Turbo Codes for PCS Applications

D. Divsalar and F. Pollara[1]
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

ABSTRACT: Turbo codes are the most exciting and potentially important development in coding theory in many years. They were introduced in 1993 by Berrou, Glavieux and Thitimajshima [1], and claimed to achieve near Shannon-limit error correction performance with relatively simple component codes and large interleavers. A required E_b/N_o of 0.7 dB was reported for BER of 10^{-5} and code rate of 1/2 [1]. However, some important details that are necessary to reproduce these results were omitted. This paper confirms the accuracy of these claims, and presents a complete description of an encoder/decoder pair that could be suitable for PCS applications. We describe a new simple method for trellis termination, we analyze the effect of interleaver choice on the weight distribution of the code, and we introduce the use of unequal rate component codes which yields better performance. Turbo codes are extended to encoders with multiple codes and a suitable decoder structure is developed, which is substantially different from the decoder for two-code based encoders.

I. INTRODUCTION

Coding theorists have traditionally attacked the problem of designing good codes by developing codes with a lot of structure which lends to feasible decoders, although coding theory suggests that codes chosen "at random" should perform well if their block size is large enough. The challenge to find practical decoders for "almost" random, large codes has not been seriously considered until recently. Perhaps the most exciting and potentially important development in coding theory in recent years has been the dramatic announcement of "Turbo-codes" by Berrou et al. in 1993 [1]. The announced performance of these codes was so good that the initial reaction of the coding establishment was deep skepticism, but recently researchers around the world have been able to reproduce those results [3]- [4]. The introduction of turbo-codes has opened a whole new way of looking at the problem of constructing good codes and decoding them with low complexity.

These codes are claimed to achieve near Shannon-limit error correction performance with relatively simple component codes and large interleavers. A required E_b/N_0 of 0.7 dB was reported for BER of 10^{-5} [1]. However, some important details that are necessary to reproduce these results were omitted. The purpose of this paper is to shed some light on the accuracy of these claims, and to present a complete description of an encoder/decoder pair that could be suitable for PCS applications, where lower rate codes can be used.

For example, in multiple-access schemes like CDMA the capacity (maximum number of users per cell) can be expressed as $C = \frac{\eta}{E_b/N_o} + 1$, where η is the processing gain and E_b/N_o is the required signal-to-noise ratio to achieve a desired bit error rate (BER) performance. For a given BER, a smaller required E_b/N_o implies a larger capacity or cell size. Unfortunately, to reduce E_b/N_o it is necessary to use very complex codes (e.g. large constraint length convolutional codes). In this paper, we design turbo codes suitable for CDMA and PCS applications that can achieve superior performance with limited complexity. For example, if a (7,1/2) convolutional code is used at BER=10^{-3}, the capacity is $C = 0.5\eta$. However, if two (5,1/3) punctured convolutional codes or three (4,1/3) punctured codes are used in a turbo encoder structure, the capacity can be increased to $C = 0.8\eta$ (with 192-bits and 256-bits interleavers which correspond to 9.6 Kbps and 13 Kbps with roughly 20ms frames). Higher capacity can be obtained with larger interleavers. Note that low rate codes can be used for CDMA since an integer number of chips per coded symbol is used and bandwidth is defined mainly by chip rate.

Three new contributions are reported in this paper: a new simple method for trellis termination, the use of unequal rate component codes which results in better performance, and the development of decoders for multiple-code encoders — the original turbo decoder scheme operates in serial mode, while for multiple-code encoders we found that the decoder for the whole turbo code based on the optimum MAP rule must operate in parallel mode, and we derived the appropriate metric as illustrated in Sec. III.

II. PARALLEL CONCATENATION OF CONVOLUTIONAL CODES

The codes considered in this paper consist of the parallel concatenation of multiple convolutional codes with random interleavers (permutations) at the input of each encoder. Fig. 1 illustrates a particular example that will be used in this paper to verify the performance of these codes. The

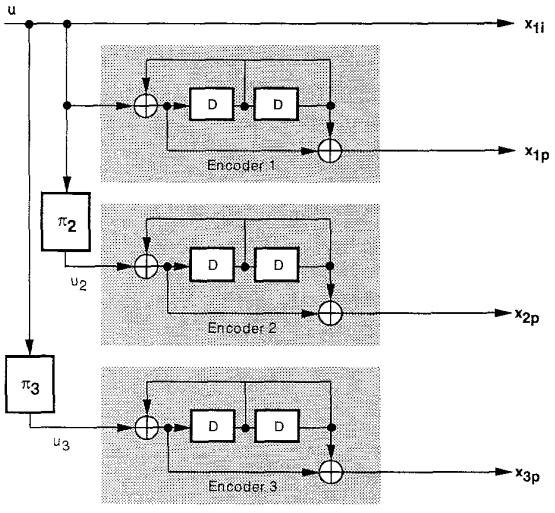

Figure 1: Example of encoder with three codes

encoder contains three recursive binary convolutional encoders, with M_1, M_2 and M_3 memory cells respectively. In general, the three component encoders may not be identical. The first component encoder operates directly on the information bit sequence $\mathbf{u} = (u_1, \ldots, u_N)$ of length N, producing the two output sequences $\mathbf{y}_{1i}$ and $\mathbf{y}_{1p}$. The second component encoder operates on a reordered sequence of information bits $\mathbf{u}_2$ produced by an interleaver π_2 of length N, and outputs the sequence $\mathbf{y}_{2p}$. Similarly, subsequent component encoders operate on a reordered sequence of information bits $\mathbf{u}_j$ produced by interleaver π_j and output the sequence $\mathbf{y}_{jp}$. The interleaver is a pseudo-random block scrambler defined by a permutation of N elements with no repetitions: a complete block is read into the the interleaver and read out in a specified (fixed) random order. The same interleaver is used repeatedly for all subsequent blocks. Figure 1 shows an example where a rate $r = 1/n = 1/4$ code is generated by three

[1]The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

component codes with $M_1 = M_2 = M_3 = M = 2$, producing the outputs $\mathbf{y}_{1i} = \mathbf{u}$, $y_{1p} = \mathbf{u} \cdot g_b/g_a$, $\mathbf{y}_{2p} = \mathbf{u}_2 \cdot g_b/g_a$, and $\mathbf{y}_{3p} = \mathbf{u}_3 \cdot g_b/g_a$, where the generator polynomials g_a and g_b have octal representation $(7)_{octal}$ and $(5)_{octal}$, respectively. Note that various code rates can be obtained by proper puncturing of $\mathbf{y}_{1p}$, $\mathbf{y}_{2p}$ and $\mathbf{y}_{3p}$. The design of the constituent convolutional codes, which are not necessarily optimum convolutional codes, is still under investigation. It was suggested in [5] that good codes are obtained if g_b is a primitive polynomial.

Trellis Termination — We use the encoder in Fig. 1 to generate a $(n(N + M), N)$ block code, where the M tail bits of code 2 and code 3 are not transmitted. Since the component encoders are recursive, it is not sufficient to set the last M information bits to zero in order to drive the encoder to the all zero state, i.e. to terminate the trellis. The termination (tail) sequence depends on the state of each component encoder after N bits, which makes it impossible to terminate both component encoders with just M bits. This issue has not been resolved in previously proposed turbo code implementations. Fortunately, the simple stratagem illustrated in Fig. 2 is sufficient to terminate the trellis at the end of the block. (The specific code shown is not important). Here the switch is in position "A" for the first N clock cycles and is in position "B" for M additional cycles, which will flush the encoders with zeros. The decoder does not assume knowledge of the M tail bits. The same termination method will be used for all encoders.

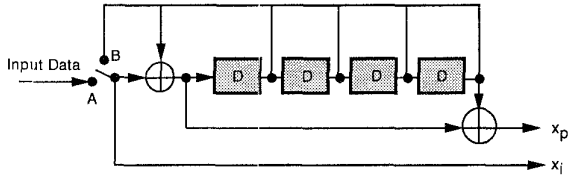

Figure 2: Trellis Termination

Weight Distribution — In order to estimate the performance of a code it is necessary to have information about its minimum distance, weight distribution, or actual code geometry, depending on the accuracy required for the bounds or approximations. The challenge is in finding the pairing of codewords from each individual encoder, induced by a particular set of interleavers. Intuitively, we would like to avoid joining low-weight codewords from one encoder with low-weight words from the other encoders. In the example of Fig. 1, the component codes have minimum distances 5,2 and 2. This will produce a worst-case minimum distance of 9 for the overall code. Note that this would be unavoidable if the encoders were not recursive since, in this case, the minimum weight word for all three encoders is generated by the input sequence $\mathbf{u} = (00\ldots0000100\ldots000)$ with a single "1", which will appear again in the other encoders, for any choice of interleavers. This motivates the use of recursive encoders, where the key ingredient is the recursiveness and not the fact that the encoders are systematic. For our example, the input sequence $\mathbf{u} = (00\ldots00100100\ldots000)$ generates a low weight codeword with weight 6, for the first encoder. If the interleavers do not "break" this input pattern, the resulting codewords weight will be 14. In general weight-2 sequences with $2 + 3t$ zeros separating the 1's would result in a total weight of $14 + 6t$ if there were no permutations.

With permutations before the second and third encoders, a weight-2 sequence with its 1's separated by $2 + 3t_1$ zeros will be permuted into two other weight-2 sequences with 1's separated by $2 + 3t_i$ zeros, $i = 2, 3$, where each t_i is defined as a multiple of $1/3$. If any t_i is not an integer, the corresponding encoded output will have a high weight because then the convolutional code output is non-terminating (until the end of the block). If all t_i's are integers, the total encoded weight will be $14+2\sum_{i=1}^{3} t_i$. Thus, one of the considerations in designing the interleaver is to avoid integer triplets (t_1, t_2, t_3) that are simultaneously small in all three components. In fact, it would be nice to design an interleaver to guarantee that the smallest value of $\sum_{i=1}^{3} t_i$ (for integer t_i) grows with the block size N.

For comparison we consider the same encoder structure in Fig. 1, except with the roles of g_a and g_b reversed. Now the minimum distances of the three component codes are 5, 3, and 3, producing an overall minimum distance of 11 for the total code without any permutations. This is apparently a better code, but it turns out to be inferior as a turbo code. This paradox is explained by again considering the critical weight-2 data sequences. For this code, weight-2 sequences with $1 + 2t_1$ zeros separating the two 1's produce non-terminating output and hence low-weight encoded words. In the turbo encoder, such sequences will be permuted to have separations $1 + 2t_i$, $i = 2, 3$, for the second and third encoders, where now each t_i is defined as a multiple of $1/2$. But now the total encoded weight for integer triplets (t_1, t_2, t_3) is $11 + \sum_{i=1}^{3} t_i$. Notice how this weight grows only half as fast with $\sum_{i=1}^{3} t_i$ as the previously calculated weight for the original code. If $\sum_{i=1}^{3} t_i$ can be made to grow with block size by proper choice of interleaver, then clearly it is important to choose component codes that cause the overall weight to grow as fast as possible with the individual separations t_i. This consideration outweighs the criterion of selecting component codes that would produce the highest minimum distance if unpermuted.

There are also many weight-n, $n = 3, 4, 5, \ldots$, data sequences that produce terminating output and hence low encoded weight. However, as argued below, these sequences are much more likely to be broken up by the random interleavers than the weight-2 sequences and are therefore likely to produce non-terminating output from at least one of the encoders. Thus, turbo code structures which have low minimum distances (if unpermuted) due strictly to higher-weight input sequences are often superior to other turbo codes with higher unpermuted minimum distances that are caused by weight-2 input sequences.

Weight Distribution with Random Interleavers — Now we briefly examine the issue of whether one or more random interleavers can avoid matching small separations between the 1's of a weight-2 data sequence with equally small separations between the 1's of its permuted version(s). Consider for example a particular weight-2 data sequence $(\ldots001001000\ldots)$ which corresponds to a low weight codeword in each of the encoders of Fig. 1. If we select randomly an interleaver of size N, the probability that this sequence will be permuted into another sequence of the same form is roughly $2/N$ (assuming that N is large, and ignoring minor edge effects). The probability that such an unfortunate pairing happens for at least one possible position of the original sequence $(\ldots001001000\ldots)$ within the block size of N, is approximately $1 - (1 - 2/N)^N \approx 1 - e^{-2}$. This implies that the minimum distance of a two-code turbo code constructed with a random permutation is not likely to be much higher than the encoded weight of such an unpermuted weight-2 data sequence, e.g. 14 for the code in Fig. 1. (For the worst case permutations, the d_{min} of the code is still 9, but these permutations are highly unlikely if chosen randomly). By contrast, if we use three codes and two different interleavers, the probability that a particular sequence $(\ldots001001000\ldots)$ will be reproduced by both interleavers is only $(2/N)^2$. Now the probability of finding such an unfortunate data sequence somewhere within the block of size N is roughly $1 - \left[1 - (2/N)^2\right]^N \approx 4/N$. Thus it is probable that a three-code turbo code using two random interleavers will see an increase in its minimum distance beyond the encoded weight of an unpermuted weight-2 data sequence. This argument can be extended to account for other weight-2 data sequences which may also produce low weight codewords, e.g. $(\ldots00100(000)'1000\ldots)$, for the code in Fig. 1. For comparison, let

us consider a weight-3 data sequence such as (...0011100...) which for our example corresponds to the minimum distance of the code (using no permutations). The probability that this sequence is reproduced with one random interleaver is roughly $6/N^2$, and the probability that some sequence of the form (...0011100...) is paired with another of the same form is $1-(1-6/N^2)^N \approx 6/N$. Thus for large block sizes, the bad weight-3 data sequences have a small probability of being matched with bad weight-3 permuted data sequences, even in a two-code system.

For a turbo code using q codes and $q-1$ random interleavers this probability is even smaller, $1-\left[1-(6/N^2)^{q-1}\right]^N \approx \frac{6}{N}(\frac{6}{N^2})^{q-2}$. This implies that the minimum distance codeword of the turbo code in Fig. 1 is more likely to result from a weight-2 data sequence of the form (...001001000...) than from the weight-3 sequence (...0011100...) that produces the minimum distance in the unpermuted version of the same code. Higher weight sequences have even smaller probability of reproducing themselves after being passed through a random interleaver.

For a turbo code using q codes and $q-1$ interleavers, the probability that a weight-n data sequence will be reproduced somewhere within the block by all $q-1$ permutations is of the form $1-\left[1-(\beta/N^{n-1})^{q-1}\right]^N$, where β is a number that depends on the weight-n data sequence but does not increase with block size N. For large N, this probability is proportional to $(1/N)^{nq-n-q}$, which falls off rapidly with N, when n and q are greater than two. Furthermore, the symmetry of this expression indicates that increasing either the weight of the data sequence n or the number of codes q has roughly the same effect on lowering this probability.

In summary, from the above arguments we conclude that weight-2 data sequences are an important factor in the design of the component codes, and that higher weight have decreasing importance. Also, increasing the number of codes may result in better turbo codes. More accurate results and derivations are discussed in [6].

The minimum distance is not the most important quantity of the turbo code, except for its asymptotic performance, at very high E_b/N_o. At moderate SNRs, the weight distribution for the first several possible weights is necessary to compute the code performance. Estimating the complete weight distribution of these codes for large N and fixed interleavers is still an open problem. However, it is possible to estimate the weight distribution for large N for random interleavers by using probabilistic arguments. (See [4] for further considerations on the weight distribution).

Interleaver Design — Interleavers should be capable of spreading low-weight input sequences so that the resulting codeword has high weight. Block interleavers, defined by a matrix with ν_r rows and ν_c columns such that $N = \nu_r \times \nu_c$, may fail to spread certain sequences. For example, the weight 4 sequence shown in Fig. 3 cannot be broken by a block interleaver. In order to break such sequences random interleavers are desirable. (A method for the design of interleavers is discussed in [3]). Block interleavers are effective if the low-weight sequence is confined to a row. If low-weight sequences (which can be regarded as the combination of lower weight sequences) are confined to several consecutive rows, then the ν_c columns of the interleaver should be sent in a specified order to spread as much as possible the low-weight sequence. A method for reordering the columns is given in [8]. This method guarantees that for any number of columns $\nu_c = aq + r$, $(r \leq a-1)$, the minimum separation between data entries is $q-1$, where a is the number of columns affected by a burst. However, as can be observed in the example in Fig. 3, the sequence 1001 will still appear at the input of the encoders for any possible column permutation. Only if we permute the rows of the interleaver in addition to its columns it is possible to break the low-weight sequences. The method in [8] can be used again for the permutation of rows. Appropriate selection of of a, and q for rows and columns depends on the particular set of codes used and on the specific low-weight sequences that we would like to break. We designed random permutations (interleavers) by generating random integers i, $1 \leq i \leq N$, without replacement. We define a "S-random" permutation as follows: each randomly selected integer is compared to S previously selected integers. If the current selection is equal to any S previous selections within a distance of $\pm S$, then the current selection is rejected. This process is repeated until all N integers are selected. While the searching time increases with S, we observed that choosing $S < \sqrt{N/2}$ usually produces a solution in reasonable time. (For $S = 1$ we have a purely random interleaver).

In the simulations we used $S = 11$ for $N = 256$ and $S = 31$ for $N = 4096$.

The advantage of using three or more constituent codes is that the corresponding two or more interleavers have a better chance to break sequences that were not taken care by another interleaver. The disadvantage is that, for an overall desired code rate, each code must be punctured more, resulting in weaker constituent codes. In our experiments, we have used randomly selected interleavers and interleavers based on the row-column permutation described above. In general, randomly selected permutations are good for low SNR operation (e.g., PCS applications requiring $P_b = 10^{-3}$) where the overall weight distribution of the code is more important than the minimum distance

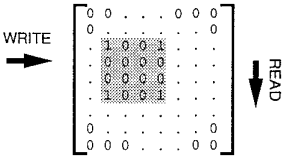

Figure 3: Example where a block interleaver fails to "break" the input sequence.

III. Turbo Decoding Configuration

The turbo decoding configuration proposed in [1] for two codes is shown schematically in Fig. 4. This configuration operates in serial mode, i.e. "Dec1" processes data before "Dec2" starts its operation, and so on. An obvious extension of this configuration to three codes is shown in Fig. 5(a), which also operates in serial mode. But, with more than two codes, there are other possible configurations, as that shown in Fig. 5(b) where "Dec1" communicates with the other decoders, but these decoders do not exchange information among each other. This "Master & Slave" configuration operates in a mixed serial-parallel mode, since all other decoders except the first operate in parallel. Another possibility, shown in Fig. 5(c) is that all decoders operate in parallel at any given time. Note that self loops are not allowed in these structures since they cause degradation or divergence in the decoding process (positive feedback). We are not considering other possible hybrid configurations. Which configuration performs better? Our selection of the best configuration and its associated decoding rule is based on a detailed analysis of the minimum bit error decoding rule (MAP algorithm) as described below.

Figure 4: Decoding structure for two codes.

Turbo Decoding for Multiple Codes — Let u_k be a binary random variable taking values in $\{0, 1\}$, representing the sequence of information bits $\mathbf{u} = (u_1, \ldots, u_N)$. The MAP algorithm [7] provides the log likelihood ratio L_k given the received symbols $\mathbf{y}$:

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

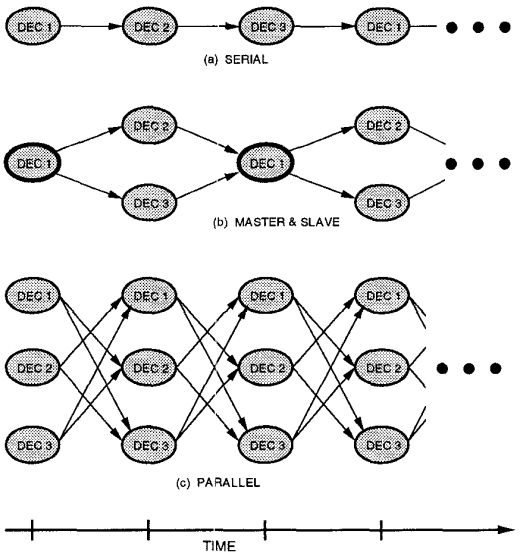

Figure 5: Different decoding structures for three codes.

$$= \log \frac{\sum_{\mathbf{u},u_k=1} P(\mathbf{y}|\mathbf{u}) \prod_{j\neq k} P(u_j)}{\sum_{\mathbf{u},u_k=0} P(\mathbf{y}|\mathbf{u}) \prod_{j\neq k} P(u_j)} + \log \frac{P(u_k=1)}{P(u_k=0)} \quad (2)$$

For efficient computation of eq.(2) when the a-priori probabilities $P(u_j)$ are non-uniform, the modified MAP algorithm in [2] is simpler to use than the version considered in [1]. Therefore, in this paper we use the modified MAP algorithm of [2] as we did in [4].

The channel model is shown in Fig. 6 where the n_{ik}'s and the n_{pk}'s are i.i.d. zero mean Gaussian random variables with unit variance, and $\rho = \sqrt{2rE_b/N_o}$ is the signal-to-noise ratio. (The same model is used for each encoder). To explain the basic decoding concept we restrict

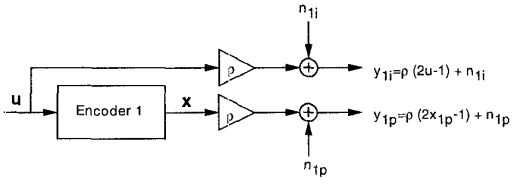

Figure 6: Channel model.

ourselves to three codes, but extension to several codes is straightforward. In order to simplify the notation, consider the combination of permuter and encoder as a block code with input $\mathbf{u}$ and outputs $\mathbf{x}_i$, $i = 1, 2, 3$ and the corresponding received sequences $\mathbf{y}_i, i = 1, 2, 3$. The optimum MAP decision metric on each bit is (for data with uniform probabilities)

$$L_k = \frac{\sum_{\mathbf{u},u_k=1} P(\mathbf{y}_1|\mathbf{u}) P(\mathbf{y}_2|\mathbf{u}) P(\mathbf{y}_3|\mathbf{u})}{\sum_{\mathbf{u},u_k=0} P(\mathbf{y}_1|\mathbf{u}) P(\mathbf{y}_2|\mathbf{u}) P(\mathbf{y}_3|\mathbf{u})} , \quad (3)$$

but in practice we cannot compute eq.(3) for large N. Suppose that we evaluate $P(\mathbf{y}_i|\mathbf{u})$, $i = 2, 3$, in eq.(3) using Bayes' rule and using the following approximation

$$P(\mathbf{u}|\mathbf{y}_i) \approx \prod_{k=1}^{N} \tilde{P}_i(u_k) \quad (4)$$

Note that $P(\mathbf{u}|\mathbf{y}_i)$ is not separable in general A reasonable criterion for this approximation is to choose $\prod_{k=1}^{N} \tilde{P}_i(u_k)$ such that it minimizes the Kullback distance or free energy [9, 10]. Define

$$\tilde{P}_i(u_k) = \frac{e^{u_k \tilde{L}_{ik}}}{1 + e^{\tilde{L}_{ik}}} , \quad (5)$$

where $u_k \in \{0, 1\}$. Then the Kullback distance is given by

$$F(\tilde{L}_i) = \sum_{\mathbf{u}} \frac{e^{\sum_{k=1}^{N} u_k \tilde{L}_{ik}}}{\prod_{k=1}^{N}(1 + e^{\tilde{L}_{ik}})} \log \frac{e^{\sum_{k=1}^{N} u_k \tilde{L}_{ik}}}{\prod_{k=1}^{N}(1 + e^{\tilde{L}_{ik}}) P(\mathbf{u}|\mathbf{y}_i)} \quad (6)$$

Such minimization involves forward and backward recursions analogous to the MAP decoding algorithm! Therefore, if such an approximation can be obtained, we can use it in eq.(3) for $i = 2$ and $i = 3$ (by Bayes' rule) to complete the algorithm. Now, instead of using eq.(6) to obtain $\{\tilde{P}_i\}$ or equivalently $\{\tilde{L}_i\}$, we use (4) and (5) for $i = 2, 3$ (by Bayes' rule) to express (3) as

$$L_k = f(\mathbf{y}_1, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k) + \tilde{L}_{2k} + \tilde{L}_{3k} \quad (7)$$

where

$$f(\mathbf{y}_1, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k) = \log \frac{\sum_{\mathbf{u},u_k=1} P(\mathbf{y}_1|\mathbf{u}) \prod_{j\neq k} e^{u_j(\tilde{L}_{2j}+\tilde{L}_{3j})}}{\sum_{\mathbf{u},u_k=0} P(\mathbf{y}_1|\mathbf{u}) \prod_{j\neq k} e^{u_j(\tilde{L}_{2j}+\tilde{L}_{3j})}} \quad (8)$$

We can use (4) and (5) again, but this time for $i = 1, 3$, to express (3) as

$$L_k = f(\mathbf{y}_2, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3, k) + \tilde{L}_{1k} + \tilde{L}_{3k} \quad (9)$$

and similarly

$$L_k = f(\mathbf{y}_3, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2, k) + \tilde{L}_{1k} + \tilde{L}_{2k} \quad (10)$$

A solution to eqs. 7,9, and 10 is

$$\tilde{L}_{1k} = f(\mathbf{y}_1, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k);\ \tilde{L}_{2k} = f(\mathbf{y}_2, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3, k);\ \tilde{L}_{3k} = f(\mathbf{y}_3, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2, k) \quad (11)$$

provided that a solution does indeed exist. for $k = 1, 2, \ldots, N$. The final decision is then based on

$$L_k = \tilde{L}_{1k} + \tilde{L}_{2k} + \tilde{L}_{3k} \quad (12)$$

which is passed through a hard-limiter with zero threshold. We attempted

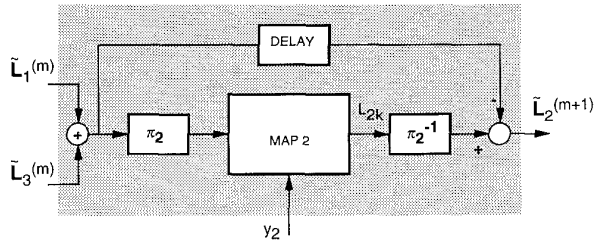

Figure 7: Structure of block decoder 2

to solve the nonlinear equations in (11) for $\tilde{\mathbf{L}}_1$, $\tilde{\mathbf{L}}_2$, and $\tilde{\mathbf{L}}_3$ by using the iterative procedure

$$\tilde{L}_{1k}^{(m+1)} = \alpha_1^{(m)} f(\mathbf{y}_1, \tilde{\mathbf{L}}_2^{(m)}, \tilde{\mathbf{L}}_3^{(m)}, k) \quad (13)$$

for $k = 1, 2, \ldots, N$, iterating on m. Similar recursions hold for $\tilde{L}_{2k}^{(m)}$ and $\tilde{L}_{3k}^{(m)}$. The gain $\alpha_1^{(m)}$ should be equal to one, but we noticed experimentally that better convergence can be obtained by optimizing this gain for each iteration starting from a value slightly less than one, and increasing toward

one with the iterations, as often done in simulated annealing methods. We start the recursion with the initial condition[2] $\tilde{\mathbf{L}}_1^{(0)} = \tilde{\mathbf{L}}_2^{(0)} = \tilde{\mathbf{L}}_3^{(0)} = \mathbf{0}$. For the computation of $f(\cdot)$ we use the modified MAP algorithm with permuters (direct and inverse) where needed, as shown in Fig. 7 for block decoder 2. The MAP algorithm always starts and ends at the all-zero state since we used perfect termination. Similar structures apply for block decoder 1 ($\pi_1 = I$, identity), and block decoder 3. The overall decoder is composed of block decoders connected as in Fig. 5(c), which can be implemented as a pipeline or by feedback.

Multiple Code Algorithm Applied to Two Codes. — For turbo codes with only two constituent codes, eq. (13) reduce to

$$\tilde{L}_{1k}^{(m+1)} = \alpha_1^{(m)} f(\mathbf{y}_1, \tilde{\mathbf{L}}_2^{(m)}, k) \quad k = 1, 2, \ldots, N$$

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

where, for each iteration, $\alpha_1^{(m)}$ and $\alpha_2^{(m)}$ can be optimized (simulated annealing) or set to 1 for simplicity. The decoding configuration for two codes, according to the previous section, is shown in Fig. 8. In this special case, since the two paths in Fig. 8 are disjoint, the decoder structure reduces to that of Fig. 4, i.e. to the serial mode.

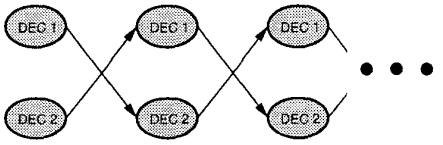

Figure 8: Parallel structure for two codes.

If we optimize $\alpha_1^{(m)}$ and $\alpha_2^{(m)}$, our method for two codes is similar to the decoding method proposed in [1], which requires estimates of the variances of $\tilde{L}_{1k}$ and $\tilde{L}_{2k}$ for each iteration in presence of errors. In the method proposed in [2] the received "systematic" observation was subtracted from $\tilde{L}_{1k}$, which results in performance degradation. In [3] the method proposed in [2] was used but the received "systematic" observation was interleaved and provided to decoder 2. In [4], we argued that there is no need to interleave the received "systematic" observation and provide it to decoder 2, since $\tilde{L}_{1k}$ does this job. It seems that our proposed method with $\alpha_1^{(m)}$ and $\alpha_2^{(m)}$ equal to 1 is the simplest and achieves the same performance reported in [3] for rate 1/2 codes.

Terminated Parallel Convolutional Codes as Block Codes. — Consider the combination of permuter and encoder as a linear block code. Define P_i as the parity matrix of the terminated convolutional code i. Then the overall generator matrix for three parallel codes is

$$G = [I \;\; P_1 \;\; \pi_2 P_2 \;\; \pi_3 P_3]$$

where π_i are the permutations (interleavers). In order to maximize the minimum distance of the code given by G we should maximize the number of linearly independent columns of the corresponding parity check matrix H. This suggests that the design of P_i (code) and π_i (permutation) are closely related and it does not necessarily follow that optimum component codes (maximum d_{min}) yield optimum parallel concatenated codes. For very small N we used this concept to design jointly the permuter and the component convolutional codes.

IV. Performance

Two Codes — The performance obtained by turbo decoding the code with two constituent codes $(1, g_b/g_a)$, where $g_a = (37)_{octal}$ and $g_b = (21)_{octal}$, and with random permutations of lengths $N = 4096$ and $N = 16384$ is compared in Fig. 9 to the capacity of a binary-input Gaussian channel for rate $r = 1/4$. The best performance curve in Fig. 9 is approximately 0.7 dB from the Shannon limit at BER=10^{-4}.

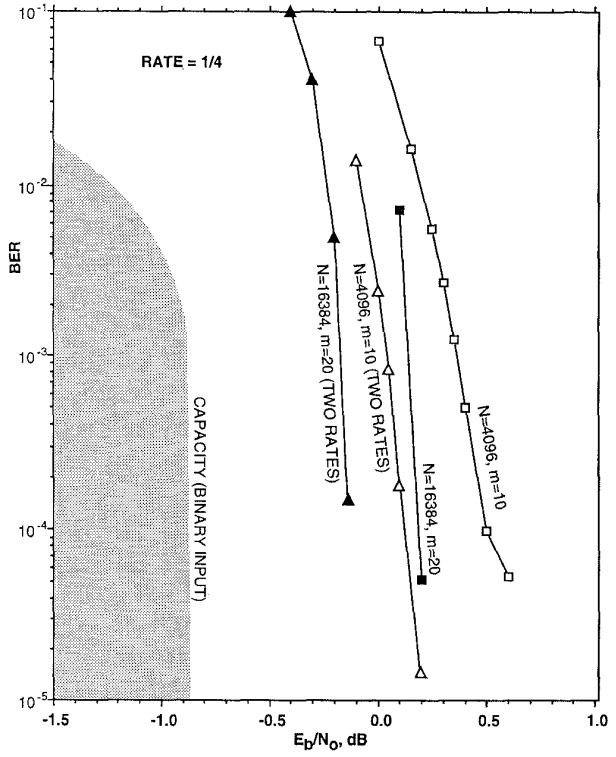

Figure 9: Turbo codes performance, $r = 1/4$

Unequal Rate Encoders — We now extend the results to encoders with unequal rates with two $K = 5$ constituent codes $(1, g_b/g_a, g_c/g_a)$ and (g_b/g_a), where $g_a = (37)_{octal}$, $g_b = (33)_{octal}$ and $g_c = (25)_{octal}$. This structure improves the performance of the overall, rate 1/4, code, as shown in Fig.9. This improvement is due to the fact that we can avoid using the interleaved information data at the second encoder and that the rate of the first code is lower than that of the second code. For PCS applications, short interleavers should be used, since the vocoder frame is usually 20ms. Therefore we selected 192 and 256 bits interleavers as an example, corresponding to 9.6 and 13 Kbps. (Note that this small difference of interleaver size does not affect significantly the performance). The performance of codes with short interleaver is shown in Fig. 10 for the $K = 5$ codes described above for random permutation and row-column permutation with $a = 2$ for rows and $a = 4$ for columns.

Three Codes — The performance of a three-code turbo code with random interleavers is shown in Fig. 11 for $N = 4096$. The three recursive codes shown in Fig. 1 where used for $K = 3$. Three recursive codes with $g_a = (13)_{octal}$ and $g_b = (11)_{octal}$ were used for $K = 4$. Note that the non-systematic version of this encoder is catastrophic, but the recursive systematic version is non-catastrophic. We found that this $K = 4$ code has better performance than several others.

Although it was suggested [5] that g_a be a primitive polynomial, we found several counterexamples that show better performance, e.g. g_a for $K = 5$ proposed in [1] is not primitive.

In Fig. 11 , the performance of the $K = 4$ code was improved by going to 30 iterations and using a S-random interleaver with $S = 31$. For shorter blocks (192 and 256), the results are shown in Fig. 10 where it

[2]Note that the components of the $\tilde{\mathbf{L}}$'s corresponding to the tail bits are set to zero for all iterations.

can be observed that approximately 1 dB SNR is required for BER=10^{-3}, which implies a CDMA capacity $C = 0.8\eta$ We have noticed that the slope of the BER curve changes around BER=10^{-5} (flattening effect) if the interleaver is not designed properly to maximize d_{min} or is chosen at random.

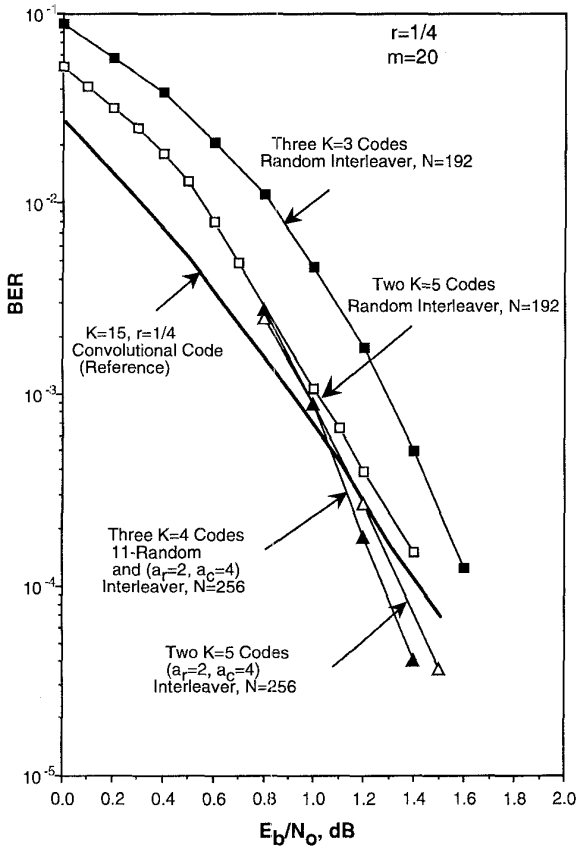

Figure 10: Performance with short block sizes.

V. Conclusions

We have shown how turbo codes and decoders can be used to improve the coding gain for PCS applications. These are just preliminary results that require extensive further analysis. In particular, we need to improve our understanding of the influence of the interleaver choice on the code performance, to explore the sensitivity of the decoder performance to the precision with which we can estimate E_b/N_o.

An interesting theoretical question is to determine "how random" these codes can be so as to draw conclusions on their performance based on comparison with random coding bounds. In [4] we obtained the complete weight distribution of a turbo code, calculated the upper bound on BER and compared it with maximum-likelihood (ML) decoding. Those results showed that the performance of turbo decoding is close to ML decoding and to optimum MAP decoding. However, the approximation used in eq.(4) implies that turbo decoding is only close to but not equal to MAP decoding.

VI. Acknowledgments

The authors are grateful to S. Dolinar and R.J. McEliece for their helpful comments.

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding: Turbo Codes." Proc. 1993 IEEE International Conference on Communications, pp. 1064–1070.

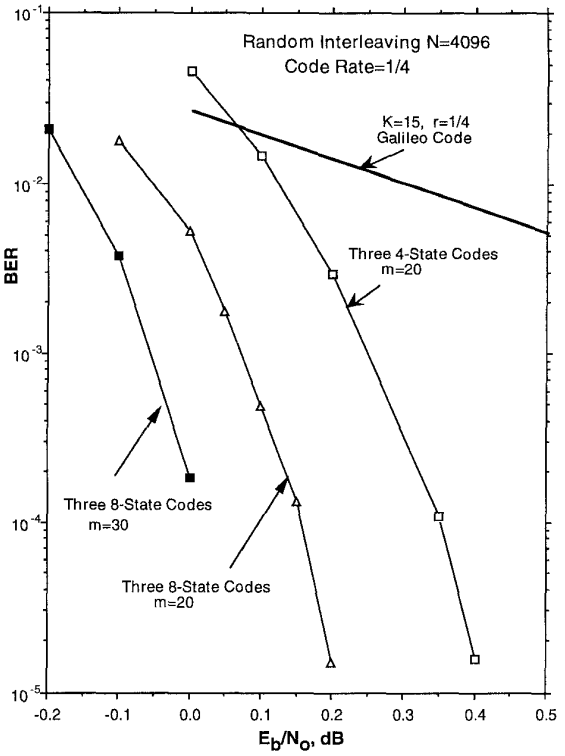

Figure 11: Three-code performance

[2] J. Hagenauer and P. Robertson, "Iterative (Turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms", Proc. of the ITG conference "Source and channel coding", Oct. 1994, Frankfurt.

[3] P. Robertson, "Illuminating the structure of code and decoder of parallel concatenated recursive systematic (Turbo) codes", Proceedings GLOBECOM '94, Dec. 1994, pp.1298-1303.

[4] D. Divsalar and F. Pollara, "Turbo Codes for Deep-Space Communications", JPL TDA Progress Report 42-120, Feb. 15, 1995.

[5] G. Battail, C. Berrou and A. Glavieux, "Pseudo-random recursive convolutional coding for near-capacity performance", Comm. Theory Mini-conference, GLOBECOM '93, Dec. 1993.

[6] D. Divsalar, S. Dolinar and F. Pollara, "Weight distribution of multiple turbo codes", JPL TDA Progress Report, (In preparation).

[7] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inform. Theory*, vol. IT-20 (1974), pp. 284–287.

[8] E. Dunscombe and F.C. Piper, " Optimal interleaving scheme for convolutional codes", Electronic Letters, 26 Oct. 1989, Vol. 25, No. 22, pp. 1517-1518.

[9] M. Moher, "Decoding via Cross-entropy Minimization", Proceedings GLOBECOM '93, Dec. 1993, p.809-813.

[10] G. Battail and R. Sfez, "Suboptimum Decoding using the Kullback Principle", Lecture Notes in Computer Science, Vol. 313, pp. 93-101, 1988.