=c} A_{i-1}[s^S(e)]P_i[u(e);I]B_i[s^E(e)] \quad (16)$$

$$A_i(s) = \sum_{e:s^E(e)=s} A_{i-1}[s^S(e)]P_i[u(e);I]P_i[c(e);I] \quad (17)$$

$$i=(m-1)N_{bl}+1,\ldots,mN_{bl}$$

8. Store the values of $A_{mN_{bl}}$.

It can be noticed that the SWG-SISO algorithm coincide with the SW-SISO when $N_{bl}=1$.

C. 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 mainly due to the fact that they are *multiplicative* algorithms. In this section, we overcome this drawback by proposing the additive version of the SISO algorithm. The same procedure can obviously be applied to its two sliding window versions SW-SISO and SWG-SISO.

To convert the previous SISO algorithm from multiplicative to additive form, we exploit the monotonicity of the logarithm function, and use for the quantities $P(u;\cdot), P(c;\cdot), A, B$ 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 equations (7),(8) and (3),(4) becomes the following:

- At time k, the output probability distributions are computed as

$$\pi_k(c;O) = \log\left[\sum_{e:c(e)=c} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \beta_k[s^E(e)]\}\right] + h_c \quad (18)$$

$$\pi_k(u;O) = \log\left[\sum_{e:u(e)=u} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[c(e);I] + \beta_k[s^E(e)]\}\right] + h_u \quad (19)$$

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

$$\alpha_k(s) = \log\left[\sum_{e:s^E(e)=s} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \pi_k[c(e);I]\}\right], k = 1,\ldots,n-1 \quad (20)$$

$$\beta_k(s) = \log\left[\sum_{e:s^S(e)=s} \exp\{\beta_{k+1}[s^E(e)] + \pi_{k+1}[u;I] + \pi_{k+1}[c(e);I]\}\right], k = n-1,\ldots,1 \quad (21)$$

with initial values:

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

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

The quantities h_c, h_u are normalization constants needed to prevent from an excessive growing of the numerical values of $\alpha's$ and $\beta's$.

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

$$a = \log\left[\sum_i^L \exp\{a_i\}\right] \ . \quad (22)$$

To evaluate a in (22), we can use two approximations, with increasing accuracy (and complexity). The first approximation is

$$a = \log\left[\sum_i^L \exp\{a_i\}\right] \simeq a_M \ , \quad (23)$$

where we have defined

$$a_M \stackrel{\triangle}{=} \max_i a_i \ , \quad i = 1,\ldots,L \ .$$

This approximation assumes that

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

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

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

Using (23), the recursions (20) and (21) become

$$\alpha_k(s) = \max_{e:s^E(e)=s} \left\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \pi_k[c(e);I]\right\} \quad k = 1,\ldots,n-1 \tag{24}$$

$$\beta_k(s) = \max_{e:s^S(e)=s} \left\{\beta_{k+1}[s^E(e)] + \pi_{k+1}[u(e);I] + \pi_{k+1}[c(e);I]\right\} \quad k = n-1,\ldots,1\ , \tag{25}$$

and the $\pi's$ of (18) and (19):

$$\pi_k(c;O) = \max_{e:c(e)=c} \{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \beta_k[s^E(e)]\} + h_c \tag{26}$$

$$\pi_k(u;O) = \max_{e:u(e)=u} \{\alpha_{k-1}[s^S(e)] + \pi_k[c(e);I] + \beta_k[s^E(e)]\} + h_u\ . \tag{27}$$

When the accuracy of the previously proposed approximation is not sufficient, we can evaluate a in (22) using the following recursive algorithm (already proposed in [11], [25]):

$$\begin{aligned} a^{(1)} &= a_1 \\ a^{(l)} &= \max(a^{(l-1)}, a_l) + \log[1+\exp(-|a^{(l-1)} - a_l|)]\ , \quad l = 2,\ldots,L \\ a &\equiv a_1^{(L)}\ . \end{aligned} \tag{28}$$

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

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

The second operation can be implemented using a single-entry look-up table up to the desired accuracy (in [11] 8 values were shown to be enough to guarantee almost ideal performance).

The additive form of the SISO algorithm can obviously be applied to both versions of the sliding window SISO algorithms described in the previous section, with straightforward modifications. In the section of applications, we will use the additive form of the second (simpler) sliding-window algorithm, denoted by additive, sliding-window SISO with grouped decisions (ASWG-SISO).

D. The SISO module as a MAP decoder

Consider the transmission system shown in Fig. 4. A source generates a sequence of n symbols U_k with a constant a-priori distribution $p(u) = \mathrm{P}[U_k = u]$. These symbols are encoded by a trellis encoder that starts at time $k = 0$ in the state $S_0 = s_1$ and generates a sequence of n output symbols C_k ending in the final state $S_n = s_n$ that is supposed to be known at the receiver.

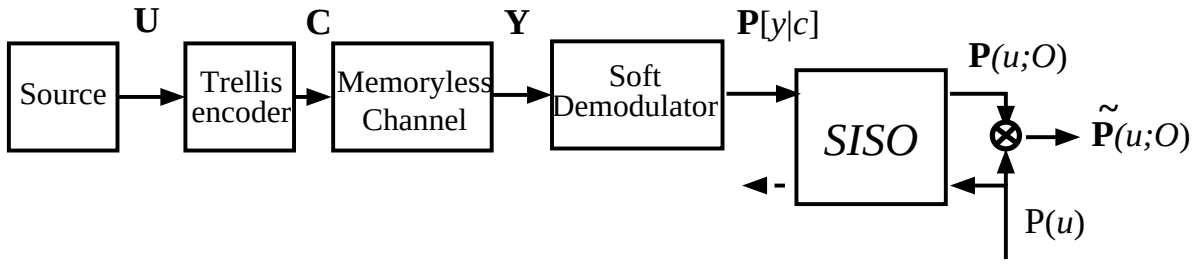


Fig. 4. A trellis-coded transmission system with the SISO module as a MAP decoder

The memoryless channel maps the transmitted symbols c_k into the the received symbols y_k, according to a known conditional probability density function (pdf)

$$p(y|c) \stackrel{\triangle}{=} \mathrm{P}[Y_k = y|C_k = c] \ .$$

For each received symbol y_k, the soft demodulator evaluates the set of probabilities $\mathrm{P}[Y_k = y_k|C_k = c]$ according to its knowledge of the channel pdf $p(y|c)$.

Our objective is to prove that the SISO module, used as in Fig. 4 so that its inputs are defined as follow:

$$\begin{aligned} P_k(u;I) &\propto \mathrm{P}[U_k = u] = P(u) \\ P_k(c;I) &\propto \mathrm{P}[y_k|C_k = c] \end{aligned}$$

permits to obtain at its output the following functions:

$$P_k(u;O) \propto \mathrm{P}[y_1^n|U_k = u] \ ,$$

$$\tilde{P}_k(u;O) \propto \mathrm{P}[y_1^n, U_k = u] \ ,$$

where y_1^n is a synthetic notation for the sequence $(Y_1 = y_1, \ldots, Y_n = y_n)$. As a consequence, its outputs can be used to perform a symbol-by-symbol MAP decision on the source symbols.

In order to compute $\tilde{P}_k(u;O)$ consider the joint probability that a given edge $e = (s^S(e), u(e))$ occurs in the trellis at time k and that the received sequence is y_1^n; it is given by

$$\begin{aligned} \mathrm{P}[E_k = e, y_1^n] &= \mathrm{P}[S_{k-1} = s^S(e), U_k = u(e), y_1^{k-1}, y_k, y_{k+1}^n] \\ &= \mathrm{P}[S_{k-1} = s^S(e), y_1^{k-1}]\, \mathrm{P}[U_k = u(e), y_k, y_{k+1}^n|S_{k-1} = s^S(e)] \\ &= \mathrm{P}[S_{k-1} = s^S(e), y_1^{k-1}]\, \mathrm{P}[U_k = u(e), y_k|S_{k-1} = s^S(e)] \\ &\quad \mathrm{P}[y_{k+1}^n|S_{k-1} = s^S(e), U_k = u(e)] \ . \end{aligned}$$

where $\mathrm{P}[y_{k+1}^n|S_{k-1}=s^S(e),U_k=u(e)]=\mathrm{P}[y_{k+1}^n|S_k=s^E(e)]$. We used the fact that given a state at time k, the future events (after the time k) are independent of the past events (before the time k). Also based on the same fact we have

$$\begin{aligned}\mathrm{P}[U_k=u(e),y_k|S_{k-1}=s^S(e)] &= \mathrm{P}[y_k|S_{k-1}=s^S(e),U_k=u(e)]\\ &\times\ \mathrm{P}[U_k=u(e)|S_{k-1}=s^S(e)]\\ &= \mathrm{P}[y_k|C_k=c(e)]\mathrm{P}[U_k=u(e)]\,.\end{aligned}$$

Let us define

$$\begin{aligned}A_k(s) &= \mathrm{P}[S_k=s,y_1^k]\\ B_k(s) &= \mathrm{P}[y_{k+1}^n|S_k=s]\,.\end{aligned}$$

The $A_k(s)$ is related to the probability of state at time k given the past observations, and $B_k(s)$ is related to the probability of state at time k given the future observations. Then we can obtain the following forward recursion for computation of $A_k(s)$:

$$\begin{aligned}A_k(s) &= \mathrm{P}[S_k=s,y_1^k]\\ &= \sum_{e:s^E(e)=s}\mathrm{P}[S_{k-1}=s^S(e),U_k=u(e),y_1^k]\\ &= \sum_{e:s^E(e)=s}\mathrm{P}[S_{k-1}=s^S(e),y_1^{k-1}]\mathrm{P}[U_k=u(e),y_k|S_{k-1}=s^S(e)] \qquad (29)\\ &= \sum_{e:s^E(e)=s}A_{k-1}[s^S(e)]\mathrm{P}[y_k|C_k=c(e)]\mathrm{P}[U_k=u(e)]\ ,\ k=1,\ldots,n\,.\end{aligned}$$

Similarly we can obtain the following backward recursion for computation of $B_k(s)$.

$$\begin{aligned}B_k(s) &= \mathrm{P}[y_{k+1}^n|S_k=s]\\ &= \sum_{e:s^S(e)=s}\mathrm{P}[U_{k+1}=u(e),y_{k+1}^n|S_k=s^S(e)]\\ &= \sum_{e:s^S(e)=s}\mathrm{P}[y_{k+2}^n|S_k=s^S(e),U_{k+1}=u(e)]\mathrm{P}[U_{k+1}=u(e),y_{k+1}|S_k=s^S(e)] \qquad (30)\\ &= \sum_{e:s^S(e)=s}B_{k+1}[s^E(e)]\mathrm{P}[y_{k+1}|C_{k+1}=c(e)]\mathrm{P}[U_{k+1}=u(e)]\ ,\ k=n-1,\ldots,1\,.\end{aligned}$$

The recursions (29) and (30) are equal to equations (5) and (7) of [6].

Using definition of $A_k(s)$ and $B_k(s)$ we obtain

$$\mathrm{P}[E_k = e, y_1^n] \;=\; A_{k-1}[s^S(e)]\,\mathrm{P}[y_k|C_k = c(e)]\,\mathrm{P}[U_k = u(e)]\,B_k[s^E(e)]\,. \tag{31}$$

We call (31) the *Key Equation*. To obtain the a-posteriori probabilities used in the MAP decision on the input symbols, we need to sum (31) over the proper subset of edges:

$$\begin{aligned}\mathrm{P}[U_k = u, y_1^n] &= \sum_{e:u(e)=u} \mathrm{P}[E_k = e, y_1^n] \\ &= \sum_{e:u(e)=u} A_{k-1}[s^S(e)]\,B_k[s^E(e)]\,\mathrm{P}[y_k|C_k = c(e)]\,\mathrm{P}[U_k = u(e)]\,. \end{aligned}\tag{32}$$

The RHS of (32) corresponds to the expression (1) of $\tilde{P}_k(u;O)$, and thus we have proved that the SISO output yields the required APP's.

IV. Construction and iterative decoding of code networks

In this section, we will introduce six building blocks (the most important one is the SISO module previously described) to construct *networks* of codes, and the corresponding soft-input soft-output blocks to be employed in their iterative decoding. The construction is very general, and encompasses multiple parallel concatenated codes (in the case of two convolutional codes, they are known in the literature as "turbo codes"), multiple serially concatenated codes (analyzed in [26]), and more complex concatenations.

We will show that, no matter how complicated is the code network, an iterative decoding scheme can be immediately devised based on the network structure. Although suboptimum, the iterative decoding yields remarkable performance, so that the designer is provided with a great range of trade-offs between performance and complexity to choose from.

To keep the rather abstract description well grounded to the earth of practice, we will include within the exposition several examples, whose aim is to show the potential of this new approach to code design.

A. The code network building blocks

To build a general code network, we need the six building blocks of Fig. 5 (note that two blocks have been put together in the figure, i.e. the parallel to serial and serial to parallel converters). They are:

1. The *trellis Encoder* E, already described in Subsection II-A.
2. The *Interleaver* I: It provides at its output a sequence $\mathbf{Y}$ that is a permuted version of the input sequence $\mathbf{X}$.
3. The *Mapper*, which maps the sequence $\mathbf{X} = (\mathbf{X}_1, \ldots, \mathbf{X}_m)$ whose symbols belong to the alphabets $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_m$ into the sequence $\mathbf{Y} = (\mathbf{Y}_1, \ldots, \mathbf{Y}_n)$ with symbols belonging

to the alphabets $\mathcal{Y} = \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_n$. The mapping is performed according to the memoryless function:

$$\mathbf{y}(\mathbf{x}) = \begin{cases} y_1(\mathbf{x}) & = & y_1(x_1, \ldots, x_m) \\ & \vdots & \\ y_n(\mathbf{x}) & = & y_n(x_1, \ldots, x_m) \end{cases}$$

In a classical system, the Mapper can correspond, with $m = 1$, to the mapper that precedes the demodulator, which, in turn, maps the multiplexed symbols into the signal waveforms on a one-to-one basis. As an example, the $n = 3$ $\mathbf{X}_i$ can represent three binary sequences with symbols belonging to Z_2, that are mapped into $m = 1$ symbol $\mathbf{Y}_1$ belonging to the signals drawn from an 8-PSK constellation. However, we do not assume the mapping to be one-to-one so that in general the inverse mapping may be not defined. For classical encoders, the one-to-one relationship between input and output is mandatory for unique decoding; in code networks, on the other hand, this constraint must be satisfied by the network as a whole, but not necessarily by all its constituent modules.

4. The *Parallel-to-Serial* P/S module takes n input sequences belonging to the same alphabet $\mathcal{X}$ and converts them into a unique sequence concatenating all the inputs. Symbolically, we can write:

$$(\mathcal{X}^n)^Z \stackrel{\text{P/S}}{\rightarrow} (\mathcal{X})^{nZ}$$

so that the output symbol rate is n times greater than the input symbol rate.

5. The *Serial-to-Parallel* S/P module takes an input sequence with symbols belonging to the alphabet $\mathcal{X}$ and splits it into n output sequences belonging to the same alphabet. Symbolically, we can write:

$$(\mathcal{X})^Z \stackrel{\text{S/P}}{\rightarrow} (\mathcal{X}^n)^{Z/n} \quad ,$$

so that the output symbol rate is n times smaller than the input symbol rate.

6. The *BroadCaster* BC takes the input sequence $\mathbf{X}$ and replicates it into the sequences $\mathbf{Y_1} = \ldots = \mathbf{Y_n} = \mathbf{X}$. [4]

B. The iterative decoder building blocks

To each of the building blocks previously described, there corresponds a module to be used in the iterative decoder. With reference to Fig. 5,[5] we describe in this section the Input-Output relationships of the modules, that can be employed in a decoding network.

[4]This blocks is actually a special case of the mapper when $n = 1$, $m = 1$ and $y_1(x) = \cdots = y_n(x) = x$. We prefer to keep it distinct to simplify the representation of the code networks.

[5]In the figure, we use shorthand notations for simplicity: the symbol $P(\cdot)$ means $P(\cdot; I)$ for the arrow pointing inside a block, and $P(\cdot; O)$ for the arrow pointing outside.

1. The SISO module, already described in Subsection II-C.
2. The *Interleaver/Deinterleaver* $\mathrm{I}/\mathrm{I}^{-1}$. It reorders the time axis k so that the input-output relationships are time-dependent, as

$$P_k(x = q; O) = P_{I(k)}(y = q; I) \tag{33}$$

$$P_k(y = q; O) = P_{I^{-1}(k)}(x = q; I)\,. \tag{34}$$

From the equations, we notice that the two sections I and I^{-1} act independently.

3. The *Soft Mapper* (SOMAP). It evaluates the output pdf over the sets $\mathcal{X}_i$, $i = 1, \ldots, m$ and $\mathcal{Y}_j$, $j = 1, \ldots, n$ through its knowledge of the input pdf defined on the same sets assumed to be independent and the mapping $\mathbf{y} = \mathbf{y}(\mathbf{x})$.

The output probability distributions are computed as follows

$$P(y_j = q; O) = \sum_{\mathbf{x}: y_j(\mathbf{x}) = q} \prod_{\substack{l=1 \\ l \neq j}}^{n} P(y_l(\mathbf{x}); I) \prod_{k=1}^{m} P(x_k; I) \tag{35}$$

$$P(x_i = q; O) = \sum_{\mathbf{x}: x_i = q} \prod_{l=1}^{n} P(y_l(\mathbf{x}); I) \prod_{\substack{k=1 \\ k \neq i}}^{m} P(x_k; I)\,. \tag{36}$$

$$\tag{37}$$

Their expressions will be explained in the following example.

`Example 1`

Consider the system of Fig. 6, where m parallel sequences of symbols $\mathbf{X}$ are mapped by the mapper into n output sequences $\mathbf{Y}$ that are transmitted independently on n memoryless channels characterized by their conditional probabilities. The soft demodulator observes the channel output sequences $\mathbf{Z}$ and generates the sequences of conditional probabilities $\mathbf{P}(z|y)$.

Insert now the module SOMAP after the soft demodulator, with input probabilities

$$\begin{aligned} P(x_i; I) &= P(x_i) \\ P(y_j; I) &= P_k(z_j|y_j) = P[Z_k = z_j | Y_j = y_j]\ . \end{aligned}$$

where the first equality derives from some a-priori information about the information symbols, and the second uses directly the outputs from the soft demodulator.

We want to prove that the the products of outputs from and inputs to the SOMAP module yield the a posteriori probabilities for both information symbols x_i and transmitted symbols y_j, i.e.

$$\begin{aligned} P(x_i; I)P(x_i; O) &= P(x_i, \mathbf{z}) \\ P(y_j; I)P(y_j; O) &= P(y_j, \mathbf{z})\ . \end{aligned}$$

The first can be used, as an example, to perform the symbol-by-symbol maximum a posteriori detection.

From Bayes rule we obtain

$$P(\mathbf{z}|x_i) = \sum_{\mathbf{x}\in\mathcal{X}} P(\mathbf{z}|\mathbf{x}, x_i) \cdot P(\mathbf{x}|x_i)$$

Since, given $\mathbf{x}$, $\mathbf{z}$ does not depend on x_i we have

$$P(\mathbf{z}|x_i) = \sum_{\mathbf{x}\in\mathcal{X}} P(\mathbf{z}|\mathbf{y}(\mathbf{x})) \cdot P(\mathbf{x}|x_i)$$

From the statistical independence of the noise samples we obtain

$$P(\mathbf{z}|x_i) = \sum_{\mathbf{x}\in\mathcal{X}} \prod_{l=1}^{n} P(z_l|y_l(\mathbf{x})) \cdot P(\mathbf{x}|x_i)$$

Considering the last term of the RHS of the last equation

$$P(\mathbf{x}|x_i) = \begin{cases} \prod_{\substack{k=1\\k\neq i}}^{m} P(x_k) & , x_i(\mathbf{x}) = x_i \\ 0 & , x_i(\mathbf{x}) \neq x_i \end{cases}$$

we finally obtain

$$P(\mathbf{z}|x_i) = \sum_{\substack{\mathbf{x}\in\mathcal{X}\\ x_i(\mathbf{x})=x_i}} \prod_{l=1}^{n} P(z_l|y_l(\mathbf{x})) \prod_{\substack{k=1\\k\neq i}}^{m} P(x_k)$$

In the same way

$$\begin{aligned} P(y_j, \mathbf{z}) &= \sum_{\mathbf{x}\in\mathcal{X}} P(y_j, \mathbf{z}|\mathbf{x}) P(\mathbf{x}) \\ &= \sum_{\mathbf{x}\in\mathcal{X}} P(y_j, \mathbf{z}|\mathbf{x}) \prod_{k=1}^{m} P(x_k) \end{aligned}$$

Consider now the first term of the RHS of the last equation

$$P(y_j, \mathbf{z}|\mathbf{x}) = \begin{cases} P(\mathbf{z}|\mathbf{y}(\mathbf{x})) & y_j(\mathbf{x}) = y_j \\ 0 & , y_j(\mathbf{x}) \neq y_j \end{cases}$$

we finally get

$$P(y_j, \mathbf{z}) = \sum_{\substack{\mathbf{x}\in\mathcal{X}\\ y_j(\mathbf{x})=y_j}} \prod_{l=1}^{n} P(z_l|y_l(\mathbf{x})) \prod_{k=1}^{m} P(x_k)$$

◇

4. The *Soft-Output Serial-to-Parallel* (SOSP). The output probability distributions are computed as follows:

$$\begin{aligned} P_k(x_i; O) &= P_{kn+i}(y; I) \;\; i = 1, \ldots, n \\ P_{kn+i}(y; O) &= P_k(x_i; I) \end{aligned}$$

Also in this case, being the SOSP a bidirectional device, it is the inverse of both the S/P and P/S modules. As for the interleaver, each output depends only on the corresponding input, so that the SOSP module can be seen as two separate modules working independently.

5. The *Soft-Output BroadCaster* (SOBC). It is a device with $n+1$ inputs and outputs. A given output does not use the information from the corresponding input, in order to prevent from undesired information feedback loops. The SOBC operates according to the following input-output relationships:

$$\begin{aligned} P(x;O) &= \prod_i P(y_i;I) \\ P(y_j;O) &= P(x;I)\prod_{i\neq j} P(y_i;I)\ . \end{aligned}$$

All previous input-output relationships can be rewritten using the additive logarithmic approach; in software and hardware implementation of the algorithms, this solution may be more efficient.

V. Decoder complexity

The decoding algorithm works in a distributed, iterative fashion. Its complexity depends on the number of soft-input soft-output blocks employed, and on the number of iterations. The second aspect (number of iterations) will be discussed later. We concentrate now on the complexity of each soft module, defined in terms of computation and memory requirements. We will refer to the multiplicative versions of each block, so that the computational complexity will be measured as the number of multiplications and additions per decoded symbol. Extension to the additive versions is straightforward, and simply requires the substitution of multiplications by additions and of additions by the operation performed in (28).

A. Algorithm SW-SISO

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-SISO requires to store the N values of $\alpha's$ and $D(N_I + N_O)$ values for the input unconstrained probabilities $P_k(u;I), P_k(c;I)$.

Moreover, to update the A's and B's for each time instant, it needs to perform $2 \times N \cdot N_I$ multiplications and $2 \times N \cdot N_I$ additions. To output the set of probability distributions at each time instant, we need a $D-1$-times long backward recursion. Thus the computational complexity requires overall:

- $2D \cdot N \cdot N_I$ multiplications
- $D \cdot N \cdot N_I$ additions.

In Table I we report the previous values, together with those referring to the classical Viterbi algorithm, for the sake of comparison. With respect to it, the SW-SISO algorithm requires a

Block name	Delay D	Computations (per decoded symbol) Products	Sums	Memory requirements
Viterbi Alg.	5ν	$2NN_I$	NN_I	DN
SW-SISO	5ν	$D \cdot 2NN_I$	$D \cdot NN_I$	$D \cdot (N_I + N_O) + N$
SWG-SISO	$N_{bl} + 5\nu - 1$	$\frac{(D-1+N_{bl})}{N_{bl}} \cdot 2NN_I$	$\frac{(D-1+N_{bl})}{N_{bl}} \cdot NN_I$	$D \cdot (N_I + N_O) + N_{bl}N$
$\mathrm{I/I}^{-1}$	N	0	0	$2NN_X$
SOMAP	0	$(n+m)N_X$	$(n+m)N_X$	0
SOSP-SOPS	n	0	0	n
SOBC	0	$(n+1)N_X$	$(n+1)N_X$	0

TABLE I

DELAY AND COMPLEXITY OF THE SOFT-INPUT SOFT-OUTPUT MODULES

number of computations D times greater, and less memory.

B. Algorithm SWG-SISO

To compute the APPs of a block of N_{bl} symbols, the SWG-SISO algorithm requires a $D-1$ long backward recursion and an N_{bl} long forward recursion. Thus the computational complexity requires overall:

- $\frac{(D-1+N_{bl})}{N_{bl}} \cdot 2NN_I$ multiplications
- $\frac{(D-1+N_{bl})}{N_{bl}} \cdot 2NN_I$ sums.

and a memory requirement equal to $D(N_I+N_O)+N_{bl}N$. These values are reported in Table I. A comparison with the Viterbi and the SW-SISO algorithms shows that, for large N_{bl}, and keeping constant the relationship $D = N_{bl} + 5\nu - 1$ between D and N_{bl}, the complexity of the SWG-SISO algorithm tends to be only twice that of the Viterbi algorithm, with essentially the same memory requirements. Also, the SWG-SISO algorithm shows, for large N_{bl}, a reduction by a factor D of complexity with respect to the SW-SISO.

The complexity analysis of the other soft modules is straightforward, and is left to the reader.

In Table I we summarize the delay and implementation complexity of each decoding soft module as a function of their characteristic parameters. The implementation complexity is measured in terms of number of operations per decoded symbols and in terms of memory requirements.

It is important to mention that the estimate of the implementation complexity reported in the Table I does not consider some important implementation issues that are summarized in the following list:

- It is possible to use probability density functions normalized with respect to one of the symbols, so that the normalized probability of that symbol is by definition 1 and does not need to be stored. As a consequence, for the binary case, a probability density function can be represented using a single likelihood ratio.
- When a symbol set is defined as the cartesian product of different sets:

$$\mathcal{X} = (\mathcal{X}_1, \ldots, \mathcal{X}_n) \ ,$$

and the pdf over the component sets are given, the probability of a symbol x belonging to $\mathcal{X}$ can be computed as follows:

$$P(x) = \prod_{i=1}^{n} P(x_i) \tag{38}$$

So that instead of storing a single pdf over

$$|\mathcal{X}| = |\mathcal{X}_1| \cdots |\mathcal{X}_n|$$

symbols, it is sufficient to store the n pdfs over the components sets that requires only

$$|\mathcal{X}_1| + \cdots + |\mathcal{X}_n|$$

storing units.

On the other hand, this solution increases the computational requirements as (38) must be computed each time one needs the probability of a given symbol belonging to $|X|$.

As an example, using a rate k/n binary convolutional encoder the input and output set are respectively k-tuples and n-tuples of bits. Use of the previous simplification (i.e. storing the bit likelihood ratios instead of symbol pdfs) leads for the SISO module to the implementation requirements of Table II.

- A given interleaver can be realized with a minimum amount of delay and memory according to suitable design rules [27]. The values reported in Table I refer to an interleaver realized using a random access memory with memory N_{int}, in which the inputs are written following the natural ascending order and the outputs are read following a permutation law of period N_{int}.

VI. Examples and simulation results

The following examples should provide more insight into the procedure of constructing and decoding code networks. In the code description and figure drawing, we will always omit the delays involved: they are required for a practical implementation, but bring an unnecessary heaviness to the otherwise neat block diagrams.

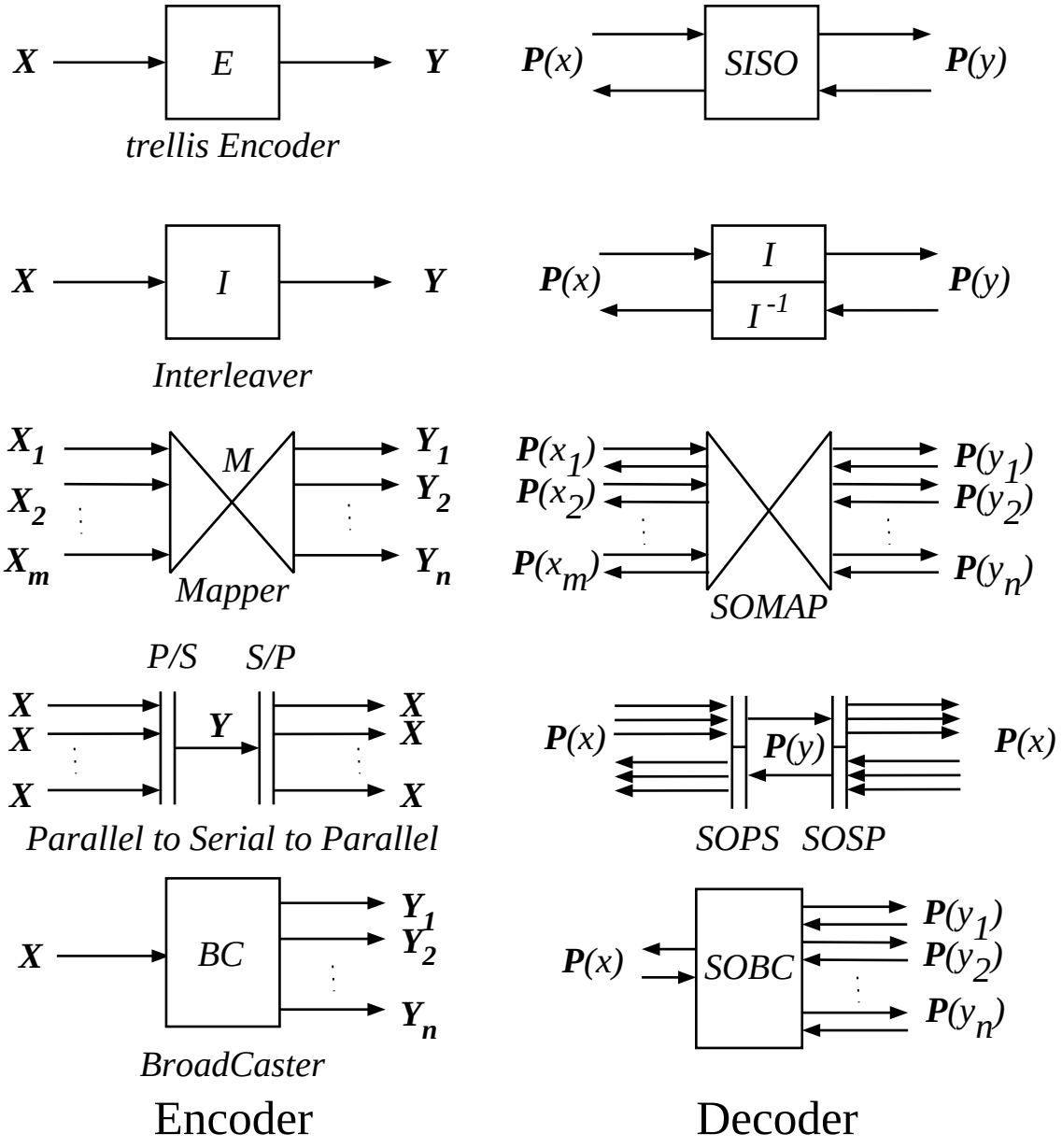


Fig. 5. The modules of a code network.

Block name	Delay D	Computations (per decoded symbol) Products	sums	Memory requirements
SISO (Binary case)	5ν	$(k+n)D \cdot NN_I$	$D \cdot NN_I$	$2D(k+n)+N$

TABLE II

IMPLEMENTATION REQUIREMENTS FOR THE SISO MODULE RELATIVE TO A k/n ENCODER USING THE BIT-LEVEL STORAGE

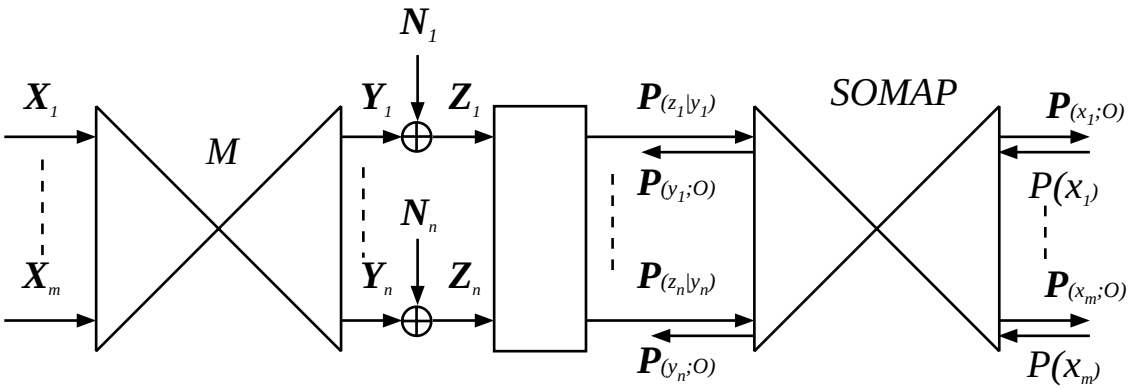


Fig. 6. A transmission system system employing mapper and soft-mapper.

A. A parallel concatenated code with 8-PSK modulation

Consider, as a first example, a parallel convolutional concatenated code (turbo code, or PCCC) obtained as in Fig. 7, where the input binary stream is replicated into three separate flows: the first is left unchanged, the second is encoded by a rate 1 recursive, 4-state convolutional encoder (E_1 in Fig. 7) and the third is permuted by the interleaver I with length N_{int} and then encoded by the convolutional encoder $E_2 = E_1$.

The three bits at the output are then mapped by the Mapper according to the Gray coding and transmitted over an AWGN channel using an 8-PSK modulation, obtaining a bandwidth efficiency of 1 bit/s/Hz. In practice, this is a trellis-coded modulation scheme. Overall, the code network requires two trellis encoders, two broadcasters, one interleaver and one mapper,

The received signal enters the soft-demodulator, which provides the conditional probabilities

$$\mathrm{P}[Y_k = y_k | M_k = m]$$

for all the PSK signals, according to its knowledge of the channel pdf $p(y|m)$. These quantities are fed to the decoding network (shown in Fig. 7) that is constructed specularly from the encoding network replacing each coding module with the corresponding decoder module of Fig. 5.

Two observations are pertinent here:

1. The inputs to the decoding network are the outputs of the coding network and vice-versa.
2. Each module of the decoding network is a bidirectional device so that in each section there are two opposite flows of informations. This feature of the decoding network originates feedbacks in the scheme that are responsible of the "turbo" nickname [1].

In a practical implementation, we have two possible solutions to build the iterative scheme. The first corresponds to the one shown in Fig. 7, in which every blocks performs the required number of iterations successively: this solution requires that the processing speed of the implementation be sufficiently higher than the transmission speed, and that the encoder state is completely

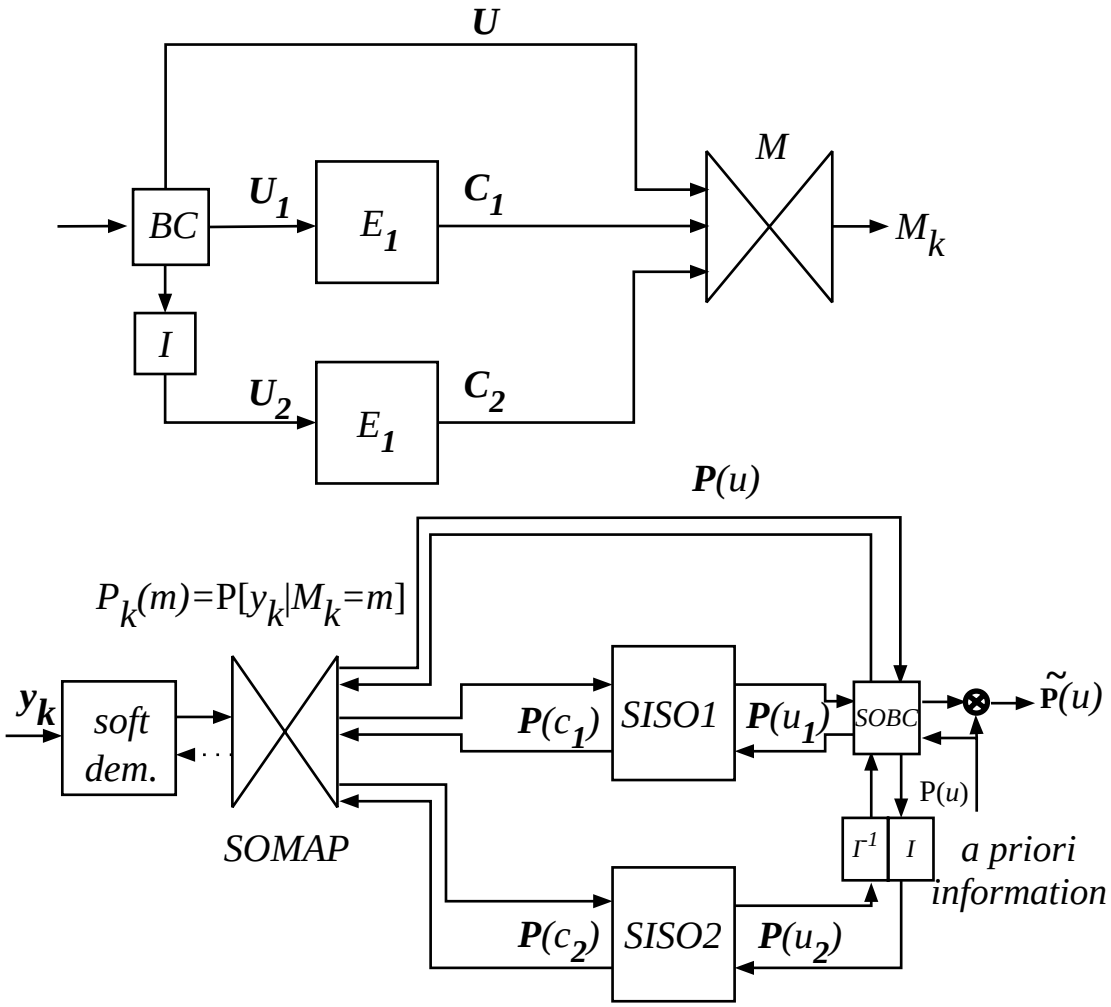


Fig. 7. The parallel concatenated convolutional coding scheme and its decoding network.

known at the receiver periodically (trellis termination and block-oriented transmission). The second solution splits each block in n_i (n_i being the number of iterations) blocks that work in a cascaded fashion, like in Fig. 8. In Fig. 8, we have drawn separately the first stage and the second (equal to all successive stages) of the cascaded implementation. The dashed lines at the outputs of the blocks mean that the corresponding quantities are passed to the next stage, whereas the dashed lines entering the delays symbols in the second (successive) stages represent quantities arriving from the first (previous) stage. The correspondence between origin and destination of dashed lines is obvious; an example is represented by the "A" symbols in the figure. The delay symbols represent the physical delays that must be applied to the various quantities for a proper operation of the iterative decoder. This implementation does not require trellis termination and

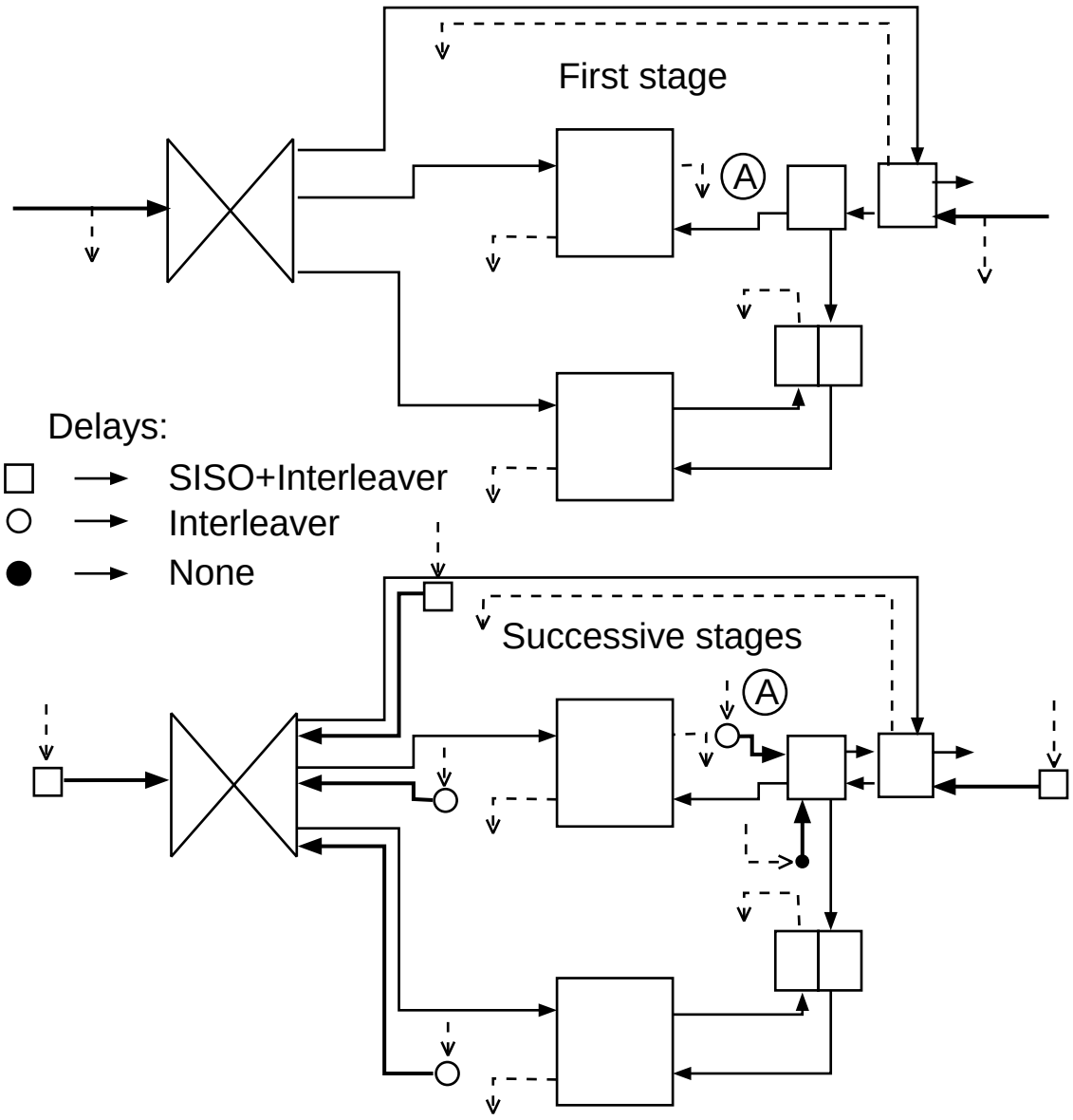


Fig. 8. The decoding network of the parallel concatenated coding scheme in a cascaded implementation.

works at the same speed of the incoming data.

We have simulated the PCCC co-decoding scheme of Fig. 7, for an interleaver of length $N_{int} = 1024$ using the ASWG-SISO and the look-up table algorithms. To show the importance of the feedback from the SISO modules to the SOMAP module, we have implemented two versions of the demodulator, one without the feedback (as it has been done so far in the literature), and the second with the feedback. The results are shown in Fig. 9, where we report the bit error probability versus the signal-to-noise ratio for three values of the number of iterations of the

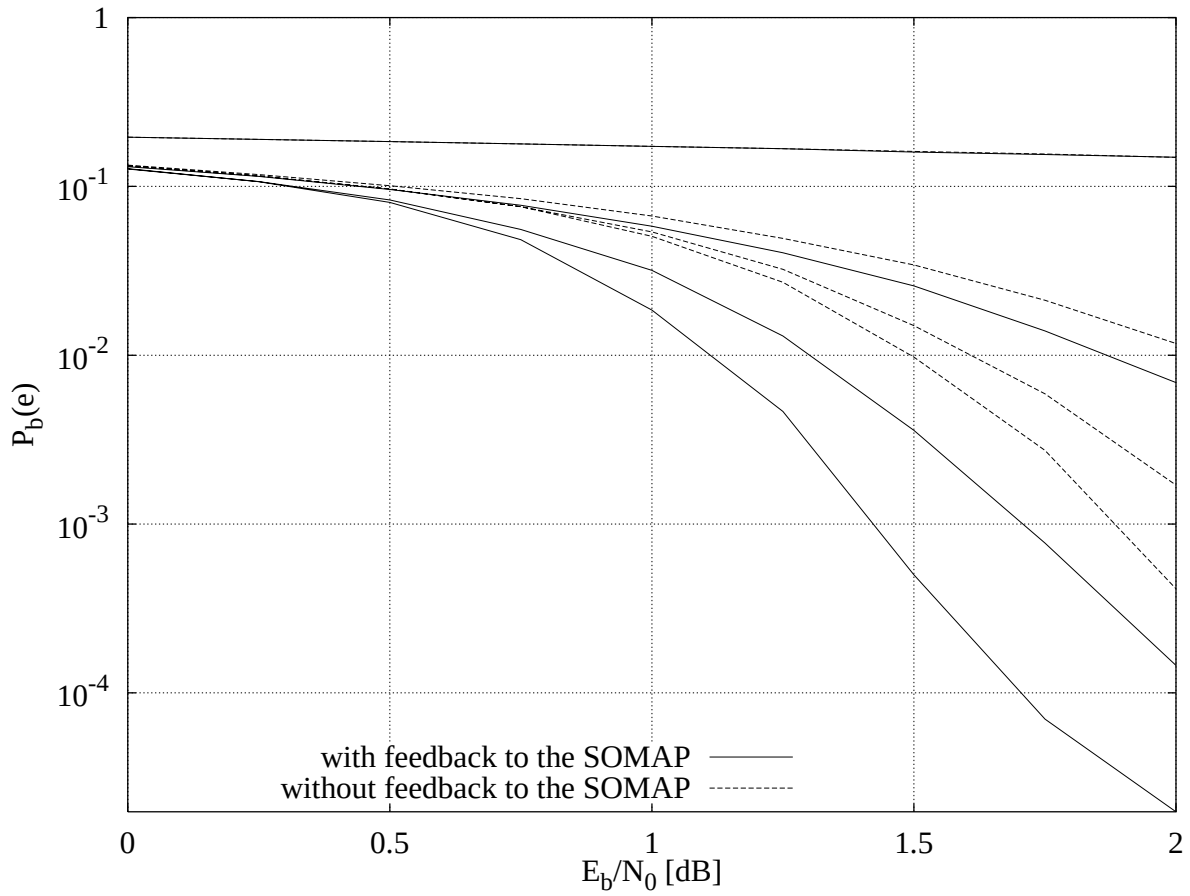


Fig. 9. Simulation results for the coding scheme of Fig. 7 with and without the feedback to the SOMAP.

decoding algorithm. At $P_b(e) = 10^{-4}$, a gain of 0.5 dB is achieved through the proper use of the SOMAP module.

As an alternative to the previous scheme, the three bits at the output of the PCCC, instead of being transmitted in parallel with an 8-PSK modulation, can be serially converted and then transmitted using a binary PSK modulation (see Fig 10). In this case, the Mapper at the modulator front-end is replaced by a simple P/S. Consequently, the decoding network can be simplified, since the correspondent SOPS does not use the information from the decoding network to update its output (dashed lines in Fig. 10).

B. A serial concatenated code with 2-PSK modulation

As a second example, we construct the serial concatenation of two convolutional codes (SCCC) as shown in Fig. 11. The input sequence is encoded by a rate 1/2, 4-state, systematic recursive convolutional encoder E_1, and the encoded sequence is interleaved by the symbol interleaver I. The output symbols from the interleaver are then encoded by the rate 2/3, 4-state, systematic

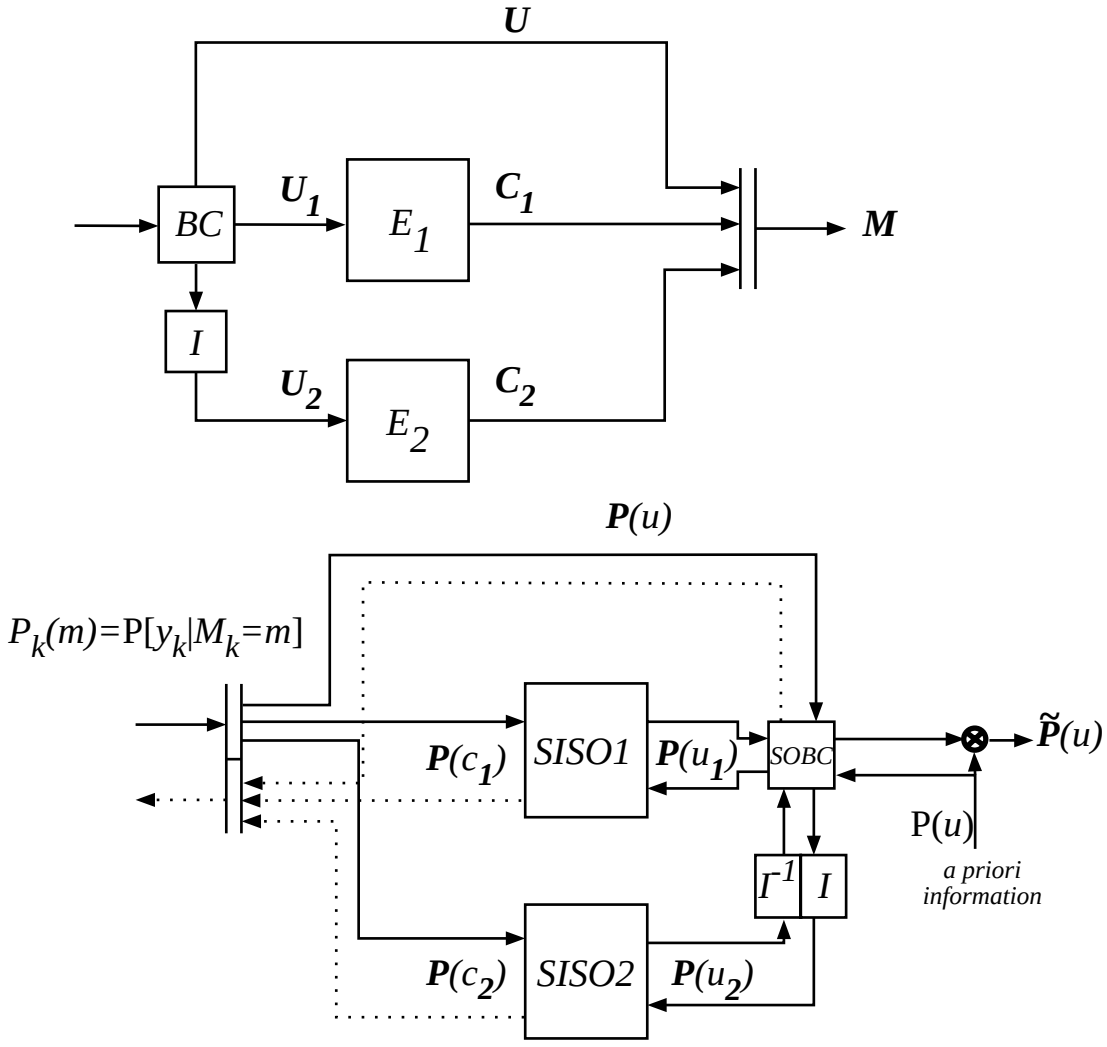


Fig. 10. The parallel concatenated convolutional coding scheme with BPSK modulation and its decoding network.

recursive convolutional encoder E_2. We obtain a rate 1/3 SCCC. The output symbols are transmitted over an AWGN channel using binary PSK modulation. The concatenated code requires two trellis encoders and one interleaver.

The iterative decoder is shown in the same Fig. 11, and is obtained specularly in a straightforward fashion.

The previous scheme uses a *symbol* interleaver acting on symbol belonging to a quaternary alphabet. In some cases, it can be more efficient using a *bit* interleaver, preceded by the cascade of a Mapper ($4 \leftrightarrow 2 \times 2$) and a P/S module, so as to obtain the desired bit sequence. The inverse block (S/P+SOMAP) must be placed before the inner encoder E_2 to regenerate the proper input

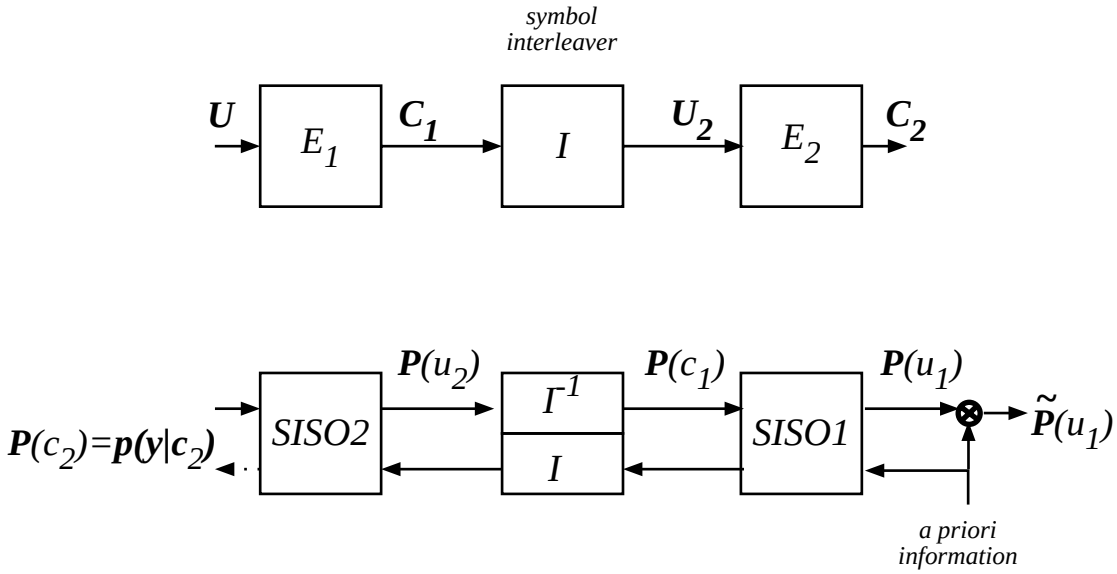


Fig. 11. The serial concatenated coding scheme using a symbol interleaver and its decoding network.

symbols. The obtained structure is shown in Fig. 12.

The performance of the SCCC with interleaving at bit level as in Fig. 12, obtained by simulation, are shown in Fig. 13, for an interleaver with length $N_{int} = 32,768$, which yields a decoding delay of 16,384 measured in information bits. The scheme simulated uses two 4-state encoders with rate $R_1 = 1/2$ and $R_2 = 2/3$. The bit error probability is plotted versus the signal-to-noise ratio E_b/N_0 with the number of iterations as a parameter.

C. Hybrid concatenated codes

Another example regards two different networks obtained through the concatenation of constituent codes part in parallel and part in series. We call them *hybrid concatenations*. Two examples of hybrid concatenations are reported in Figs. 14 and 15, where we show both the encoders and iterative decoders. Both hybrid schemes involve the use of three CCs and two interleavers. In Fig. 16 we show the bit error probability simulated performance of a rate 1/4, Type I hybrid scheme employing two interleavers with length $N_{int} = 256$ and the following three convolutional encoders: E_1 is a 4-state rate 1/2 feedforward convolutional encoder with generating matrix

$$G(D) = \left[1 + D + D^2, 1 + D^2\right] \quad .$$

E_2 is a 4-state rate 1/2 convolutional encoder with generating matrix

$$G(D) = \left[1, \frac{1 + D^2}{1 + D + D^2}\right] \quad ,$$

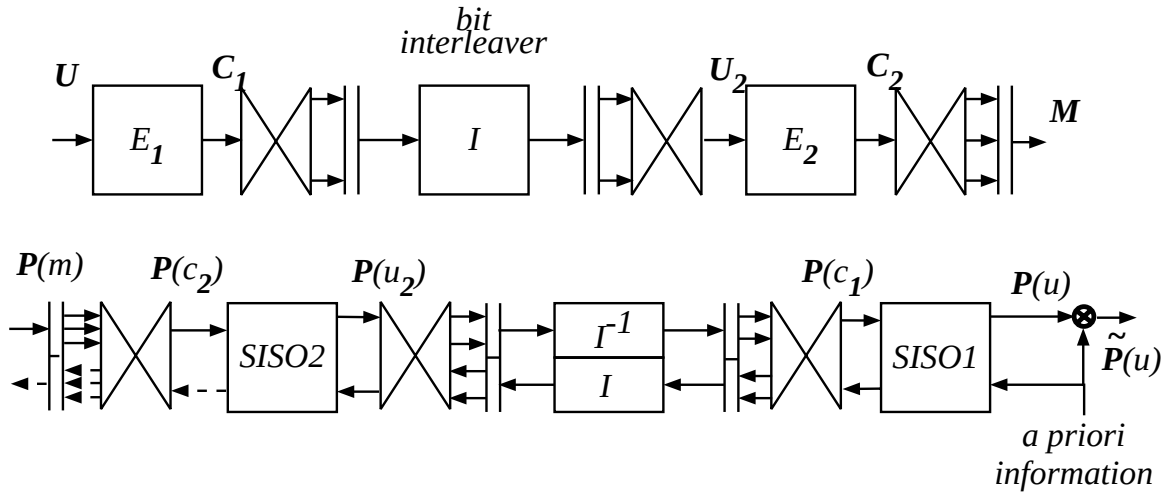


Fig. 12. The serial concatenated coding scheme using a bit interleaver and its decoding network.

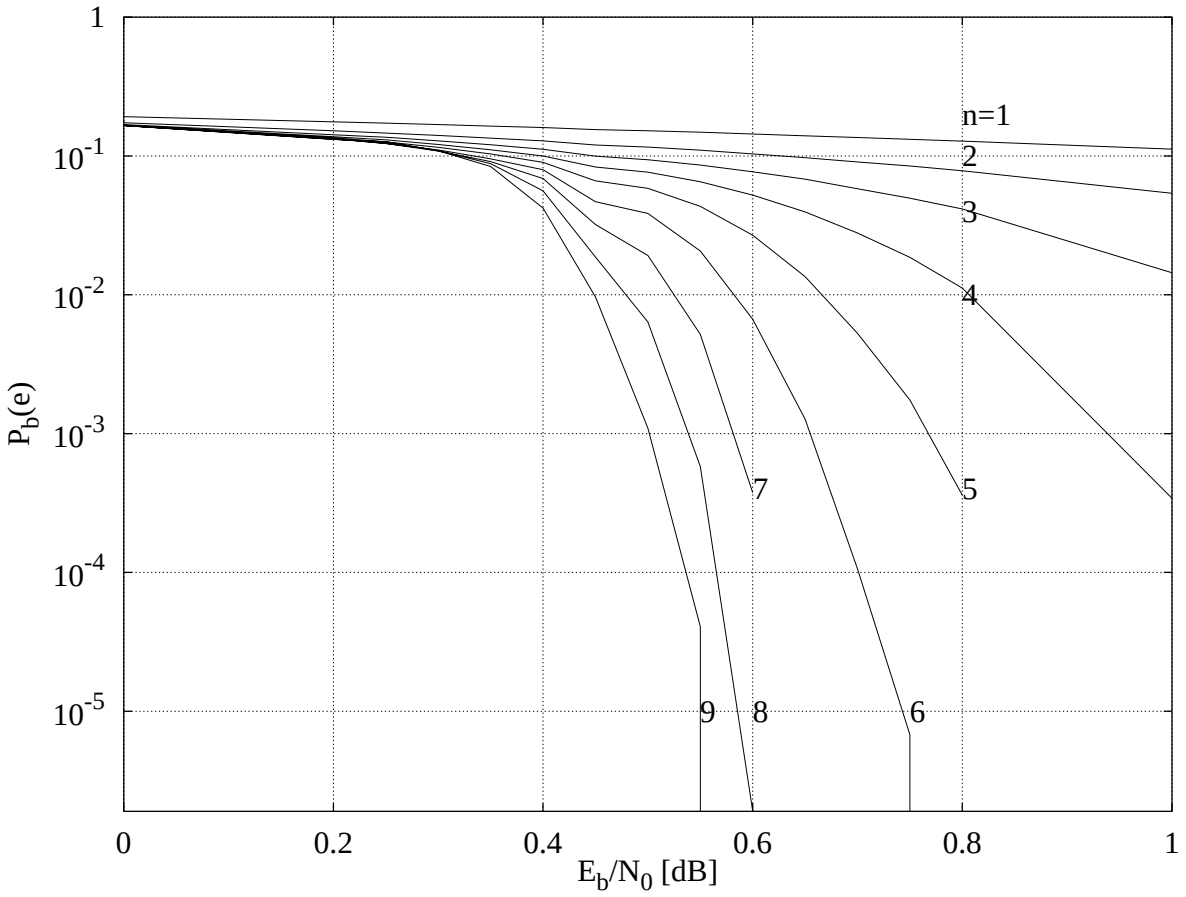


Fig. 13. Simulation results for the coding scheme of Fig. 12. E_1 is a nonrecursive 4 state encoder with rate $R_1 = 1/2$ and E_2 is a recursive 4 state encoder with rate $R_2 = 2/3$. The interleaver length is $N_{int} = 32,768$

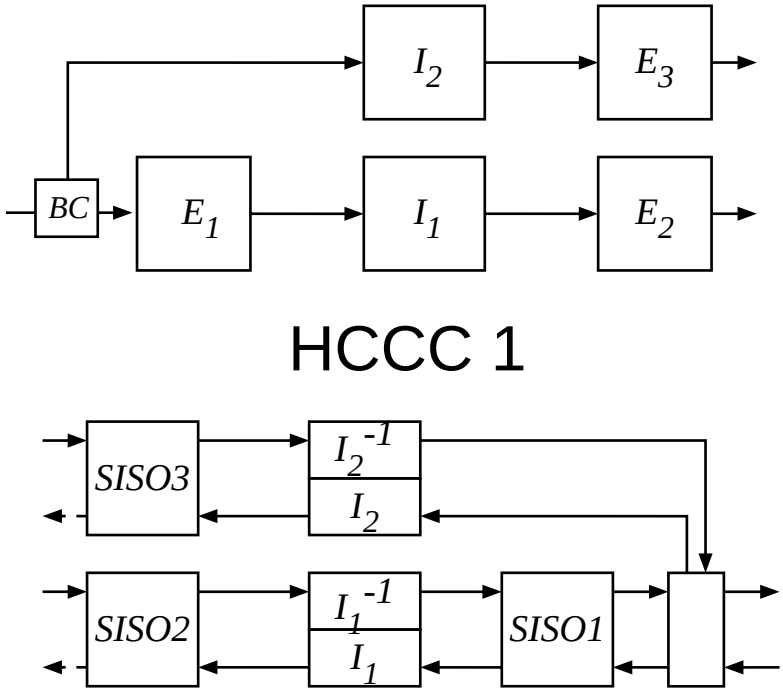


Fig. 14. Co-decoder for hybrid concatenation. (Type I)

uniformly punctured in the parity check bit in order to obtain a rate 2/3 encoder. E_2 is a 4 state rate 1/1 convolutional encoder with generating matrix

$$G(D) = \frac{1 + D^2}{1 + D + D^2} \quad .$$

The bit error probability is plotted versus the signal-to-noise ratio for a number of iterations in the range 1-5.

D. "Self"-concatenated codes

Consider the concatenated code shown in Fig. 17, together with its iterative decoder. It is composed by a rate 1/3 repetition code concatenated, after interleaving ($N_{int} = 256$) and parallel-to-serial conversion, with a 4-state rate 1/1 recursive convolutional encoder with generating matrix

$$G(D) = \frac{1 + D^2}{1 + D + D^2}$$

Transmitting also the information bit, we obtain overall a rate 1/4 code. Since the outer repetition code can be represented using one broadcasters, whose soft-input soft-output relationships are straightforward and do not require any computations, we need in this case only one SISO module (see the decoder in Fig. 17). For this reason, we call this scheme Type I "self"-concatenated code with interleaver. In Fig. 18 we plot the simulated bit error probability versus

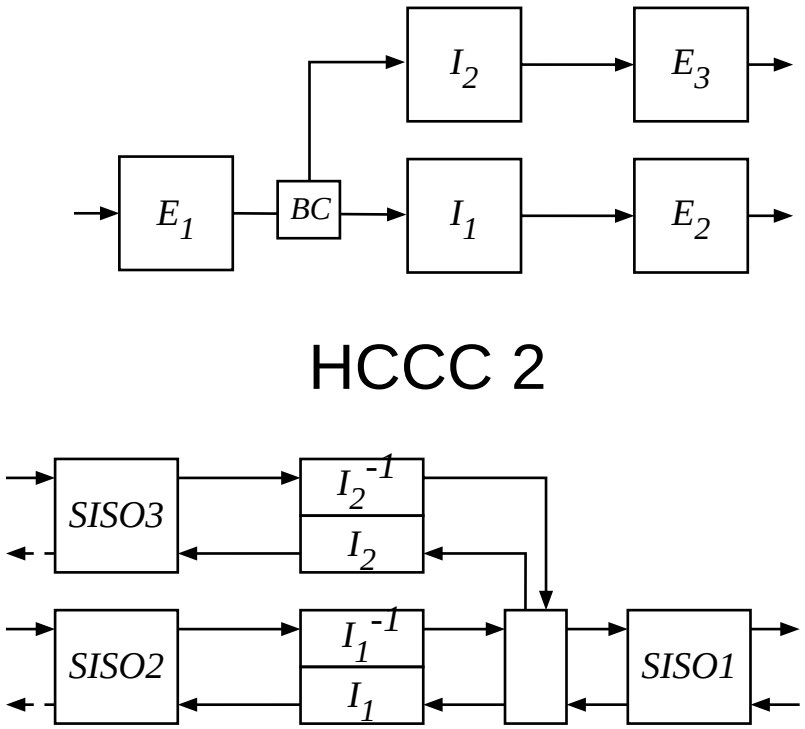


Fig. 15. Co-decoder for hybrid concatenation. (Type II)

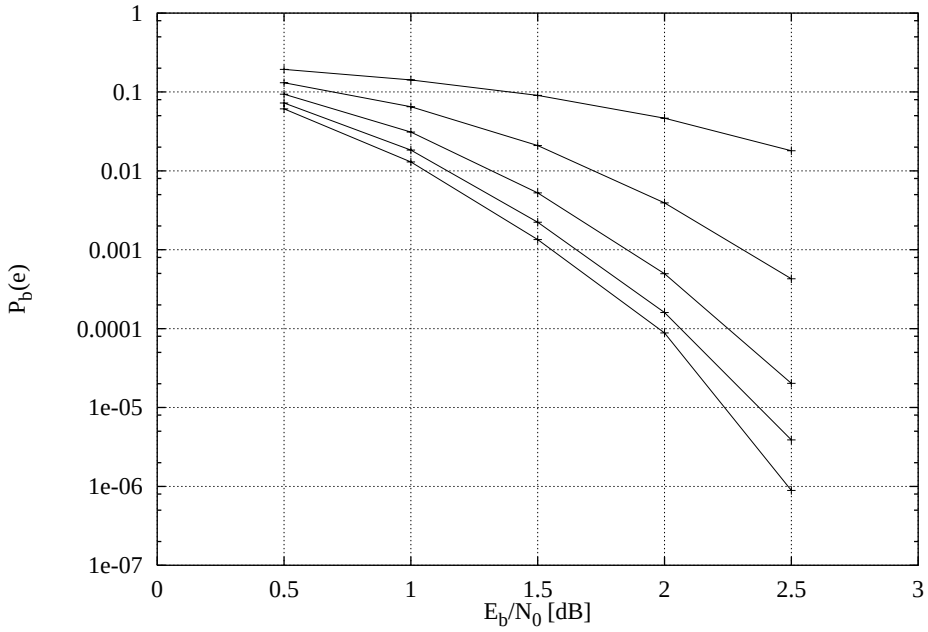


Fig. 16. Hybrid concatenation (Type I). Simulated bit-error probability versus signal-to-noise ratio for 1-5 iterations.

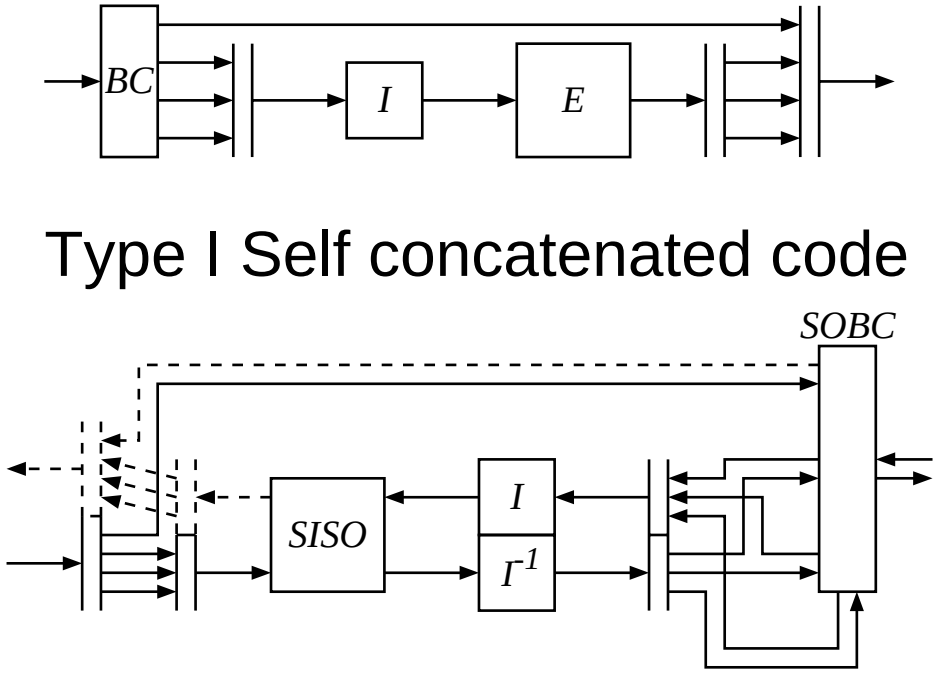


Fig. 17. Type I Self concatenation. Coding and decoding structures.

the signal to noise ratio for 1-5 iterations (for a fair comparison with turbo codes two uses of SISO were considered as one iteration).

Consider now the Type II rate 1/3 self-concatenated code shown in Fig. 19 together with its iterative decoder. A broadcaster produces three replicas of the input sequence: the first is directly sent to the modulator, while the second and third are permuted in different ways and jointly encoded using a 16-state rate 2/2 binary convolutional encoder with generating matrix

$$G(D) = \begin{bmatrix} \frac{1+D^2+D^3+D^4}{1+D^3+D^4} & \frac{1+D+D^4}{1+D^3+D^4} \\ \frac{1+D+D^4}{1+D^3+D^4} & \frac{1+D^4}{1+D^3+D^4} \end{bmatrix}$$

The simulation results for this concatenated code using the iterative decoder in Fig. 19 are shown in Fig. 20(a) for input block $N_{int} = 256$ and Fig. 20(b) for input block $N_{int} = 1024$.

E. Iterative decoder of a standard convolutional code

Convolutional codes, and, more generally, trellis codes, can be decoded using several decoding algorithms. The ML sequence decoding algorithm is the universally known Viterbi algorithm [28], suitable for codes of low-medium complexity (constraint lengths up to 15). The Viterbi algorithm presents memory requirements and computational complexity that increase exponentially with the constraint length. Thus, when dealing with very long constraint lengths (like for example in deep-space applications) to obtain large coding gains, one must abandon the Viterbi algorithm,

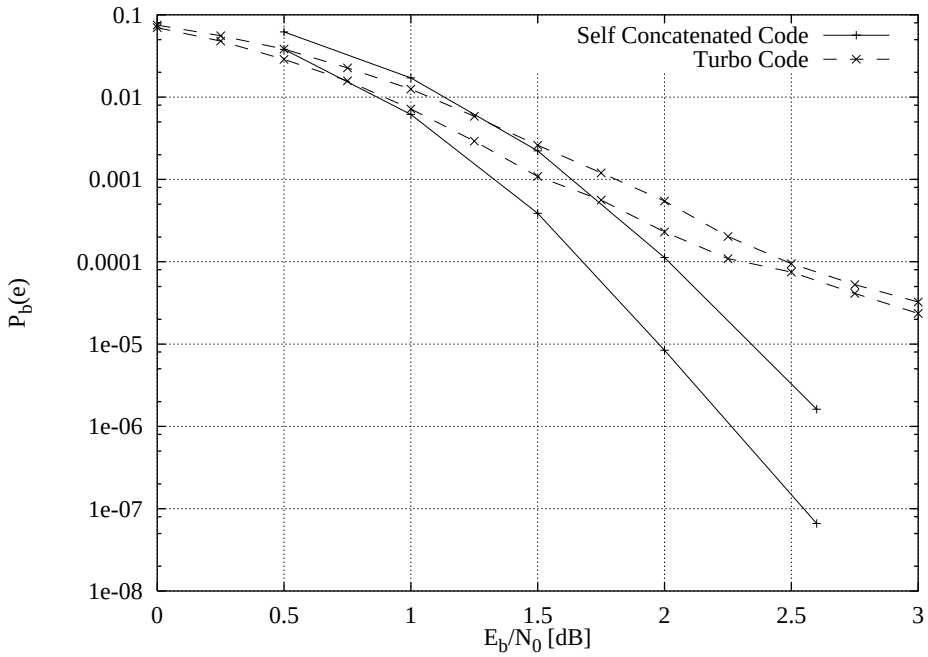


Fig. 18. Self concatenation. Simulated bit-error probability versus signal-to-noise ratio for 1-5 iterations.

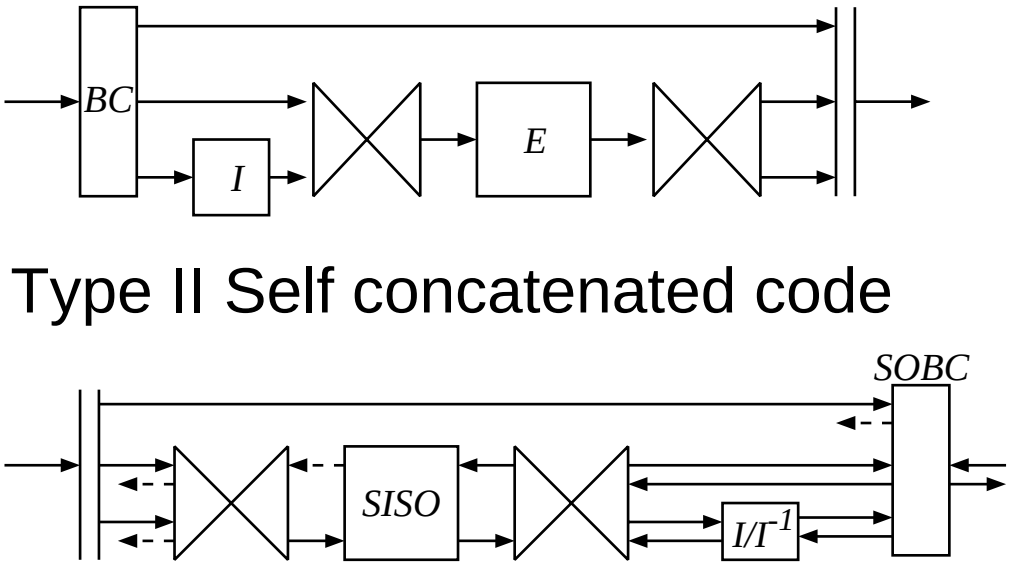
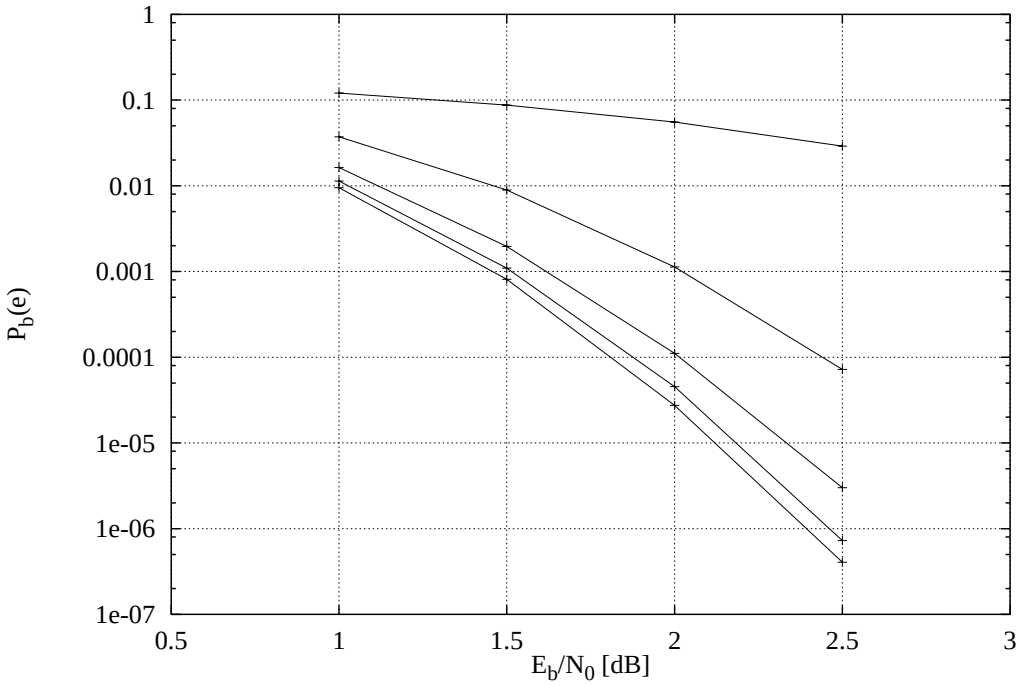
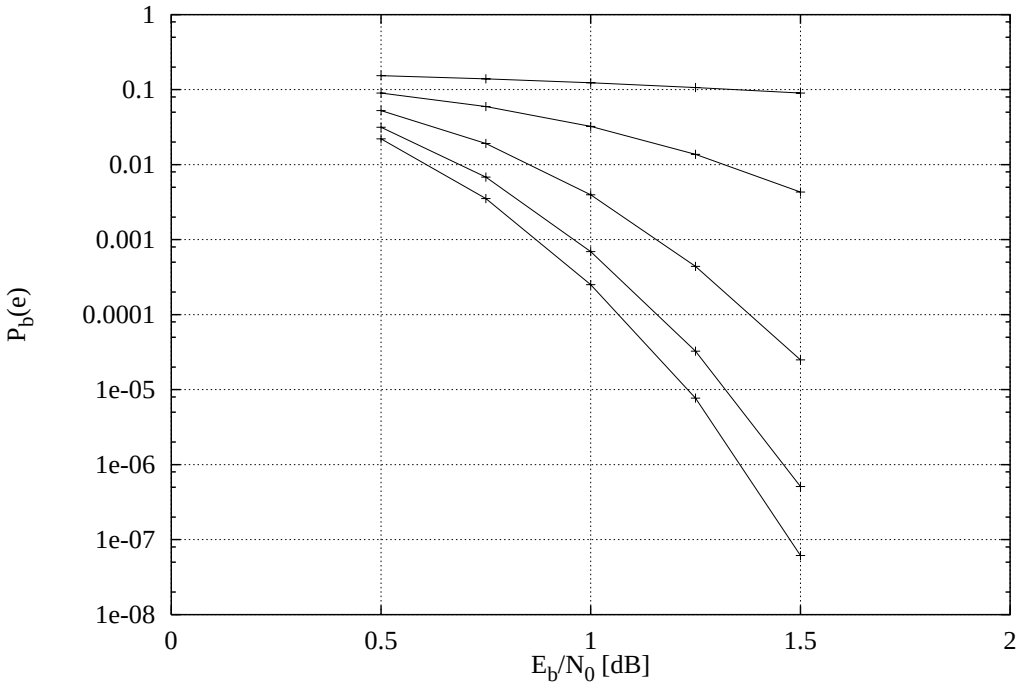


Fig. 19. Type II Self concatenation. Coding and decoding structures.



(a) Interleaving size $N_{int} = 256$



(b) Interleaving size $N_{int} = 1024$

Fig. 20. Type II Self concatenation. Simulated bit-error probability versus signal-to-noise ratio for 1-5 iterations.

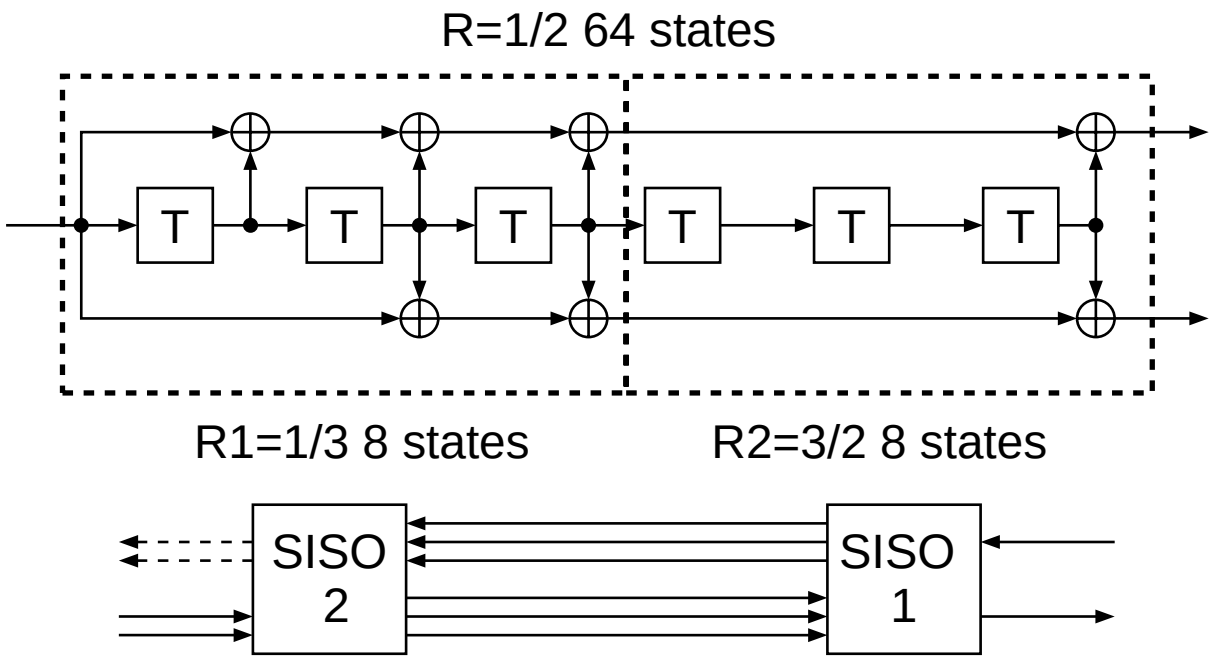


Fig. 21. A 64 state convolutional encoder split into two 8 state encoder and its corresponding iterative decoding network.

and use decoding techniques that do not track all paths in the trellis, like M-algorithms [29], sequential algorithms [30] or ad-hoc algorithms [31].

As the last application of the general iterative decoding technique described in this paper, we propose a completely new approach to decode convolutional, or, more generally, trellis codes. To describe it, let us consider, as an example, a rate 1/2 convolutional code with constraint length 6, whose encoder is shown in Fig. 21. Instead of decoding it using the Viterbi algorithm based on its 64-state trellis, we partition the encoder into the concatenation of two simpler subencoders with 8 states. The encoder can now be seen as the cascade of an 8-state, rate 1/3 "outer" encoder with an 8-state, rate 3/2 "inner" encoder, and we can applied to this concatenation the iterative decoder shown in Figure 21 formed by the two SISO module relative to the two cascaded encoders.

The performances of the iterative decoding algorithm, compared to those obtained with the optimal Viterbi algorithm applied to the 64-state original code, are plotted in Fig. 22 for a number of iterations ranging in the range 1-10.

VII. Conclusions

In this paper, we have introduced soft-input soft-output modules employed as building blocks to construct and iteratively decode in a distributed fashion *code networks*, a new concept that

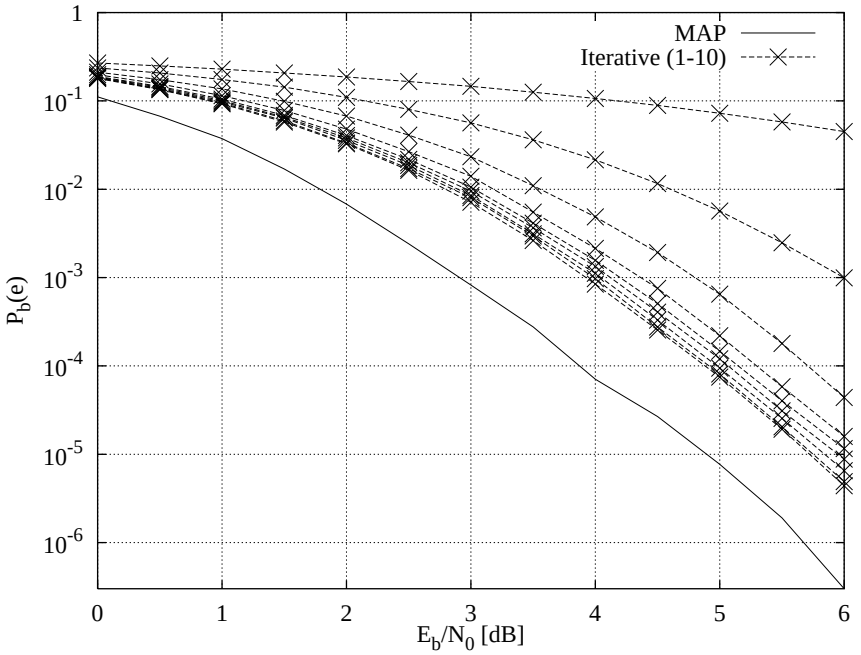


Fig. 22. Comparison of iterative decoding (1-10 iterations) and Viterbi ML decoding of a 64-state convolutional code.

includes, and generalizes, various forms of concatenated coding schemes. It has been shown that each encoder module has a well defined counterpart in the decoder, whose structure can be immediately derived by specularity from the encoder one. A variety of applications have been shown, including "turbo codes", serially concatenated codes with interleavers and a new technique to decode convolutional codes with large constraint lengths.

References

[1] Claude Berrou, Alain Glavieux, and Punya Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo-Codes", in *Proceedings of ICC'93*, Geneve, Switzerland, May 1993, pp. 1064–1070.

[2] Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara, "Serial concatenation of interleaved codes: Performance analysis, design and iterative decoding", *JPL TDA Progress Report*, vol. 42-126, Aug. 1996.

[3] Sergio Benedetto, Ezio Biglieri, and Valentino Castellani, *Digital Transmission Theory*, Prentice-Hall, New York, 1987.

[4] 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.

[5] 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, Oct. 1966.

[6] 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, Mar. 1974.

[7] P.L. McAdam, L. Welch, and C. Weber, "Map bit decoding of convolutional codes", in *Abstract of papers, ISIT'72*, Asilomar, California, Jan. 1972, p. 91.

[8] 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, Sept. 1976.

[9] Branka Vucetic and Yunxin Li, "A Survey of Soft-Output Algorithms", in *Proceedings of ISITA'94*, Sydney, Australia, Nov. 1994, pp. 863–867.

[10] S.S. Pietrobon and A.S. Barbulescu, "A simplification of the modified bahl algorithm for systematic convolutional codes", in *Proceedings of ISITA'94*, Sydney, Australia, Nov. 1994, pp. 1073–1077.

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

[12] Peter Jung, "Novel Low Complexity Decoder for Turbo Codes", *Electronic Letters*, vol. 31, no. 2, pp. 86–87, Jan. 1995.

[13] Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara, "Algorithm for continuous decoding of turbo codes", *Electronics Letters*, vol. 32, no. 4, pp. 314–315, Feb. 1996.

[14] L. Papke, "Improved decoding with the sova in a parallel concatenated (turbo-code) scheme", in *Proceedings of ICC'96*, Dallas, Texas, June 1996.

[15] Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara, "Soft-output decoding algorithms for continuous decoding of parallel concatenated convolutional codes", in *Proceedings of ICC'96*, Dallas, Texas, June 1996.

[16] G.D. Forney Jr., *Concatenated Codes*, M.I.T., Cambridge, MA, 1966.

[17] V.V. Ginzburg, "Multidimensional Signals for a Continuous Channel", *Problems of Information Transmission*, vol. 23, no. 4, pp. 20–34, Jan. 1984.

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

[19] Joachim Hagenauer and Peter Hoeher, "Concatenated Viterbi Decoding", in *Proceedings of Fourth Joint Swedish-Soviet Int. Workshop on Information Theory*, Gotland, Sweden, Studenlitteratur, Lund, Aug. 1989, pp. 29–33.

[20] Joachim Hagenauer, Elke Offer, and Lutz Papke, "Iterative decoding of binary block and convolutional codes", *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[21] P. Hoeher, "Tcm on frequency-selective fading channels: a comparison of soft-output probabilistic equalizers", in *Proceedings of GLOBECOM'90*, San Diego, California, Dec. 1990, pp. 401.4.1–401.4.6.

[22] Yunxin Li, Branka Vucetic, and Yoichi Sato, "Optimum Soft-Output Detection for Channels with Intersymbol Interference", *IEEE Transactions on Information Theory*, vol. 41, no. 3, pp. 704–713, May 1995.

[23] Robert McEliece, Eugene Rodemich, and Jung-Fu Cheng, "The turbo decision algorithm", in *Proceedings of 33rd Allerton Conference on Communication, Control and Computing*, Monticello, Illinois, Aug. 1995.

[24] Niclas Wiberg, Hans-Andrea Loeliger, and Ralf Kötter, "Codes and iterative decoding on general graphs", *European Transactions on Telecommunications*, vol. 6, no. 5, pp. 513–526, Oct. 1995.

[25] Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes", *JPL TDA Progress Report*, vol. 42-124, pp. 63–87, Feb. 1996.

[26] Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara, "Analysis, design and iterative decoding of double serially concatenated codes with interleavers", *IEEE Journal on Selected Areas on Communication*, Feb. 1998.

[27] Sergio Benedetto, Roberto Garello, and Guido Montorsi, "The trellis complexity of turbo codes", in *Proceedings of GLOBECOM '97 - Communications Theory Miniconference*, Phoenix, Arizona, Oct. 1997.

[28] A.J. Viterbi and J.K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New-York, 1979.

[29] John B. Anderson and Seshadri Mohan, *Source and Channel Coding*, Kluwer Academic Publishers, Boston, Dordrecht, London, 1991.

[30] George C. Clark and J. Bibb Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York and London, 1981.

[31] Sergio Benedetto and Guido Montorsi, "A new decoding algorithm for geometrically uniform codes: Description and performance", *IEEE Transactions on Communications*, vol. 44, no. 5, pp. 581–590, May 1996.

List of Figures

List of Tables