A Turbo Code Tutorial

William E. Ryan
New Mexico State University
Box 30001 Dept. 3-O, Las Cruces, NM 88003
wryan@nmsu.edu

Abstract— **We give a tutorial exposition of turbo codes and the associated algorithms. Included are a simple derivation for the performance of turbo codes, and a straightforward presentation of the iterative decoding algorithm. The derivations of both the performance estimate and the modified BCJR decoding algorithm are novel. The treatment is intended to be a launching point for further study in the field and, significantly, to provide sufficient information for the design of computer simulations.**

I. Introduction

Turbo codes, first presented to the coding community in 1993 [1], represent the most important breakthrough in coding since Ungerboeck introduced trellis codes in 1982 [2]. Whereas Ungerboeck's work eventually led to coded modulation schemes capable of operation near capacity on bandlimited channels [3], the original turbo codes offer near-capacity performance for deep space and satellite channels. The invention of turbo codes involved reviving some dormant concepts and algorithms, and combining them with some clever new ideas. Because the principles surrounding turbo codes are both uncommon and novel, it has been difficult for the initiate to enter into the study of these codes. Complicating matters further is the fact that there exist now numerous papers on the topic so that there is no clear place to begin study of these codes.

In this paper, we hope to address this problem by including in one paper an introduction to the study of turbo codes. We give a detailed description of the encoder and present a simple derivation of its performance in additive white Gaussian noise (AWGN). Particularly difficult for the novice has been the understanding and simulation of the iterative decoding algorithm, and so we give a thorough description of the algorithm here. This paper borrows from some of the most prominent publications in the field [4]-[9], sometimes adding details that were omitted in those works. However, the general presentation and some of the derivations are novel. Our goal is a self-contained, simple introduction to turbo codes for those already knowledgeable in the fields of algebraic and trellis codes.

The paper is organized as follows. In the next section we present the structure of the encoder, which leads to an estimate of its performance. The subsequent section then describes the iterative algorithm used to decode these codes. The treatment in each of these sections is meant to be sufficiently detailed so that one may with reasonable ease design a computer simulation of the encoder and decoder.

II. The Encoder and Its Performance

Fig. 1 depicts a standard turbo encoder. As seen in the figure, a turbo encoder is consists of two binary rate 1/2 convolutional encoders separated by an N-bit interleaver or permuter, together with an optional puncturing mechanism. Clearly, without the puncturer, the encoder is rate 1/3, mapping N data bits to $3N$ code bits. We observe that the encoders are configured in a manner reminiscent of classical concatenated codes. However, instead of cascading the encoders in the usual *serial* fashion, the encoders are arranged in a so-called *parallel* concatenation. Observe also that the consituent convolutional encoders are of the recursive systematic variety. Because any non-recursive (i.e., feedforward) non-catastrophic convolutional encoder is equivalent to a recursive systematic encoder in that they possess that same set of code sequences, there was no compelling reason in the past for favoring recursive encoders. However, as will be argued below, recursive encoders are necessary to attain the exceptional performance provided by turbo codes. Without any essential loss of generality, we assume that the constituent codes are identical. Before describing further details of the turbo encoder in its entirety, we shall first discuss its individual components.

A. The Recursive Systematic Encoders

Whereas the generator matrix for a rate 1/2 non-recursive convolutional code has the form $G_{NR}(D) = [g_1(D) \quad g_2(D)]$, the equivalent recursive systematic encoder has the generator matrix

$$G_R(D) = \left[1 \quad \frac{g_2(D)}{g_1(D)}\right] .$$

Observe that the code sequence corresponding to the encoder input $u(D)$ for the former code is $u(D)G_{NR}(D) = [u(D)g_1(D) \quad u(D)g_2(D)]$, and that the identical code sequence is produced in the recursive code by the sequence $u'(D) = u(D)g_1(D)$, since in this case the code sequence is $u(D)g_1(D)G_R(D) = u(D)G_{NR}(D)$. Here, we loosely call the pair of polynomials $u(D)G_{NR}(D)$ a code sequence, although the actual code sequence is derived from this polynomial pair in the usual way.

Observe that, for the recursive encoder, the code sequence will be of finite weight if and only if the input sequence is divisible by $g_1(D)$. We have the following immediate corollaries of this fact which we shall use later.

Corollary 1. A weight-one input will produce an infinite weight output (for such an input is never divisible by a polynomial $g_1(D)$).

Corollary 2. For any non-trivial $g_1(D)$, there exists a family of weight-two inputs of the form $D^j(1+D^{q-1})$, $j \geq 0$, which produce finite weight outputs, i.e., which are divisible by $g_1(D)$. When $g_1(D)$ is a primitive polynomial of degree m, then $q = 2^m$; more generally, $q-1$ is the length of the pseudorandom sequence generated by $g_1(D)$.

In the context of the code's trellis, Corollary 1 says that a weight-one input will create a path that diverges from the all-zeros path, but never remerges. Corollary 2 says that there will always exist a trellis path that diverges and remerges later which corresponds to a weight-two data sequence.

Example 1. Consider the code with generator matrix

$$G_R(D) = \begin{bmatrix} 1 & \dfrac{1+D^2+D^3+D^4}{1+D+D^4} \end{bmatrix} .$$

Thus, $g_1(D) = 1+D+D^4$ and $g_2(D) = 1+D^2+D^3+D^4$ or, in octal form, $(g_1, g_2) = (31, 27)$. Observe that $g_1(D)$ is primitive so that, for example, $u(D) = 1+D^{15}$ produces the finite-length code sequence $(1+D^{15}, 1+D+D^2+D^3+D^5+D^7+D^8+D^{11})$. Of course, any delayed version of this input, say, $D^7(1+D^{15})$, will simply produce a delayed version of this code sequence. Fig. 2 gives one encoder realization for this code. We remark that, in addition to elaborating on Corollary 2, this example serves to demonstrate the conventions generally used in the literature for specifying such encoders. □

B. The Permuter

As the name implies, the function of the permuter is to take each incoming block of N data bits and rearrange them in a pseudo-random fashion prior to encoding by the second encoder. Unlike the classical interleaver (e.g., block or convolutional interleaver), which rearranges the bits in some systematic fashion, it is important that the permuter sort the bits in a manner that lacks any apparent order, although it might be tailored in a certain way for weight-two and weight-three inputs as explained in Example 2 below. Also important is that N be selected quite large and we shall assume $N \geq 1000$ hereafter. The importance of these two requirements will be illuminated below. We point out also that one pseudo-random permuter will perform about as well as any other provided N is large.

C. The Puncturer

While for deep space applications low-rate codes are appropriate, in other situations such as satellite communications, a rate of 1/2 or higher is preferred. The role of the turbo code puncturer is identical to that of its convolutional code counterpart, to periodically delete selected bits to reduce coding overhead. For the case of iterative decoding to be discussed below, it is preferrable to delete only parity bits as indicated in Fig. 1, but there is no guarantee that this will maximize the minimum codeword distance. For example, to achieve a rate of 1/2, one might delete all even parity bits from the top encoder and all odd parity bits from the bottom one.

D. The Turbo Encoder and Its Performance

As will be elaborated upon in the next section, a maximum-likehood (ML) sequence decoder would be far too complex for a turbo code due to the presence of the permuter. However, the suboptimum iterative decoding algorithm to be described there offers near-ML performance. Hence, we shall now estimate the performance of an ML decoder (analysis of the iterative decoder is much more difficult).

Armed with the above descriptions of the components of the turbo encoder of Fig. 1, it is easy to conclude that it is linear since its components are linear. The constituent codes are certainly linear, and the permuter is linear since it may be modeled by a permutation matrix. Further, the puncturer does not affect linearity since all codewords share the same puncture locations. As usual, the importance of linearity is that, in considering the performance of a code, one may choose the all-zeros sequence as a reference. Thus, hereafter we shall assume that the all-zeros codeword was transmitted.

Now consider the all-zeros codeword (the 0^{th} codeword) and the k^{th} codeword, for some $k \in \{1, 2, ..., 2^N - 1\}$. The ML decoder will choose the k^{th} codeword over the 0^{th} codeword with probability $Q\left(\sqrt{2d_k r E_b/N_0}\right)$ where r is the code rate and d_k is the weight of the k^{th} codeword. The bit error rate for this two-codeword situation would then be

$$\begin{aligned} P_b(k \mid 0) &= w_k \text{ (bit errors/cw error)} \times \\ &\quad \frac{1}{N} \text{ (cw/ data bits)} \times \\ &\quad Q\left(\sqrt{2rd_kE_b/N_0}\right) \text{ (cw errors/cw)} \\ &= \frac{w_k}{N} Q\left(\sqrt{\frac{2rd_kE_b}{N_0}}\right) \text{ (bit errors/data bit)} \end{aligned}$$

where w_k is the weight of the k^{th} data word. Now including all of the codewords and invoking the usual union bounding argument, we may write

$$\begin{aligned} P_b &= P_b(\text{choose any } k \in \{1, 2, ..., 2^N - 1\} \mid 0) \\ &\leq \sum_{k=1}^{2^N-1} P_b(k \mid 0) \\ &= \sum_{k=1}^{2^N-1} \frac{w_k}{N} Q\left(\sqrt{\frac{2rd_kE_b}{N_0}}\right) . \end{aligned}$$

Note that every non-zero codeword is included in the above summation. Let us now reorganize the summation as

$$P_b \leq \sum_{w=1}^{N} \sum_{v=1}^{\binom{N}{w}} \frac{w}{N} Q\left(\sqrt{\frac{2rd_{wv}E_b}{N_0}}\right) \tag{1}$$

where the first sum is over the weight-w inputs, the second sum is over the $\binom{N}{w}$ different weight-w inputs, and d_{wv} is the weight of the v^{th} codeword produced by a weight-w input.

Consider now the first few terms in the outer summation of (1).

$\underline{w=1}$: From Corollary 1 and associated discussion above, weight-one inputs will produce only large weight codewords at both constituent encoder outputs since the trellis paths created never remerge with the all-zeros path. Thus, each d_{1v} is significantly greater than the minimum codeword weight so that the $w = 1$ terms in (1) will be negligible.

$\underline{w=2}$: Of the $\binom{N}{2}$ weight-two encoder inputs, only a fraction will be divisible by $g_1(D)$ (i.e., yield remergent paths) and, of these, only certain ones will yield the smallest weight, $d_{2,\min}^{CC}$, at a constituent encoder output (here, CC denotes "constituent code"). Further, with the permuter present, if an input $u(D)$ of weight-two yields a weight-$d_{2,\min}^{CC}$ codeword at the first encoder's output, it is unlikely that the permuted input, $u'(D)$, seen by the second encoder will also correspond to a weight-$d_{2,\min}^{CC}$ codeword (much less, be divisible by $g_1(D)$). We can be sure, however, that there will be some minimum-weight turbo codeword produced by a $w = 2$ input, and that this minimum weight can be bounded as

$$d_{2,\min}^{TC} \geq 2d_{2,\min}^{CC} - 2\,,$$

with equality when both of the constituent encoders produce weight-$d_{2,\min}^{CC}$ codewords (minus 2 for the bottom encoder). The exact value of $d_{2,\min}^{TC}$ is permuter dependent. We will denote the number of weight-two inputs which produce weight-$d_{2,\min}^{TC}$ turbo codewords by n_2 so that, for $w = 2$, the inner sum in (1) may be approximated as

$$\sum_{v=1}^{\binom{N}{2}} \frac{2}{N} Q\left(\sqrt{\frac{2rd_{2v}E_b}{N_0}}\right) \simeq \frac{2n_2}{N} Q\left(\sqrt{\frac{2rd_{2,\min}^{TC}E_b}{N_0}}\right) \quad . \quad (2)$$

$\underline{w=3}$: Following an argument similar to the $w = 2$ case, we can approximate the inner sum in (1) for $w = 3$ as

$$\sum_{v=1}^{\binom{N}{3}} \frac{3}{N} Q\left(\sqrt{\frac{2rd_{3v}E_b}{N_0}}\right) \simeq \frac{3n_3}{N} Q\left(\sqrt{\frac{2rd_{3,\min}^{TC}E_b}{N_0}}\right) \quad , \quad (3)$$

where n_3 and $d_{3,\min}^{TC}$ are obviously defined. While n_3 is clearly dependent on the interleaver, we can make some comments on its size relative to n_2 for a "randomly generated" interleaver. Although there are $(N-2)/3$ times as many $w = 3$ terms in the inner summation of (1) as there are $w = 2$ terms, we can expect the number of weight-three terms divisible by $g_1(D)$ to be of the order of the number of weight-two terms divisible by $g_1(D)$. Thus, most of the $\binom{N}{3}$ terms in (1) can be removed from consideration for this reason. Moreover, given a weight-three encoder input $u(D)$ divisible by $g_1(D)$ (e.g., $g_1(D)$ itself in the above example), it becomes very unlikely that the permuted input $u'(D)$ seen by the second encoder will also be divisible by $g_1(D)$. For example, suppose $u(D) = g_1(D) = 1 + D + D^4$. Then the permuter output will be a multiple of $g_1(D)$ if the three input $\mathbf{1}$'s become the j^{th}, $(j+1)^{th}$, and $(j+4)^{th}$ bits out of the permuter, for some j. If we imagine that the permuter acts in a purely random fashion so that the probability that one of the $\mathbf{1}$'s lands a given position is $1/N$, the permuter output will be $D^j g_1(D) = D^j\left(1 + D + D^4\right)$ with probabilility $3!/N^3$.[1] For comparison, for $w = 2$ inputs, a given permuter output pattern occurs with probability $2!/N^2$. Thus, we would expect the number of weight-three inputs, n_3, resulting in remergent paths in both encoders to be much less than n_2,

$$n_3 << n_2,$$

with the result being that the inner sum in (1) for $w = 3$ is negligible relative to that for $w = 2$.[2]

$\underline{w \geq 4}$: Again we can approximate the inner sum in (1) for $w = 4$ in the same manner as in (2) and (3). Still we would like to make some comments on its size for the "random" interleaver. A weight-four input might appear to the first encoder as a weight-three input concatenated some time later with a weight-one input, leading to a non-remergent path in the trellis and, hence, a negligible term in the inner sum in (1). It might also appear as a concatenation of two weight-two inputs, in which case the turbo codeword weight is at least $2d_{2,\min}^{TC}$, again leading to a negligible term in (1). Finally, if it happens to be some other pattern divisible by $g_1(D)$ at the first encoder, with probability on the order of $1/N^3$ it will be simultaneously divisible by $g_1(D)$ at the second encoder.[3] Thus, we may expect $n_4 << n_2$ so that the $w \geq 4$ terms are negligible in (1). The cases for $w > 4$ are argued similarly.

To summarize, the bound in (1) can be approximated as

$$P_b \simeq \max_{w \geq 2} \left\{ \frac{wn_w}{N} Q\left(\sqrt{\frac{2rd_{w,\min}^{TC}E_b}{N_0}}\right)\right\} \qquad (4)$$

where n_w and $d_{w,\min}^{TC}$ are functions of the particular interleaver employed. From our discussion above, we might expect that the $w = 2$ term dominates for a randomly generated interleaver, although it is easy to find interleavers with $n_2 = 0$ as seen in the example to follow. In any case,

[1] This is not the only weight-three pattern divisible by $g_1(D)$ — $g_1^2(D) = 1 + D^2 + D^8$ is another one, but this too has probability $3!/N^3$ of occurring.

[2] Because our argument assumes a "purely random" permuter, the inequality $n_3 << n_2$ has to be interpreted probabilistically. Thus, it is more accurate to write $E\{n_3\} << E\{n_2\}$ where the expectation is over all interleavers. Alternatively, for the *average* interleaver, we would expect $n_3 << n_2$; thus if $n_2 = 5$, say, we would expect $n_3 = 0$.

[3] The value of $1/N^3$ derives from that fact that ideally a particular divisible output pattern occurs with probability $4!/N^4$, but there will be approximately N shifted versions of that pattern, each divisible by $g_1(D)$.

we observe that P_b decreases with N, so that the error rate can be reduced simply by increasing the interleaver length. This effect is called *interleaver gain* (or permuter gain) and demonstrates the necessity of large permuters. Finally, we note that recursive encoders are crucial elements of a turbo code since, for non-recursive encoders, division by $g_1(D)$ (non-remergent trellis paths) would not be an issue and (4) would not hold (although (1) still would).

Example 2. We consider the performance of a rate 1/2, (31, 33) turbo code for two different interleavers of size $N = 1000$. We start first with an interleaver that was randomly generated . We found for this particular interleaver, $n_2 = 0$ and $n_3 = 1$, with $d^{TC}_{3,\min} = 9$, so that the $w = 3$ term dominates in (4). Interestingly, the interleaver input corresponding to this dominant error event was $D^{168}(1 + D^5 + D^{10})$ which produces the interleaver output $D^{88}(1 + D^{15} + D^{848})$, where of course both polynomials are divisible by $g_1(D) = 1 + D + D^4$. Figure 3 gives the simulated performance of of this code for 15 iterations of the iterative decoding algorithm detailed in the next section. Also included in Fig. 3 is the estimate of (4) for the same interleaver which is observed to be very close to the simulated values. The interleaver was then modified by hand to improve the weight spectrum of the code. It was a simple matter to attain $n_2 = 1$ with $d^{TC}_{2,\min} = 12$ and $n_3 = 4$ with $d^{TC}_{3,\min} = 15$ for this second interleaver so that the $w = 2$ term now dominates in (4). The simulated and estimated performance curves for this second interleaver are also included in Fig. 3. □

In addition to illustrating the use of the estimate (4), this example helps explain the unusual shape of the error rate curve: it may be interpreted as the usual Q-function shape for a signaling scheme with a modest $d_{\min}$, "pushed down" by the interleaver gain $w^* n_{w^*}/N$, where w^* is the maximizing value of w in (4).

III. The Decoder

Consider first an ML decoder for a rate 1/2 convolutional code (recursive or not), and assume a data word of length N, $N \geq 1000$ say. Ignoring the structure of the code, a naive ML decoder would have to compare (correlate) 2^N code sequences to the noisy received sequence, choosing in favor of the codeword with the best correlation metric. Clearly, the complexity of such an algorithm is exorbitant. Fortunately, as we know, such a brute force approach is simplified greatly by Viterbi's algorithm which permits a systematic elimination of candidate code sequences (in the first step, 2^{N-1} are eliminated, then another 2^{N-2} are eliminated on the second step, and so on). Unfortunately, we have no such luck with turbo codes, for the presence of the permuter immensely complicates the structure of a turbo code trellis, making these codes look more like block codes.

Just prior to the discovery of turbo codes, there was much interest in the coding community in suboptimal decoding strategies for concatenated codes, involving multiple (usually two) decoders operating cooperatively and iteratively. Most of the focus was on a type of Viterbi decoder which provides soft-output (or reliability) information to a companion soft-output Viterbi decoder for use in a subsequent decoding [10]. Also receiving some attention was the symbol-by-symbol maximum *a posteriori* (MAP) algorithm of Bahl, *et al* [11], published over 20 years ago. It was this latter algorithm, often called the BCJR algorithm, that Berrou, *et al* [1], utilized in the iterative decoding of turbo codes. We will discuss in this section the BCJR algorithm employed by each constituent decoder, but we refer the reader to [11] for the derivations of some of the results.

We first discuss a modified version of the BCJR algorithm for performing symbol-by-symbol MAP decoding. We then show how this algorithm is incorporated into an iterative decoder employing two BCJR-MAP decoders. We shall require the following definitions:

- E1 is a notation for encoder 1
- E2 is a notation for encoder 2
- D1 is a notation for decoder 1
- D2 is a notation for decoder 2
- m is the constituent encoder memory
- S is the set of all 2^m constituent encoder states
- $\mathbf{x}^s = (x_1^s, x_2^s, ..., x_N^s) = (u_1, u_2, ..., u_N)$ is the encoder input word
- $\mathbf{x}^p = (x_1^p, x_2^p, ..., x_N^p)$ is the parity word generated by a constituent encoder
- $y_k = (y_k^s, y_k^p)$ is a noisy (AWGN) version of (x_k^s, x_k^p)
- $\mathbf{y}_a^b = (y_a, y_{a+1}, ..., y_b)$
- $\mathbf{y} = \mathbf{y}_1^N = (y_1, y_2, ..., y_N)$ is the noisy received codeword

A. The Modified BCJR Algorithm

In the symbol-by-symbol MAP decoder, the decoder decides $u_k = +1$ if $P(u_k = +1 \mid \mathbf{y}) > P(u_k = -1 \mid \mathbf{y})$, and it decides $u_k = -1$ otherwise. More succinctly, the decision $\hat{u}_k$ is given by

$$\hat{u}_k = \text{sign } [L(u_k)]$$

where $L(u_k)$ is the log *a posteriori* probability (LAPP) ratio defined as

$$L(u_k) \triangleq \log \left(\frac{P(u_k = +1 \mid \mathbf{y})}{P(u_k = -1 \mid \mathbf{y})} \right) .$$

Incorporating the code's trellis, this may be written as

$$L(u_k) = \log \left(\frac{\sum\limits_{S^+} p(s_{k-1} = s', s_k = s, \mathbf{y})/p(\mathbf{y})}{\sum\limits_{S^-} p(s_{k-1} = s', s_k = s, \mathbf{y})/p(\mathbf{y})} \right) \qquad (5)$$

where $s_k \in S$ is the state of the encoder at time k, S^+ is the set of ordered pairs (s', s) corresponding to all state transitions $(s_{k-1} = s') \to (s_k = s)$ caused by data input $u_k = +1$, and S^- is similarly defined for $u_k = -1$.

Observe we may cancel $p(\mathbf{y})$ in (5) which means we require only an algorithm for computing $p(s', s, \mathbf{y}) = p(s_{k-1} = s', s_k = s, \mathbf{y})$. The BCJR algorithm [11] for doing this is

$$p(s', s, \mathbf{y}) = \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s) \tag{6}$$

where $\alpha_k(s) \triangleq p(s_k = s, \mathbf{y}_1^k)$ is computed recursively as

$$\alpha_k(s) = \sum_{s' \in S} \alpha_{k-1}(s') \gamma_k(s', s) \tag{7}$$

with initial conditions

$$\alpha_0(0) = 1 \text{ and } \alpha_0(s \neq 0) = 0 \ . \tag{8}$$

(These conditions state that the encoder is expected to start in state 0.) The probability $\gamma_k(s', s)$ in (7) is defined as

$$\gamma_k(s', s) \triangleq p(s_k = s, y_k \mid s_{k-1} = s') \tag{9}$$

and will be discussed further below. The probabilities $\beta_k(s) \triangleq p(\mathbf{y}_{k+1}^N \mid s_k = s)$ in (6) are computed in a "backward" recursion as

$$\beta_{k-1}(s') = \sum_{s \in S} \beta_k(s) \gamma_k(s', s) \tag{10}$$

with boundary conditions

$$\beta_N(0) = 1 \text{ and } \beta_N(s \neq 0) = 0 \ . \tag{11}$$

(The encoder is expected to end in state 0 after N input bits, implying that the last m input bits, called *termination bits*, are so selected.)

Unfortunately, cancelling the divisor $p(\mathbf{y})$ in (5) leads to a numerically unstable algorithm. We can include division by $p(\mathbf{y})/p(y_k)$ in the BCJR algorithm[4] by defining modified probabilities

$$\tilde{\alpha}_k(s) = \alpha_k(s)/p(\mathbf{y}_1^k)$$

and

$$\tilde{\beta}_k(s) = \beta_k(s)/p(\mathbf{y}_{k+1}^N \mid \mathbf{y}_1^k) \ .$$

Dividing (6) by $p(\mathbf{y})/p(y_k) = p(\mathbf{y}_1^{k-1})p(\mathbf{y}_{k+1}^N \mid \mathbf{y}_1^k)$, we obtain

$$p(s', s \mid \mathbf{y})p(y_k) = \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s', s) \cdot \tilde{\beta}_k(s) \ . \tag{12}$$

Note since $p(\mathbf{y}_1^k) = \sum_{s \in S} \alpha_k(s)$, the values $\tilde{\alpha}_k(s)$ may be computed from $\{\alpha_k(s) : s \in S\}$ via

$$\tilde{\alpha}_k(s) = \alpha_k(s) / \sum_{s \in S} \alpha_k(s) \ . \tag{13}$$

[4] Unfortunately, dividing by simply $p(\mathbf{y})$ to obtain $p(s', s \mid \mathbf{y})$ also leads to an unstable algorithm. Obtaining $p(s', s \mid \mathbf{y})p(y_k)$ instead of the APP $p(s', s \mid \mathbf{y})$ presents no problem since an APP ratio is computed so that the unwanted factor $p(y_k)$ cancels; see equation (16) below.

But since we would like to avoid storing both $\{\alpha_k(s)\}$ and $\{\tilde{\alpha}_k(s)\}$, we can use (7) in (13) to obtain a recursion involving only $\{\tilde{\alpha}_k(s)\}$,

$$\begin{aligned} \tilde{\alpha}_k(s) &= \frac{\sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s)}{\sum_s \sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s)} \\ &= \frac{\sum_{s'} \tilde{\alpha}_{k-1}(s') \gamma_k(s', s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s') \gamma_k(s', s)} \ , \end{aligned} \tag{14}$$

where the second equality follows by dividing the numerator and the denominator by $p(\mathbf{y}_1^{k-1})$.

The recursion for $\tilde{\beta}_k(s)$ can be obtained by noticing that

$$\begin{aligned} p(\mathbf{y}_k^N \mid \mathbf{y}_1^{k-1}) &= p(\mathbf{y}_1^k) \cdot \frac{p(\mathbf{y}_{k+1}^N \mid \mathbf{y}_1^k)}{p(\mathbf{y}_1^{k-1})} \\ &= \sum_s \sum_{s'} \alpha_{k-1}(s') \gamma_k(s', s) \cdot \frac{p(\mathbf{y}_{k+1}^N \mid \mathbf{y}_1^k)}{p(\mathbf{y}_1^{k-1})} \\ &= \sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s') \gamma_k(s', s) \cdot p(\mathbf{y}_{k+1}^N \mid \mathbf{y}_1^k) \end{aligned}$$

so that dividing (10) by this equation yields

$$\tilde{\beta}_{k-1}(s') = \frac{\sum_s \tilde{\beta}_k(s) \gamma_k(s', s)}{\sum_s \sum_{s'} \tilde{\alpha}_{k-1}(s') \gamma_k(s', s)} \ . \tag{15}$$

In summary, the modified BCJR-MAP algorithm involves computing the LAPP ratio $L(u_k)$ by combining (5) and (12) to obtain

$$L(u_k) = \log \left(\frac{\sum\limits_{S^+} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s', s) \cdot \tilde{\beta}_k(s)}{\sum\limits_{S^-} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k(s', s) \cdot \tilde{\beta}_k(s)} \right) \tag{16}$$

where the $\tilde{\alpha}$'s and $\tilde{\beta}$'s are computed recursively via (14) and (15), respectively. Clearly the $\{\tilde{\alpha}_k(s)\}$ and $\{\tilde{\beta}_k(s)\}$ share the same boundary conditions as their counterparts as given in (8) and (11). Computation of the probabilities $\gamma_k(s', s)$ will be discussed shortly.

On the topic of stability, we should point out also that the algorithm given here works in software, but a hardware implementation would employ the "log-MAP" algorithm [12], [14]. In fact, most software implementations these days use the log-MAP, although they can be slower than the algorithm presented here if not done carefully. The algorithm presented here is close to the earlier turbo decoding algorithms [1], [13], [4].

B. Iterative MAP Decoding

From Bayes' rule, the LAPP ratio for an arbitrary MAP decoder can be written as

$$L(u_k) = \log \left(\frac{P(\mathbf{y} \mid u_k = +1)}{P(\mathbf{y} \mid u_k = -1)} \right) + \log \left(\frac{P(u_k = +1)}{P(u_k = -1)} \right)$$

with the second term representing *a priori* information. Since $P(u_k = +1) = P(u_k = -1)$ typically, the *a priori* term is usually zero for conventional decoders. However, for *iterative* decoders, D1 receives *extrinsic* or *soft* information for each u_k from D2 which serves as *a priori* information. Similarly, D2 receives extrinsic information from D1 and the decoding iteration proceeds as D1→D2→D1→D2→..., with the previous decoder passing soft information along to the next decoder at each half-iteration except for the first. The idea behind extrinsic information is that D2 provides soft information to D1 for each u_k, using only information not available to D1 (i.e., E2 parity); D1 does likewise for D2.

An iterative decoder using component BCJR-MAP decoders is shown in Fig. 4. Observe how permuters and de-permuters are involved in arranging systematic, parity, and extrinsic information in the proper sequence for each decoder.

We now show how extrinsic information is extracted from the modified-BCJR version of the LAPP ratio embodied in (16). We first observe that $\gamma_k(s', s)$ may be written as (cf. equation (9))

$$\begin{aligned}\gamma_k(s', s) &= P(s \mid s')p(y_k \mid s', s) \\ &= P(u_k)p(y_k \mid u_k)\end{aligned}$$

where the event u_k corresponds to the event $s' \to s$. Defining

$$L^e(u_k) \triangleq \log\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right) ,$$

observe that we may write

$$\begin{aligned}P(u_k) &= \left(\frac{\exp[-L^e(u_k)/2]}{1 + \exp[-L^e(u_k)]}\right) \cdot \exp[u_k L^e(u_k)/2] \\ &= A_k \exp[u_k L^e(u_k)/2] \qquad (17)\end{aligned}$$

where the first equality follows since it equals

$$\begin{aligned}\left(\frac{\sqrt{P_-/P_+}}{1 + P_-/P_+}\right)\sqrt{P_+/P_-} &= P_+ \text{ when } u_k = +1 \text{ and} \\ \left(\frac{\sqrt{P_-/P_+}}{1 + P_-/P_+}\right)\sqrt{P_-/P_+} &= P_- \text{ when } u_k = -1 .\end{aligned}$$

where we have defined $P_+ \triangleq P(u_k = +1)$ and $P_- \triangleq P(u_k = -1)$ for convenience. As for $p(y_k \mid u_k)$, we may write (recall $y_k = (y_k^s, y_k^p)$ and $x_k = (x_k^s, x_k^p) = (u_k, x_k^p)$)

$$\begin{aligned}p(y_k \mid u_k) &\propto \exp\left[-\frac{(y_k^s - u_k)^2}{2\sigma^2} - \frac{(y_k^p - x_k^p)^2}{2\sigma^2}\right] \\ &= \exp\left[-\frac{{y_k^s}^2 + u_k^2 + {y_k^p}^2 + {x_k^p}^2}{2\sigma^2}\right] \\ &\quad \cdot \exp\left[\frac{u_k y_k^s + x_k^p y_k^p}{\sigma^2}\right] \\ &= B_k \exp\left[\frac{y_k^s u_k + y_k^p x_k^p}{\sigma^2}\right]\end{aligned}$$

so that

$$\gamma_k(s', s) \propto A_k B_k \exp[u_k L^e(u_k)/2] \exp\left[\frac{u_k y_k^s + x_k^p y_k^p}{\sigma^2}\right] . \qquad (18)$$

Now since $\gamma_k(s', s)$ appears in the numerator (where $u_k = +1$) and denominator (where $u_k = -1$) of (16), the factor $A_k B_k$ will cancel as it is independent of u_k. Also, since we assume transmission of the symbols ± 1 over the channel, $\frac{E_c}{N_0/2} = \frac{1}{\sigma^2}$ so that $\sigma^2 = N_0/2E_c$ where $E_c = rE_b$ is the energy per channel bit. From (18), we then have

$$\begin{aligned}\gamma_k(s', s) &\sim \exp\left[\frac{1}{2}u_k\left(L^e(u_k) + L_c y_k^s\right) + \frac{1}{2}L_c y_k^p x_k^p\right] \\ &= \exp\left[\frac{1}{2}u_k\left(L^e(u_k) + L_c y_k^s\right)\right] \cdot \gamma_k^e(s', s) \qquad (19)\end{aligned}$$

where $L_c \triangleq \frac{4E_c}{N_0}$ and where

$$\gamma_k^e(s', s) \triangleq \exp\left[\frac{1}{2}L_c y_k^p x_k^p\right] .$$

Combining (19) with (16) we obtain

$$\begin{aligned}L(u_k) &= \log\left(\frac{\sum_{S^+} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s) \cdot C_k}{\sum_{S^-} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s) \cdot C_k}\right) \\ &= L_c y_k^s + L^e(u_k) \qquad (20) \\ &\quad + \log\left(\frac{\sum_{S^+} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s)}{\sum_{S^-} \tilde{\alpha}_{k-1}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k(s)}\right) .\end{aligned}$$

where $C_k \triangleq \exp\left[\frac{1}{2}u_k\left(L^e(u_k) + L_c y_k^s\right)\right]$. The second equality follows since $C_k(u_k = +1)$ and $C_k(u_k = -1)$ can be factored out of the summations in the numerator and denominator, respectively. The first term in (20) is sometimes called the *channel value,* the second term represents any *a priori* information about u_k provided by a previous decoder, and the third term represents *extrinsic information* that can be passed on to a subsequent decoder. Thus, for example, on any given iteration, D1 computes

$$L_1(u_k) = L_c y_k^s + L_{21}^e(u_k) + L_{12}^e(u_k)$$

where $L_{21}^e(u_k)$ is extrinsic information passed from D2 to D1, and $L_{12}^e(u_k)$ is the third term in (20) which is to be used as extrinsic information from D1 to D2.

C. Pseudo-Code for the Iterative Decoder

We do not give pseudo-code for the encoder here since this is much more straightforward. However, it must be emphasized that at least E1 must be terminated correctly to avoid serious degradation. That is, the last m bits of the N-bit information word to be encoded must force E1 to the zero state by the N^{th} bit.

The pseudo-code given below for iterative decoding of a turbo code follows directly from the development above. Implicit is the fact that each decoder must have full knowledge of the trellis of the constituent encoders. For example, each decoder must have a table (array) containing the input bits and parity bits for all possible state transitions $s' \to s$. Also required are permutation and de-permutation functions (arrays) since D1 and D2 will be sharing reliability information about each u_k, but D2's information is pemuted relative to D1. We denote these arrays by $P[\cdot]$ and $Pinv[\cdot]$, respectively. For example, the permuted word $\mathbf{u}'$ is obtained from the original word $\mathbf{u}$ via the pseudo-code statement: for $k = 1 : N$, $u'_k = u_{P[k]}$, end. We next point out that due to the presence of L_c in $L(u_k)$, knowledge of the noise variance $N_0/2$ by each MAP decoder is necessary. Finally, we mention that a simple way to simulate puncturing is, in the computation of $\gamma_k(s', s)$, to set to zero the received parity samples, y_k^{1p} or y_k^{2p}, corresponding to the punctured parity bits, x_k^{1p} or x_k^{2p}. Thus, puncturing need not be performed at the encoder.

===== Initialization =====

D1:

- $\tilde{\alpha}_0^{(1)}(s) = 1$ for $s = 0$
  $\quad = 0$ for $s \neq 0$
- $\tilde{\beta}_N^{(1)}(s) = 1$ for $s = 0$
  $\quad = 0$ for $s \neq 0$
- $L_{21}^e(u_k) = 0$ for $k = 1, 2, ..., N$

D2:

- $\tilde{\alpha}_0^{(2)}(s) = 1$ for $s = 0$
  $\quad = 0$ for $s \neq 0$
- $\tilde{\beta}_N^{(2)}(s) = \tilde{\alpha}_N^{(2)}(s)$ for all s (set after computation of $\{\tilde{\alpha}_N^{(2)}(s)\}$ in the first iteration)[5]
- $L_{12}^e(u_k)$ is to be determined from D1 after the first half-iteration and so need not be initialized

========================

===== The n^{th} interation =====

D1:

for $k = 1 : N$

- get $y_k = (y_k^s, y_k^{1p})$ where y_k^{1p} is a noisy version of E1 parity
- compute $\gamma_k(s', s)$ from (19) for all allowable state transitions $s' \to s$ (u_k in (19) is set to the value of the encoder input which caused the transition $s' \to s$; $L^e(u_k)$ is in this case $L_{21}^e(u_{Pinv[k]})$, the de-permuted extrinsic informaton from the previous D2 iteration)
- compute $\tilde{\alpha}_k^{(1)}(s)$ for all s using (14)

end

for $k = N : -1 : 2$

- compute $\tilde{\beta}_{k-1}^{(1)}(s)$ for all s using (15)

end

for $k = 1 : N$

- compute $L_{12}^e(u_k)$ using

$$L_{12}^e(u_k) = \log \left(\frac{\sum\limits_{S^+} \tilde{\alpha}_{k-1}^{(1)}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k^{(1)}(s)}{\sum\limits_{S^-} \tilde{\alpha}_{k-1}^{(1)}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k^{(1)}(s)} \right)$$

end

D2:

for $k = 1 : N$

- get $y_k = (y_{P[k]}^s, y_k^{2p})$
- compute $\gamma_k(s', s)$ from (19) for all allowable state transitions $s' \to s$ (u_k in (19) is set to the value of the encoder input which caused the transition $s' \to s$; $L^e(u_k)$ is $L_{12}^e(u_{P[k]})$, the permuted extrinsic informaton from the previous D1 iteration; y_k^s is the permuted systematic value, $y_{P[k]}^s$)
- compute $\tilde{\alpha}_k^{(2)}(s)$ for all s using (14)

end

for $k = N : -1 : 2$

- compute $\tilde{\beta}_{k-1}^{(2)}(s)$ for all s using (15)

end

for $k = 1 : N$

- compute $L_{21}^e(u_k)$ using

$$L_{21}^e(u_k) = \log \left(\frac{\sum\limits_{S^+} \tilde{\alpha}_{k-1}^{(2)}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k^{(2)}(s)}{\sum\limits_{S^-} \tilde{\alpha}_{k-1}^{(2)}(s') \cdot \gamma_k^e(s', s) \cdot \tilde{\beta}_k^{(2)}(s)} \right)$$

end

========================

===== After the last iteration =====

for $k = 1 : N$

- compute

$$L_1(u_k) = L_c y_k^s + L_{21}^e(u_{Pinv[k]}) + L_{12}^e(u_k)$$

- if $L_1(u_k) > 0$

  decide $u_k = +1$

[5] Note encoder 2 cannot be simply terminated due to the presence of the interleaver. The strategy implied here leads to a negligible loss in performance. Other termination strategies can be found in the literature (e.g., [15]).

else

decide $u_k = -1$

end
===========================

Acknowledgment

The author would like to thank Omer Acikel of New Mexico State University for help with Example 2, and Esko Nieminen of Nokia and Prof. Steve Wilson of the University of Virginia for helpful suggestions.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error- correcting coding and decoding: Turbo codes," *Proc. 1993 Int. Conf. Comm.*, pp. 1064- 1070.

[2] G. Ungerboeck, "Channel coding with multi-level/phase signals," *IEEE Trans. Inf. Theory*, pp. 55-67, Jan. 1982.

[3] M. Eyuboglu, G. D. Forney, P. Dong, G. Long, "Advanced modulation techniques for V.Fast," *Eur. Trans. on Telecom.*, pp. 243-256, May 1993.

[4] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (turbo) codes," *Proc. GlobeCom 1994*, pp. 1298-1303.

[5] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inf. Theory*, pp. 409-428, Mar. 1996.

[6] S. Benedetto and G. Montorsi, "Design of parallel concatenated codes," *IEEE Trans. Comm.*, pp. 591-600, May 1996.

[7] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, pp. 429-445, Mar. 1996.

[8] D. Arnold and G. Meyerhans, "The realization of the turbo-coding system," Semester Project Report, Swiss Fed. Inst. of Tech., Zurich, Switzerland, July, 1995.

[9] L. Perez, J. Seghers, and D. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inf. Theory*, pp. 1698-1709, Nov. 1996.

[10] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," *Proc. GlobeCom 1989*, pp. 1680-1686.

[11] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, pp. 284-287, Mar. 1974.

[12] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," *Proc. 1995 Int. Conf. on Comm.*, pp. 1009-1013.

[13] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Comm.*, pp. 1261-1271, Oct. 1996.

[14] A. Viterbi, “An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes,” *IEEE JSAC*, pp. 260-264, Feb. 1998.

[15] D. Divsalar and F. Pollara, “Turbo codes for PCS applications,” *Proc. 1995 Int. Conf. Comm.*, pp. 54-59.

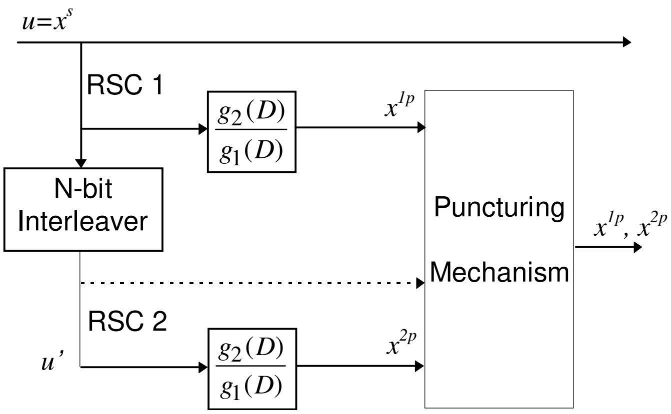

Fig. 1. Diagram of a standard turbo encoder with two identical recursive systematic encoders (RSC's).

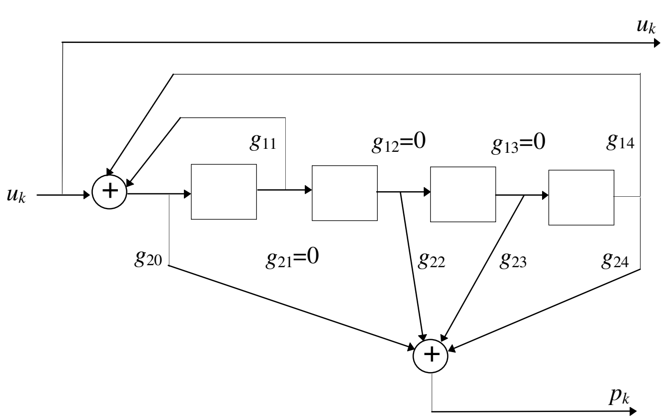

Fig. 2. Recursive systematic encoder for code generators $(g_1, g_2) = (31,\ 27)$.

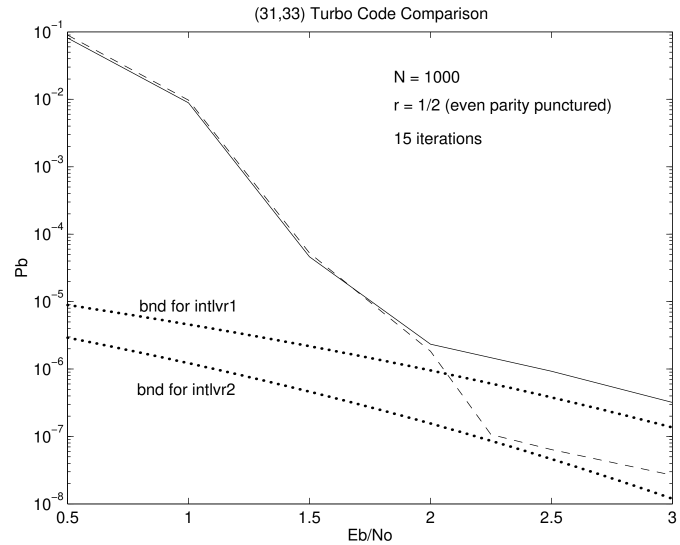

Fig. 3. Simulated performance of the rate 1/2 (31, 33) turbo code for two different interleavers ($N = 1000$) together with the asymptotic performance of each predicted by (4).

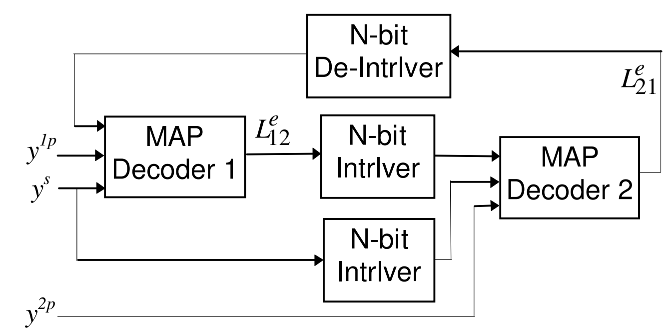

Fig. 4. Diagram of iterative (turbo) decoder which uses two MAP decoders operating cooperatively. L_{12}^e is “soft” or extrinsic information from D1 to D2, and L_{21}^e is defined similarly. The final decisions may come from either D1 or D2.