Multiple Turbo Codes for Deep-Space Communications

D. Divsalar and F. Pollara
Communications Systems Research Section

In this article, we introduce multiple turbo codes and a suitable decoder structure derived from an approximation to the maximum a posteriori probability (MAP) decision rule, which is substantially different from the decoder for two-code-based encoders. We analyze the effect of interleaver choice on the weight distribution of the code, and we describe simulation results on the improved performance of these new codes.

I. Introduction

Coding theorists have traditionally attacked the problem of designing good codes by developing codes with a lot of structure, which lends itself 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.

It is claimed these codes 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 a bit error rate (BER) of 10^{-5} [1]. However, some important details that are necessary to reproduce these results were omitted. The purpose of this article is to shed some light on the accuracy of these claims and to extend these results to multiple turbo codes with more than two component codes.

The original turbo decoder scheme, for two component codes, operates in serial mode. For multiple-code turbo codes, we found that the decoder, based on the optimum maximum a posteriori (MAP) rule, must operate in parallel mode, and we derived the appropriate metric, as illustrated in Section III.

II. Parallel Concatenation of Convolutional Codes

The codes considered in this article consist of the parallel concatenation of multiple convolutional codes with random interleavers (permutations) at the input of each encoder. This extends the analysis reported in [4], which considered turbo codes formed from just two constituent codes. Figure 1 illustrates

Acknowledgments

The author would like to thank Vince Pollmeier, Sam Thurman, and Pieter Kallemeyn for their valuable suggestions and insights for this study. Also, Peter Wolff's help in using the MIRAGE software set is deeply appreciated.

References

[1] T. W. Hamilton and W. G. Melbourne, "Information Content of a Single Pass of Doppler Data From a Distant Spacecraft," *JPL Space Programs Summary 37-39, March–April 1966*, vol. III, pp. 18–23, May 31, 1966.

[2] S. W. Thurman, "Deep-Space Navigation With Differenced Data Types Part I: Differenced Range Information Content," *The Telecommunications and Data Acquisition Progress Report 42-103, July–September 1990*, Jet Propulsion Laboratory, Pasadena, California, pp. 47–60, November 15, 1990.

[3] J. R. Guinn and P. J. Wolff, "TOPEX/Poseidon Operational Orbit Determination Results Using Global Positioning Satellites," AAS Paper 93-573, presented at the AAS/AIAA Astrodynamics Specialist Conference, Victoria, British Columbia, Canada, August 16–19, 1993.

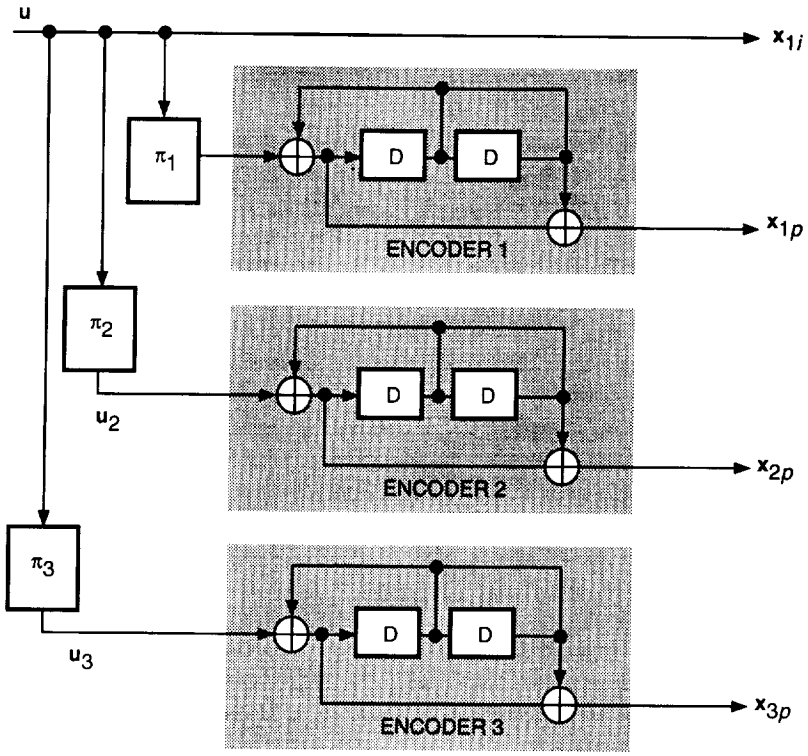

Fig. 1. Example of encoder with three codes.

a particular example that will be used in this article to verify the performance of these codes. The 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 and may not have identical code rates. The first component encoder operates directly (or through π_1) on the information bit sequence $\mathbf{u} = (u_1, \cdots, u_N)$ of length N, producing the two output sequences $\mathbf{x}_{1i}$ and $\mathbf{x}_{1p}$. 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 $\mathbf{x}_{2p}$. Similarly, subsequent component encoders operate on a reordered sequence of information bits, $\mathbf{u}_j$, produced by interleaver π_j, and output the sequence $\mathbf{x}_{jp}$. The interleaver is a pseudorandom 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 component codes with $M_1 = M_2 = M_3 = M = 2$, producing the outputs $\mathbf{x}_{1i} = \mathbf{u}$, $\mathbf{x}_{1p} = \mathbf{u} \cdot g_b/g_a$, $\mathbf{x}_{2p} = \mathbf{u}_2 \cdot g_b/g_a$, and $\mathbf{x}_{3p} = \mathbf{u}_3 \cdot g_b/g_a$ (here π_1 is assumed to be an identity, i.e., no permutation), 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 $\mathbf{x}_{1p}$, $\mathbf{x}_{2p}$, $\mathbf{x}_{3p}$, and even $\mathbf{x}_{1i}$ if the decoder works (for an example, see Section IV). 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_a is a primitive polynomial.

We use the encoder in Fig. 1 to generate an $(n(N+M), N)$ 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 all component encoders with M predetermined tail bits. This issue, which had not been resolved in previously proposed turbo code implementations, can be dealt with by applying the method described in [4], which is valid for any number of component codes.

A. 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 $\mathbf{u} = (00\cdots0000100\cdots000)$ 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 $\mathbf{u} = (00\cdots00100100\cdots000)$ generates a low-weight codeword with weight 6 for the first encoder. If the interleavers do not "break" this input pattern, the resulting codeword's 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. By contrast, if the number of zeros between the ones is not of this form, the encoded output is nonterminating until the end of the block, and its encoded weight is very large unless the sequence occurs near the end of the block.

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 nonterminating (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 self-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 the proper choice of an 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, $n=3,4,5,\cdots$, data sequences that produce self-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 nonterminating output from at least one of the encoders. Thus, turbo code structures that would have low minimum distances if unpermuted can still perform well if the low-weight codewords of the component codes are produced by input sequences with weight higher than two.

B. 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 $(\cdots 001001000\cdots)$, which corresponds to a low-weight codeword in each of the encoders of Fig. 1. If we randomly select 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 $(\cdots 001001000\cdots)$ 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 $(\cdots 001001000\cdots)$ 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 that may also produce low-weight codewords, e.g., $(\cdots 00100(000)^t 1000\cdots)$, for the code in Fig. 1. For comparison, let us consider a weight-3 data sequence such as $(\cdots 0011100\cdots)$, 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 $(\cdots 0011100\cdots)$ 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 three codes and two random interleavers, this probability is even smaller, $1-\left[1-(6/N^2)^2\right]^N \approx 36/N^3$. 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 $(\cdots 001001000\cdots)$ than from the weight-3 sequence $(\cdots 0011100\cdots)$ that produces the minimum distance in the unpermuted version of the same code. Higher weight sequences have an even smaller probability of reproducing themselves after being passed through the random interleavers.

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 sequences have successively decreasing importance. Also, increasing the number of codes and, correspondingly, the number of interleavers, makes it more and more likely that the bad input sequences will be broken up by one or more of the permutations.

The minimum distance is not the most important characteristic of the turbo code, except for its asymptotic performance, at very high E_b/N_o. At moderate signal-to-noise ratios (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).

C. Design of Nonrandom and Partially Random Interleavers

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. 2 cannot be broken

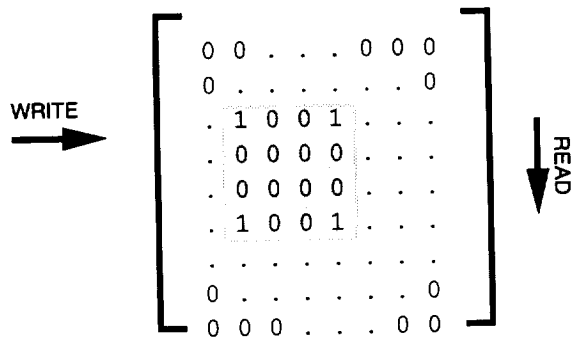

Fig. 2. Example where a block interleaver fails to "break" the input sequence.

by a block interleaver. In order to break such sequences, random interleavers are desirable, as discussed above. (A method for the design of nonrandom 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 [7]. 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. 2, 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 is it possible to break the low-weight sequences. The method in [7] can be used again for the permutation of rows. Appropriate selection 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 have also designed semirandom permutations (interleavers) by generating random integers i, $1 \leq i \leq N$, without replacement. We define an "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. The searching time for this algorithm increases with S and is not guaranteed to finish successfully. However, we have observed that choosing $S < \sqrt{N/2}$ usually produces a solution in a reasonable time. Note that for $S = 1$, we have a purely random interleaver. In the simulations, we used $S = 31$ with block size $N = 4096$.

III. Turbo Decoding for Multiple Codes

In this section, we consider decoding algorithms for multiple-code turbo codes. In general, 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 broken 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.

A. Turbo Decoding Configurations

The turbo decoding configuration proposed in [1] for two codes is shown schematically in Fig. 3. This configuration operates in serial mode, i.e., "Dec 1" processes data before "Dec 2" starts its operation, and so on. An obvious extension of this configuration to three codes is shown in Fig. 4(a), which also operates in serial mode. But, with more than two codes, there are other possible configurations, such as that shown in Fig. 4(b), where "Dec 1" communicates with the other decoders, but these decoders do

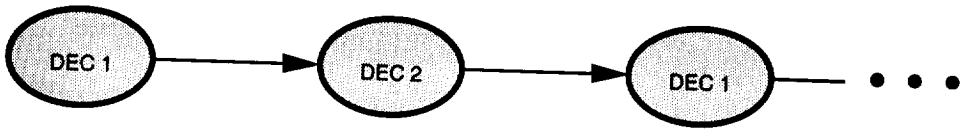

Fig. 3. Decoding structure for two codes.

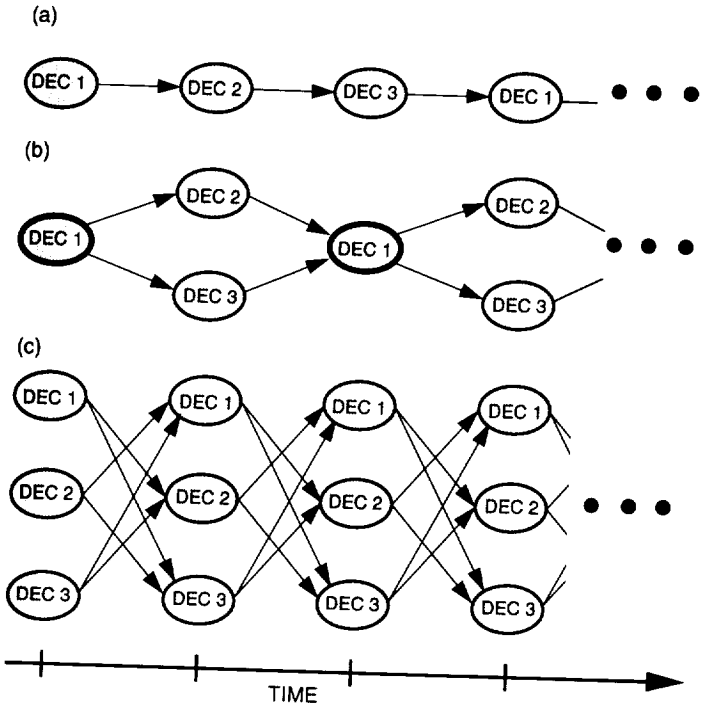

Fig. 4. Different decoding structures for three codes: (a) serial, (b) master and slave, and (c) parallel.

not exchange information between each other. This "master and slave" configuration operates in a mixed serial–parallel mode, since all other decoders except the first operate in parallel. Another possibility, shown in Fig. 4(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.

B. Turbo Decoding for Multiple Codes

Let u_k be a binary random variable taking values in $\{0, 1\}$, representing the sequence of information bits $\mathbf{u} = (u_1, \cdots, u_N)$. The MAP algorithm [6] provides the log likelihood ratio L_k, given the received symbols $\mathbf{y}$:

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

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

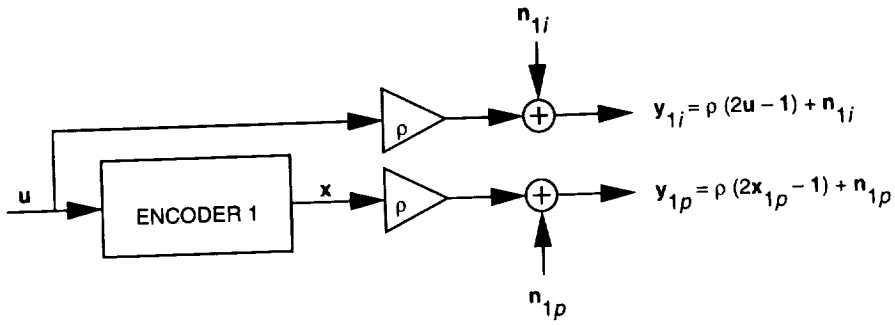

Fig. 5. Channel model.

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

The channel model is shown in Fig. 5, where the n_{ik}'s and the n_{pk}'s are independent identically distributed (i.i.d.) zero-mean Gaussian random variables with unit variance, and $\rho = \sqrt{2rE_b/N_o}$ is the SNR. The same model is used for each encoder. To explain the basic decoding concept, we restrict ourselves to three codes, but extension to several codes is straightforward. In order to simplify the notation, consider the combination of permuter and encoder as a block code with input $\mathbf{u}$ and outputs $\mathbf{x}_i$, $i = 0, 1, 2, 3 (\mathbf{x}_0 = \mathbf{u})$ and the corresponding received sequences $\mathbf{y}_i$, $i = 0, 1, 2, 3$. The optimum bit decision metric on each bit is (for data with uniform a priori probabilities)

$$L_k = \log \frac{\sum_{\mathbf{u}:u_k=1} P(\mathbf{y}_0|\mathbf{u})P(\mathbf{y}_1|\mathbf{u})P(\mathbf{y}_2|\mathbf{u})P(\mathbf{y}_3|\mathbf{u})}{\sum_{\mathbf{u}:u_k=0} P(\mathbf{y}_0|\mathbf{u})P(\mathbf{y}_1|\mathbf{u})P(\mathbf{y}_2|\mathbf{u})P(\mathbf{y}_3|\mathbf{u})} \tag{3}$$

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

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

Note that $P(\mathbf{u}|\mathbf{y}_i)$ is not separable in general. However, for $i = 0$, $P(\mathbf{u}|\mathbf{y}_0)$ is separable; hence, Eq. (4) holds with equality. If such an approximation, i.e., Eq. (4), can be obtained, we can use it in Eq. (3) for $i = 2$ and $i = 3$ (by Bayes' rule) to complete the algorithm. 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 [8,9]. Define $\tilde{L}_{ik}$ by

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

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

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

Minimizing $F(\tilde{\mathbf{L}}_i)$ involves forward and backward recursions analogous to the MAP decoding algorithm, but we have not attempted this approach in this work. Instead of using Eq. (6) to obtain $\{\tilde{P}_i\}$ or, equivalently, $\{\tilde{L}_{ik}\}$, we use Eqs. (4) and (5) for $i = 0, 2, 3$ (by Bayes' rule) to express Eq. (3) as

$$L_k = f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k) + \tilde{L}_{0k} + \tilde{L}_{2k} + \tilde{L}_{3k} \tag{7}$$

where $\tilde{L}_{0k} = 2\rho \mathbf{y}_{0k}$ and

$$f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k) = \log \frac{\sum_{\mathbf{u}:u_k=1} P(\mathbf{y}_1|\mathbf{u}) \prod_{j\neq k} e^{u_j(\tilde{L}_{0j}+\tilde{L}_{2j}+\tilde{L}_{3j})}}{\sum_{\mathbf{u}:u_k=0} P(\mathbf{y}_1|\mathbf{u}) \prod_{j\neq k} e^{u_j(\tilde{L}_{0j}+\tilde{L}_{2j}+\tilde{L}_{3j})}} \tag{8}$$

We can use Eqs. (4) and (5) again, but this time for $i = 0, 1, 3$, to express Eq. (3) as

$$L_k = f(\mathbf{y}_2, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3, k) + \tilde{L}_{0k} + \tilde{L}_{1k} + \tilde{L}_{3k} \tag{9}$$

and similarly,

$$L_k = f(\mathbf{y}_3, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2, k) + \tilde{L}_{0k} + \tilde{L}_{1k} + \tilde{L}_{2k} \tag{10}$$

A solution to Eqs. (7), (9), and (10) is

$$\tilde{L}_{1k} = f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2, \tilde{\mathbf{L}}_3, k);\ \tilde{L}_{2k} = f(\mathbf{y}_2, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_3, k);\ \tilde{L}_{3k} = f(\mathbf{y}_3, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_1, \tilde{\mathbf{L}}_2, k) \tag{11}$$

for $k = 1, 2, \cdots, N$, provided that a solution to Eq. (11) does indeed exist. The final decision is then based on

$$L_k = \tilde{L}_{0k} + \tilde{L}_{1k} + \tilde{L}_{2k} + \tilde{L}_{3k} \tag{12}$$

which is passed through a hard limiter with zero threshold. We attempted to solve the nonlinear equations in Eq. (11) for $\tilde{\mathbf{L}}_1$, $\tilde{\mathbf{L}}_2$, and $\tilde{\mathbf{L}}_3$ by using the iterative procedure

$$\tilde{L}_{1k}^{(m+1)} = \alpha_1^{(m)} f(\mathbf{y}_1, \tilde{\mathbf{L}}_0, \tilde{\mathbf{L}}_2^{(m)}, \tilde{\mathbf{L}}_3^{(m)}, k) \tag{13}$$

for $k = 1, 2, \cdots, N$, 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 is often done in simulated annealing methods. We start the recursion with the initial condition[1] $\tilde{\mathbf{L}}_1^{(0)} = \tilde{\mathbf{L}}_2^{(0)} = \tilde{\mathbf{L}}_3^{(0)} = \tilde{\mathbf{L}}_0$. For the computation of $f(\cdot)$, we use the modified MAP algorithm as described in [4] with permuters (direct and inverse) where needed, as shown in Fig. 6 for block decoder 2. The MAP algorithm always starts and ends at the all-zero state since we always terminate the trellis as described in [4]. Similar structures apply for block decoder 1 (we assumed $\pi_1 = I$ identity; however, any π_1 can be used) and block decoder 3. The overall decoder is composed of block decoders

[1] Note that the components of the $\tilde{\mathbf{L}}_i$'s corresponding to the tail bits, i.e., $\tilde{L}_{ik}$, for $k = N+1, \cdots, N+M$, are set to zero for all iterations.

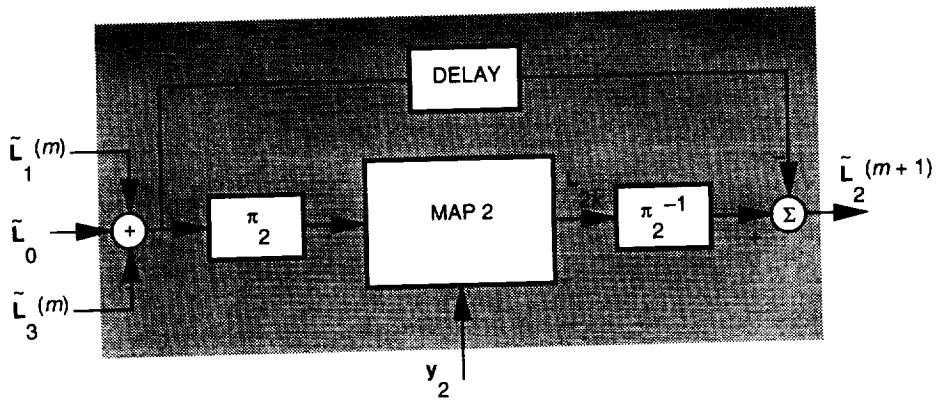

Fig. 6. Structure of block decoder 2.

connected as in Fig. 4(c), which can be implemented as a pipeline or by feedback. We proposed an alternative version of the above decoder in [10]. At this point, further approximation for turbo decoding is possible if one term corresponding to a sequence $\mathbf{u}$ dominates other terms in the summation in the numerator and denominator of Eq. (8). Then the summations in Eq. (8) can be replaced by "maximum" operations with the same indices, i.e., replacing $\sum_{\mathbf{u}:u_k=i}$ with $\max_{\mathbf{u}:u_k=i}$ for $i = 0, 1$. A similar approximation can be used for $\tilde{L}_{2k}$ and $\tilde{L}_{3k}$ in Eq. (11). This suboptimum decoder then corresponds to a turbo decoder that uses soft output Viterbi (SOVA)-type decoders rather than MAP decoders.

C. Multiple-Code Algorithm Applied to Two Codes

For turbo codes with only two constituent codes, Eq. (13) reduces to

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

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

for $k = 1, 2, \cdots, N$ and $m = 1, 2, \cdots$, where, for each iteration, $\alpha_1^{(m)}$ and $\alpha_2^{(m)}$ can be optimized (simulated annealing) or set to 1 for simplicity. The decoding configuration for two codes, according to the previous section, is shown in Fig. 7. In this special case, since the two paths in Fig. 7 are disjoint, the decoder structure reduces to duplicate copies of the structure in Fig. 3 (i.e., to the serial mode).

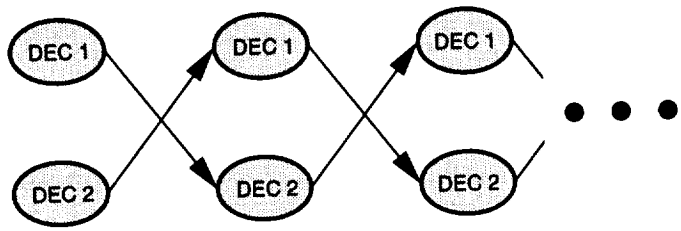

Fig. 7. Parallel structure for two codes.

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 the 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 $\tilde{L}_{0k}$ 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.

D. 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

$$G = [I \quad \pi_1 P_1 \quad \pi_2 P_2 \quad \pi_3 P_3]$$

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 and Simulation Results

For comparison with the new results on three-code turbo codes, we reproduce in Fig. 8 the performance obtained in [4] by using two-code $K = 5$ turbo codes with generators $(1, g_b/g_a)$, where $g_a = (37)_{octal}$ and $g_b = (21)_{octal}$, and with random permutations of lengths $N = 4096$ and $N = 16384$. The best performance curve in Fig. 8 is approximately 0.7 dB from the Shannon limit at BER $= 10^{-4}$. We also repeat for comparison in Fig. 8 the results obtained in [4] by using 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 $g_c = (25)_{octal}$. To show that it is possible not to send uncoded information for both codes, we used an overall rate 1/2 turbo code using two codes with $K = 2$ (differential encoder) with generator (g_b/g_a), where $g_a = (3)_{octal}$ and $g_b = (1)_{octal}$, and a $K = 5$ code with generator (g_b/g_a), where $g_a = (23)_{octal}$ and $g_b = (33)_{octal}$. A bit error rate of 10^{-5} was achieved at BSNR = 0.85 dB using an S-random permutation of length $N = 16{,}384$ with $S = 40$.

A. Three Codes

The performance of two different three-code turbo codes with random interleavers is shown in Fig. 9 for $N = 4096$. The first code uses three recursive codes shown in Fig. 1 with constraint length $K = 3$. The second code uses three recursive codes with $K = 4$, $g_a = (13)_{octal}$, and $g_b = (11)_{octal}$. Note that the nonsystematic version of the second encoder is catastrophic, but the recursive systematic version is noncatastrophic. We found that this $K = 4$ code has better performance than several others.

As seen in Fig. 9, the performance of the $K = 4$ code was improved by going from 20 to 30 iterations. We found that the performance could also be improved by using an S-random interleaver with $S = 31$.

V. Conclusions

We have shown how three-code turbo codes and decoders can be used to further improve the coding gain for deep-space applications as compared with the codes studied in [4]. These are just preliminary results that require extensive further analysis. In particular, we need to improve our understanding of the influence of the interleaver design on the code performance and to analyze how close the proposed decoding algorithm is to maximum-likelihood or MAP decoding.

These new codes offer better performance than the large constraint-length convolutional codes employed by current missions and, most importantly, achieve these gains with much lower decoding complexity.

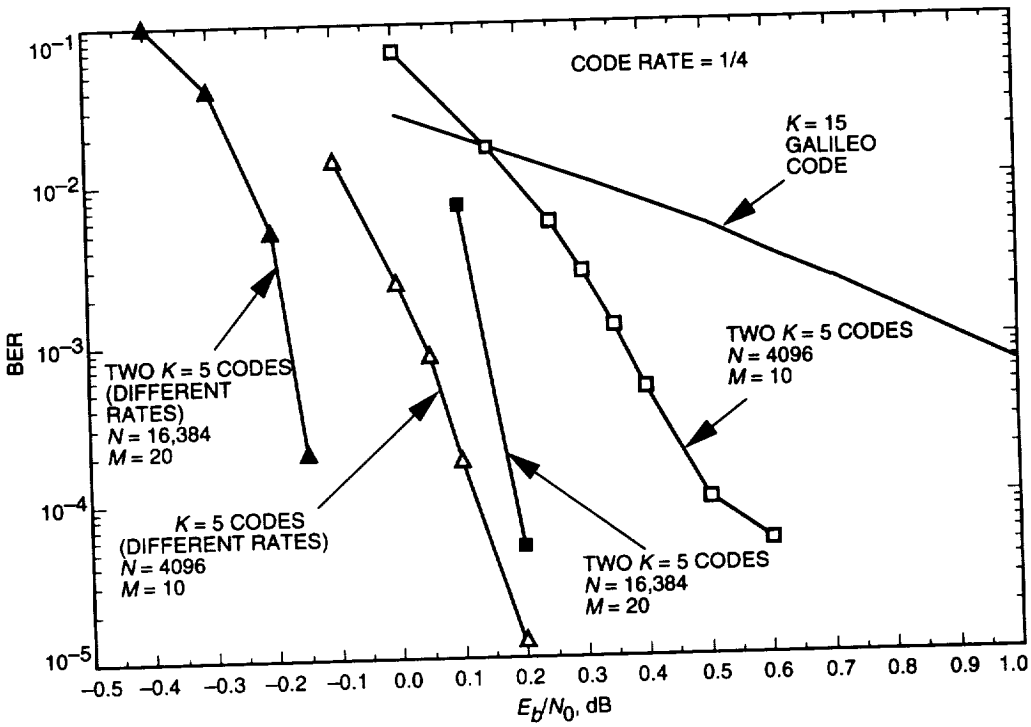

Fig. 8. Two-code performance, $r = 1/4$.

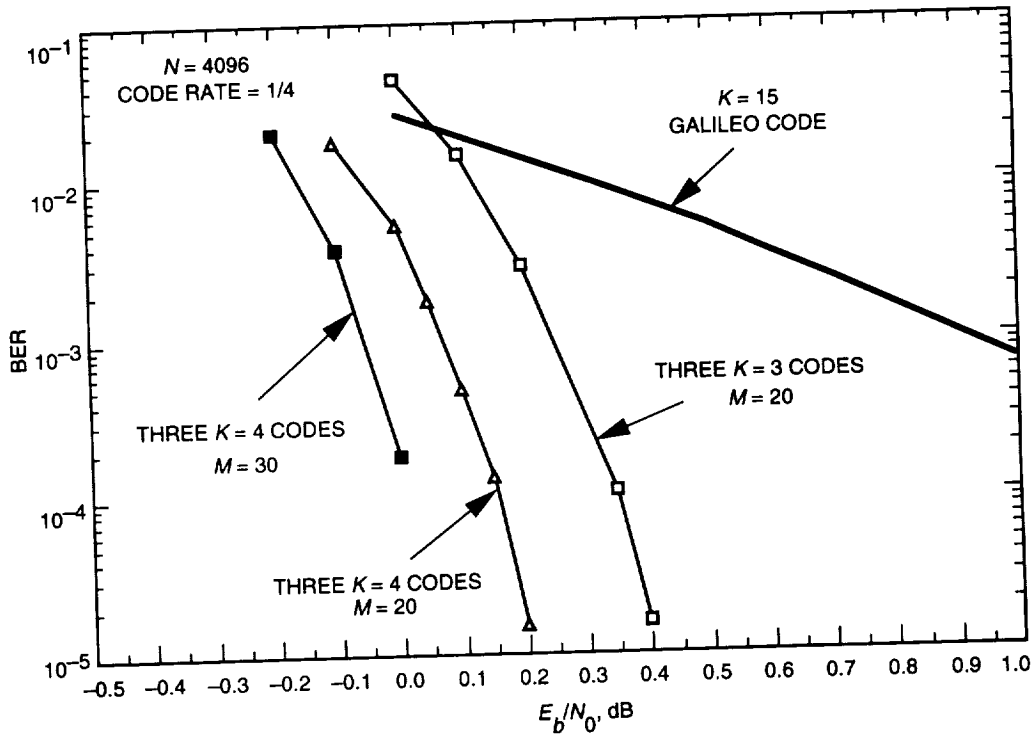

Fig. 9. Three-code performance, $r = 1/4$.

Acknowledgments

The authors are grateful to S. Dolinar for his contributions to the study of the weight distribution and interleavers[2] and to R. J. McEliece for helpful comments throughout this study.

References

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

[2] J. Hagenauer and P. Robertson, "Iterative (Turbo) Decoding of Systematic Convolutional Codes With the MAP and SOVA Algorithms," *Proc. of the ITG Conference on Source and Channel Coding*, Frankfurt, Germany, October 1994.

[3] P. Robertson, "Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) Codes, *Proceedings GLOBECOM '94*, San Francisco, California, pp. 1298–1303, December 1994.

[4] D. Divsalar and F. Pollara, "Turbo Codes for Deep-Space Communications," *The Telecommunications and Data Acquisition Progress Report 42-120, October–December 1994*, Jet Propulsion Laboratory, Pasadena, California, pp. 29–39, February 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*, Houston, Texas, December 1993.

[6] 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, pp. 284–287, 1974.

[7] E. Dunscombe and F. C. Piper, " Optimal Interleaving Scheme for Convolutional Codes," *Electronic Letters*, vol. 25, no. 22, pp. 1517–1518, October 26, 1989.

[8] M. Moher, "Decoding Via Cross-Entropy Minimization," *Proceedings GLOBECOM '93*, pp. 809–813, December 1993.

[9] G. Battail and R. Sfez, "Suboptimum Decoding Using the Kullback Principle," *Lecture Notes in Computer Science*, vol. 313, pp. 93–101, 1988.

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

[2] More detailed results are given in S. Dolinar and D. Divsalar, "Weight Distributions for Turbo Codes Using Random and Non-Random Permutations," JPL Interoffice Memorandum 331-95.2-016 (internal document), Jet Propulsion Laboratory, Pasadena, California, March 15, 1995.

Degradation in Finite-Harmonic Subcarrier Demodulation

Y. Feria and S. Townes
Communications Systems Research Section

T. Pham
Telecommunications Systems Section

Previous estimates on the degradations due to a subcarrier loop assume a square-wave subcarrier. This article provides a closed-form expression for the degradations due to the subcarrier loop when a finite number of harmonics are used to demodulate the subcarrier, as in the case of the buffered telemetry demodulator. We compared the degradations using a square wave and using finite harmonics in the subcarrier demodulation and found that, for a low loop signal-to-noise ratio, using finite harmonics leads to a lower degradation. The analysis is under the assumption that the phase noise in the subcarrier (SC) loop has a Tikhonov distribution. This assumption is valid for first-order loops.

I. Introduction

In an imperfect subcarrier demodulation, the difference between the phase of the reference signal and that of the subcarrier of the received signal causes the signal power to degrade while the noise power remains the same. This degradation is measured as the ratio of the reduced symbol energy-to-noise density ratio (E_s/N_0), or symbol signal-to-noise ratio (SNR), to the symbol SNR of an ideal demodulation where the phase difference is zero. The degradations due to the subcarrier loop were previously computed assuming a square wave [3]. This assumption is inappropriate in the case where only a finite number of harmonics of the subcarrier are there to be demodulated, as in the buffered telemetry demodulator (BTD) [2]. This article provides a closed-form expression for computing the degradation due to a finite-harmonic subcarrier tracking loop. Numerically, we found that, for low loop SNR cases, we actually have less degradation using a finite number of harmonics than using "all" the harmonics, namely, the square wave. The degradation due solely to the subcarrier loop using four harmonics is 0.15 to 0.3 dB lower than that using a square wave for loop SNRs in the range of 14 to 30 dB.

At first glance, the above may seem to contradict the intuition that the more harmonics we use, the higher the SNR we should get. This intuition is correct when the loop SNR is high, that is, when the jitter of the phase difference (between the true and the reference phases) is low. At low loop SNRs, however, we have a different scenario.

To explain this, let us first take a look at how the subcarriers are demodulated. A square-wave subcarrier is demodulated by multiplying the received signal by a square-wave reference signal. When we only have a finite number of harmonics of the square-wave subcarrier, the current design for the BTD [2]