Low-Rate Turbo Codes for Deep-Space Communications [1]

D. Divsalar and F. Pollara

Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109

Turbo codes were recently proposed by Berrou, Glavieux and Thitimajshima [2] and claimed to achieve near Shannon-1imit error correction performance with relatively simple component codes and large interleaves. A required E_b/N_o of 0.7 dB was reported for BER of 10^{-5}, using a rate 1/2 turbo code [2]. However, some important details that are necessary to reproduce these results were omitted. This article confirms the accuracy of these claims, and presents a complete description of an encoder/decoder pair that could be suitable for deep-space applications, where lower rate codes can be used. We describe a new simple method for trellis termination, we analyze the effect of interleave choice on the weight distribution of the code, and we introduce the use of unequal rate and multiple component codes.

The codes considered in this article consist of the parallel concatenation of two or more convolutional codes with a random interleave between each encoder. Fig, l(a) illustrates a particular example that will be used in this article to verify the performance of these codes. This encoder contains two recursive binary convolutional encoders, with M_1 and M_2 memory cells respectively. In general, the two component encoders may not be identical. The first component encoder operates directly on the information bit sequence u = $(u_1, \ldots . u_N)$ 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 u' produced by an interleaver of length N, and outputs the two sequences $\mathbf{x}_{2i}$ and $\mathbf{x}_{2p}$. 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 interleave and read out in a specified permuted order. Figure 1(a) shows an example where a rate $r = 1/n = 1/4$ code is generated by two component codes with $M_1 = M_2 = M = 4$, producing the outputs $\mathbf{x}_{1i} = \mathbf{u}$, $\mathbf{x}_{1p} = \mathbf{u} \cdot \frac{g_a}{g_b}$, $\mathbf{x}_{2i} = \mathbf{u}'$ and $\mathbf{x}_{2p} = \mathbf{u}' \cdot \frac{g_a}{g_b}$, where the generator polynomials g_a and g_b have octal representation 21 and 37, respectively. Note that various code rates can be obtained by puncturing the outputs.

Trellis Termination — We use the encoder in Fig, 1 (a) to generate a (n (N + *M*), *N*) block code, 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 the same *M* bits. Fortunately, the simple stratagem illustrated in Fig. 1 (b) is sufficient to terminate the trellis. 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.

Weight Distribution — In order to estimate the performance of a code it is necessary to have information about its minimum distance *d*, weight distribution, or actual code geometry, depending on the accuracy required for the bounds or approximations. The example of turbo code shown in Fig. 1 (a) produces two sets of codewords xl = $(x_{1i}, \mathbf{x}_{1p})$ and $x_2 = (x_{2i}, \mathbf{x}_{2p})$, whose weights can be easily computed. The challenge is in finding the pairing of codewords from each set, induced by a particular interleave. Intuitively, wc would like to avoid pairing low-weight codewords from one encoder with low-weight words from the other encoder. Many such pairings can be avoided by proper design of the interleaver. However, if the encoders are not recursive, the low-weight codeword generated by the input sequence **u = (00. . . 0000100 . . . 000)** with a single "1" will always appear again in the second encoder, for any choice of interleave. 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 u = (00. . .0010000100 . . . 000)

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

generates the minimum weight codeword (weight=6). If the interleaver does not properly "break" this input pattern, the resulting minimum distance will be 12.

However, the minimum distance is not the most important quantity of the code, except for its asymptotic performance, at very high E_b/N_o. At moderate SNRs, the weight distribution at the first several possible weights is necessary to compute the code per formance. Estimating the complete weight distribution for large N is still an open problem for these codes. We have investigated the effect of the interleaver on the weight distribution on a small-scale example where N = 16. This yields an (80,16) code whose weight distribution can be found by exhaustive enumeration. A good choice of the interleave can increase the minimum distance from 12 to 16, and, more importantly, can reduce the count of codewords at low weights. We have computed the weight distribution obtained by using no interleaver, a reverse permutation, and a 4 x 4 block interleaver, all with d = 12. Better weight distributions are obtained by the "random" permutation {2,13,0,3,1 1 ,15,6,14,8,9,10,4,12,1 ,7,5) with d = 12, and by the best found permutation {12,13,14,9,1 1,1 5,7,6,10,3,8,4,0,1,2,5} with d = 16 (The best known (80,16) linear block code has minimum distance 28). For interleaver length N = 1024 we were only able to enumerate all codewords produced by input sequences with weights 1, 2, and 3. This again confirmed the importance of the interleave choice for reducing the number of low-weight codewords. Better weight distributions were obtained by using "random" permutations than by structured permutations, as block or reverse permutations.

For the (80,1 6) code using the best found permutation we have compared the performance of a maximum likelihood decoder (obtained by simulation) to that of a turbo decoder with 10 iterations described later, and to the union bound computed from the weight distribution. The performance of the turbo decoder is only slightly suboptimum.

Turbo Decoding— Let u_k be a binary random variable taking values in {-I-1, – 1 }, representing the sequence of information bits. The MAP algorithm [1] provides the log likelihood ratio L(k) = log $\frac{P(u_k=+1|\mathbf{y})}{P(u_k=-1|\mathbf{y})}$ given the received symbols y. The sign of $L(k)$ is an estimate $\hat{u}_k$ of u_k and the magnitude $|L(k)|$ is the reliability of this estimate, as suggested in [3].

The channel model is shown in Fig. 2 where the n_{1ik}'s and the n_{1pk}'s are i.i.d. zero mean Gaussian random variables with unit variance, and $\rho = \sqrt{2E_s/N_o} = \sqrt{2r}E_b/N_o$ is the signal-to-noise ratio. A similar model applies for encoder 2.

Given the turbo code structure in Fig. 1(a), the optimum decoding rule maximizes either $P(u_k|\mathbf{y}_1,\mathbf{y}_2)$ (Minimum bit error probability rule) or $P(\mathbf{u}|\mathbf{y}_1,\mathbf{y}_2)$ (Maximum likelihood sequence rule). Since this rule is obviously too complex to compute., we resort to a suboptimum decoding rule [2,3] that uses separately the two observations y_1 and y_2, as shown in Fig, 3. Each decoder in Fig. 3 computes the a posteriori probabilities $P(u_k|\mathbf{y}_i,\tilde{\mathbf{u}}_i), i$ = 1,2 (see Fig. 4a), or equivalently the log-likelihood ratio $L_i(k) = \log \frac{P(u_k=+1|\mathbf{y}_i,\tilde{\mathbf{u}}_i)}{P(u_k=-1|\mathbf{y}_i,\tilde{\mathbf{u}}_i)}$ where $\tilde{\mathbf{u}}_1$ is provided by decoder 2 and $\tilde{\mathbf{u}}_2$ is provided by decoder 1 (see Fig. 4b). The quantities' $\tilde{\mathbf{u}}_i$ correspond to "new data estimates", "Innovations" or "extrinsic information" provided by decoders 1 and 2, that can be used to generate a priori probabilities on the information sequence $\mathbf{u}$ for branch metric computation in each decoder.

The question is how to generate the probabilities $P(\tilde{u}_{i,k}|u_k)$ that should be used for computation of the branch transition probabilities in MAP decoding. It can be shown that the probabilities P $(u_k|\tilde{u}_{i,k})$ or equivalently log $\frac{P(u_k=+1|\tilde{u}_{i,k})}{P(u_k=-1|\tilde{u}_{i,k})}$, i = =], 2, can be used instead of $P(\tilde{u}_{i,k}|u_k)$ for branch metric computations in the decoders. When decoder 1 generates $P(u_k|\tilde{u}_{2,k})$ or log $\frac{P(u_k=+1|\tilde{u}_{2,k})}{P(u_k=-1|\tilde{u}_{2,k})}$ for decoder 2, this quantity should not include the contribution due to $\tilde{u}_{1,k}$ which has been already generated by decoder 2. Thus we should have

$$\log \frac{P(u_k=+1|\tilde{u}_{2,k})}{P(u_k=-1|\tilde{u}_{2,k})} = \log \frac{P(u_k=+1|\mathbf{y}_1,\tilde{u}_{1,1},\ldots,\tilde{u}_{1,k-1},\tilde{u}_{1,k+1},\ldots,\tilde{u}_{1,N})}{P(u_k=-1|\mathbf{y}_1,\tilde{u}_{1,1},\ldots,\tilde{u}_{1,k-1},\tilde{u}_{1,k+1},\ldots,\tilde{u}_{1,N})}. \quad (1)$$

To compute $\log \frac{P(u_k=+1|\tilde{u}_{2,k})}{P(u_k=-1|\tilde{u}_{2,k})}$ we note that (See Fig. 4a)

$$P(u_k|\mathbf{y}_1, \tilde{\mathbf{u}}_1) = \frac{P(u_k|\mathbf{y}_1, \tilde{u}_{1,1}, \ldots, \tilde{u}_{1,k-1}, \tilde{u}_{1,k+1}, \ldots, \tilde{u}_{1,N}) P(\tilde{u}_{1,k}|u_k, \mathbf{y}_1, \tilde{u}_{1,1}, \ldots, \tilde{u}_{1,k-1}, \tilde{u}_{1,k+1}, \ldots, \tilde{u}_{1,N})}{P(\tilde{u}_{1,k}|\mathbf{y}_1, \tilde{u}_{1,1}, \ldots, \tilde{u}_{1,k-1}, \tilde{u}_{1,k+1}, \ldots, \tilde{u}_{1,N})} \tag{2}$$

Since $\tilde{u}_{1,k}$ was generated by decoder 2 and de-interleaving is used, this quantity depends only weakly on $\mathbf{y}_1$ and $\tilde{u}_{1,j}, j \neq k$. Thus we can have the following approximation

$$P(\tilde{u}_{1,k}|u_k, \mathbf{y}_1, \tilde{u}_{1,1}, \ldots, \tilde{u}_{1,k-1}, \tilde{u}_{1,k+1}, \ldots, \tilde{u}_{1,N}) \approx P(\tilde{u}_{1,k}|u_k) = 2P(u_k|\tilde{u}_{1,k})P(\tilde{u}_{1,k}). \tag{3}$$

Using eq. 3 in eq. 2 we obtain

$$P(u_k|\mathbf{y}_1, \tilde{u}_{1,1}, \ldots, \tilde{u}_{1,k-1}, \tilde{u}_{1,k+1}, \ldots, \tilde{u}_{1,N}) = \frac{P(u_k|\mathbf{y}_1, \tilde{\mathbf{u}}_1) P(\tilde{u}_{1,k}|\mathbf{y}_1, \tilde{u}_{1,1}, \ldots, \tilde{u}_{1,k-1}, \tilde{u}_{1,k+1}, \ldots, \tilde{u}_{1,N})}{2P(u_k|\tilde{u}_{1,k})P(\tilde{u}_{1,k})} \tag{4}$$

It is preferable to work with likelihood ratios to avoid computing probabilities not involving u_k (see Fig.4b), and to define $\tilde{L}_i(k) = \log \frac{P(u_k=+1|\tilde{u}_{i,k})}{P(u_k=-1|\tilde{u}_{i,k})}$, $i = 1, 2$. From eqs. 1 and **4 we** obtain $\tilde{L}_2^{(m)}(k) = L_1^{(m)}(k) - L_1^{(m-1)}(k)$ at the output of decoder '1, before interleaving, for the mth iteration. Similarly we can obtain $L_1^{(m)}(k) = L_2(k) - \tilde{L}_2^{(m)}(k)$ at the output of decoder 2, after deinterleaving. Using the above definitions, the a priori probabilities can be computed as

$$P(u_k = +1|\tilde{u}_{i,k}) = \frac{e^{\tilde{L}_i(k)}}{1 + e^{\tilde{L}_i(k)}} = 1 - P(u_k = -1|\tilde{u}_{i,k}), \quad i = 1, 2. \tag{5}$$

Then the update equation for the mth iteration of the decoder in Fig. 3 becomes

$$\tilde{L}_1^{(m)}(k) = \tilde{L}_1^{(m-1)}(k) + \alpha_m \left[L_2^{(m)}(k) - L_1^{(m)}(k) \right], \quad \alpha_m = 1. \tag{6}$$

This looks like the update equation of a steepest descent method, where $[L_2^{(m)}(k) - L_1^{(m)}(k)]$ represent the rate of change of $L(k)$ for a given u_k, and α_m is the step size.

Figure 5 shows the probability density function of $\tilde{L}_1(k)$ at the output of the second decoder in Fig. 1 (a), after de-interleaving and given $u_k = +1$. As shown in Fig. 5, this density function shifts to the right as the number m of iterations increases. The area under each density function to the left of the origin represents the bit error rate, if decoding stops after m iterations.

At this point certain observations can be made. Note that $\tilde{L}_2(k')$ at the input of decoder 2 includes an additive component $2\rho y_{1ik}$, which contributes to the branch metric computations in decoder 2 at observation y_{2ik}. This improves by 3 dB the signal-to-noise ratio of the noisy information symbols at the input of decoder 2. Similar arguments hold for $\tilde{L}_1(k)$. An apparently more powerful decoding structure can be considered, as shown in Fig .6.

However, the performance of the decoding structures in Fig .6 and Fig .3 is equivalent for a large number of iterations (the actual difference is one half iteration). If the structure in Fig .6 is used, then the log-likelihood ratio $\tilde{L}_2(k)$ fed to decoder 2 should not depend on $\tilde{u}_{1k}$ and y'_{1ik}, and similarly $\tilde{L}_1(k)$ should not depend on $\tilde{u}_{2k}$ and y'_{2ik}. Using analogous derivations based on eqs. 1 through 4, we obtain

$$\tilde{L}_2(k) = L_1(k) - \tilde{L}_1(k) - 2\rho y'_{1ik}$$

$$\tilde{L}_1(k) = L_2(k) - \tilde{L}_2(k) - 2\rho y'_{2ik},$$

where $\mathbf{y}'_{1i}$ is the sum of y_{1i} with the deinterleaved version of $\mathbf{y}_{2i}$ and $\mathbf{y}'_{2i}$ is the sum of y_{2i} with the interleaved version of $\mathbf{y}_{1i}$. Thus, the net effect of the decoding structure in Fig. 6 is to explicitly pass to decoder 2 the

information contained in y_{1i} (and vice-versa), but to remove the identical term from the input log-likelihood ratio.

Performance. The performance obtained by turbo decoding the code in Fig. 1(a) with random permutations of lengths N = 4096 and N = 16384 is compared in Fig. 7 to the capacity of a binary-input Gaussian channel for rate r = 1/4, and to the performance of a (15,1/4) convolutional code originally developed at JPL for the Galileo mission. At BER=5×10^{-3}, the turbo code is better than the (1 5,1 /4) code by 0.25 dB for N = 4096, and by 0.4 dB for N = 16384.

So far we have considered only component codes with identical rates, as shown in Fig. 1 (a). Now we propose to extend the results to encoders with unequal rates, as shown in Fig. 8. This structure improves the performance of the overall, rate 1/4, code, as shown in Fig.7. The gains at BER=5×10^{-3} relative to the (15,1/4) code are 0.55 dB for N = 4096, and 0.7 dB for N = 16384. For both cases, the performance is within 1dB of the Shannon limit at BER=5×10^{-3} and the gap narrows to 0.7 dB for N = 16384 at low BER.

Conclusions. Wc have shown how turbo codes and decoders can be used to improve the coding gain for deep-space communications, while decreasing the decoding complexity with respect to the large constraint length convolutional codes currently in use. Further analysis is needed 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, and to establish whether there might be a flattening of the performance. curves at higher E_b/N_o, as it appears in one of the curves in Fig. 7. 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.

Similar code constructions were used to build rnriltiple-encoder turbo codes. This generalizes the turbo decoding concept to a truly distributed decoding system where each sub-decoder works on a piece of the total observation and tentat ive estimates are shared among decoders until an acceptable degree of consensus is reached.

References

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (Abstract)," in 1972 Int. Symp. Information Theory, p. 90, May 1972.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in Proc. ICC '93, May 1993.

[3] 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.

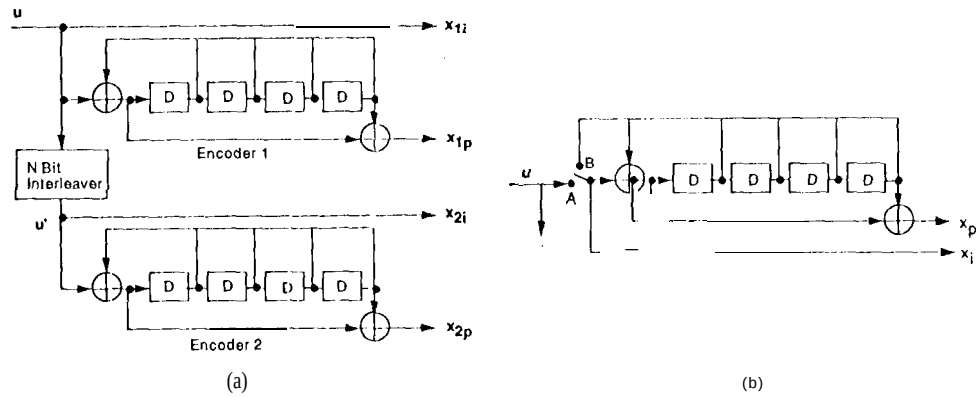


Figure 1: (a) Example of encoder. (b) Trellis Termination

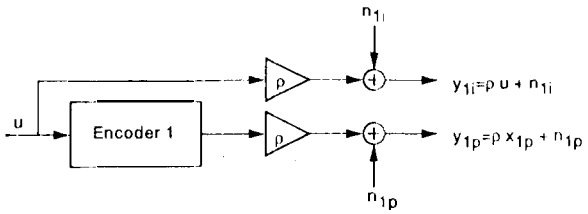


Figure 2: Channel model.

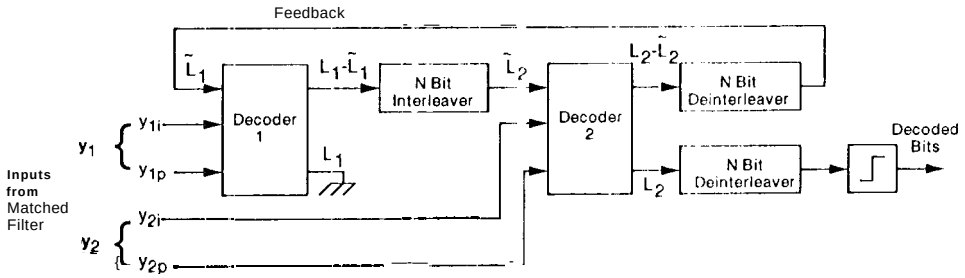
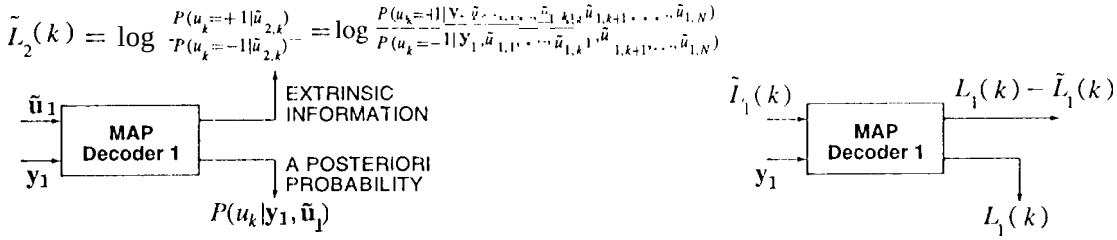


Figure 3: Turbo Decoder



(a) (b)

Figure 4: input/outputofMAPdecodcr

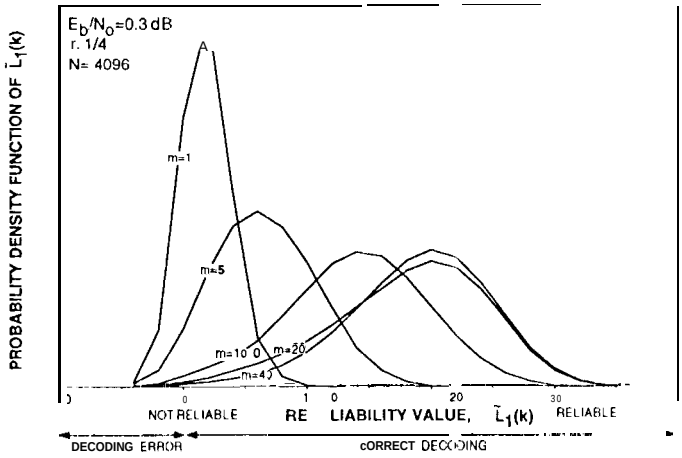


Figure 5: Reliability function

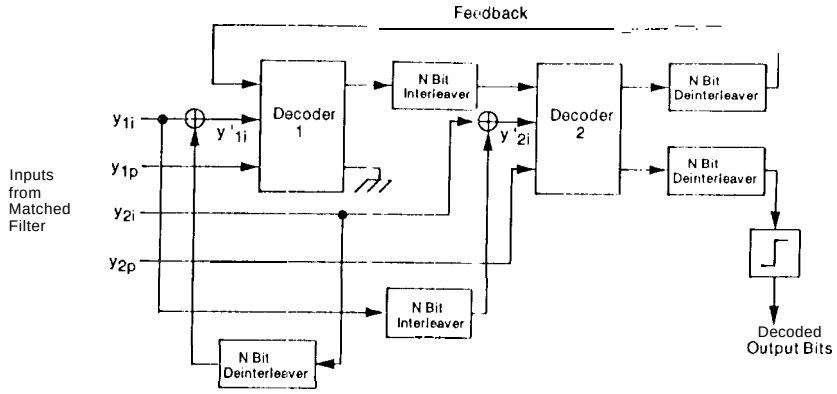


Figure 6: Equivalent turbo decoder

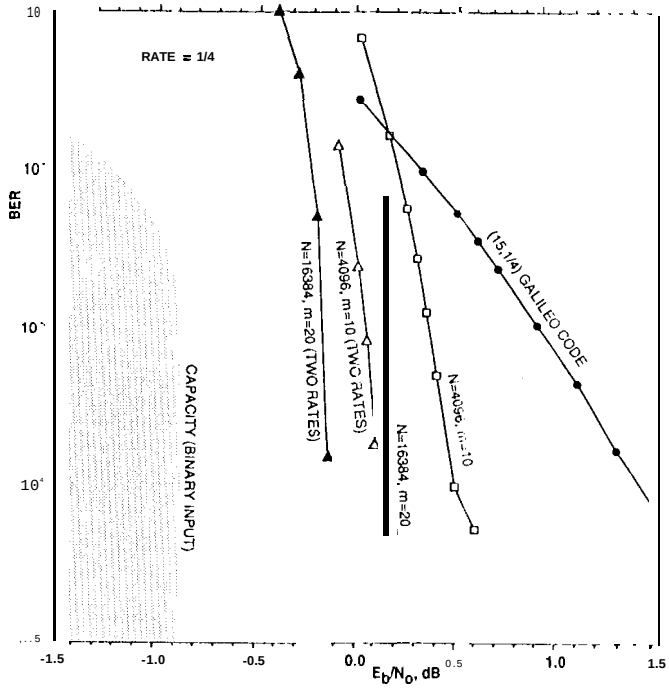


Figure 7: Turbo codes performance, $r = 1/4$

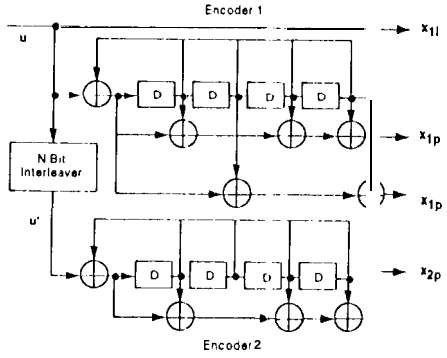


Figure 8: Two-rate encoder