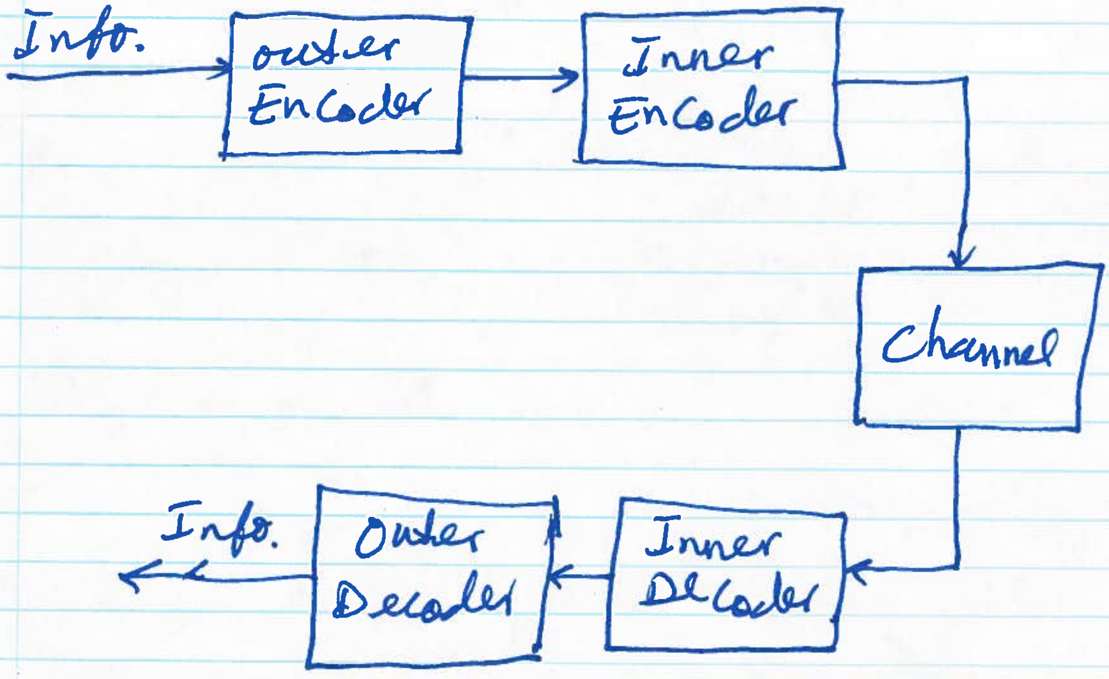
Concatenated Coding

As the error Correcting Capability of Codes increases, i.e., as their rate is reduced their decoding becomes more complex. For excample, if one uses a (255, 239) RS Code, he has to Solve a polynomial of degree 8 to find the 8 error bytes' location. But if he uses a (255, 205) Code, the number of errors that can be Corrected is increased to 25. So, a polynomial of degree 25 has to be found and solved.

Apart from rate, the length of Code has usually an exponential effect on the Complexity of decoding. And it is evident that in order to get good Coding gain, we need to have long Codewords.

In order to avoid the Complexity of decoding, Forney suggested the use of concatenated Codes. In this Scheme the data is coded with one encoder (called the outer encoder) and then with another (inner encoder). The decoding is performed in the reverse order.



It is possible to use more than two stages of coding.

The inner and outer codes can be either both block codes or one block code and another convolutional code, or some other arrangement.

One combination frequently used by industry is to use convolutional codes as the inner code and

Reed-Solomon Codes as the outer Code.

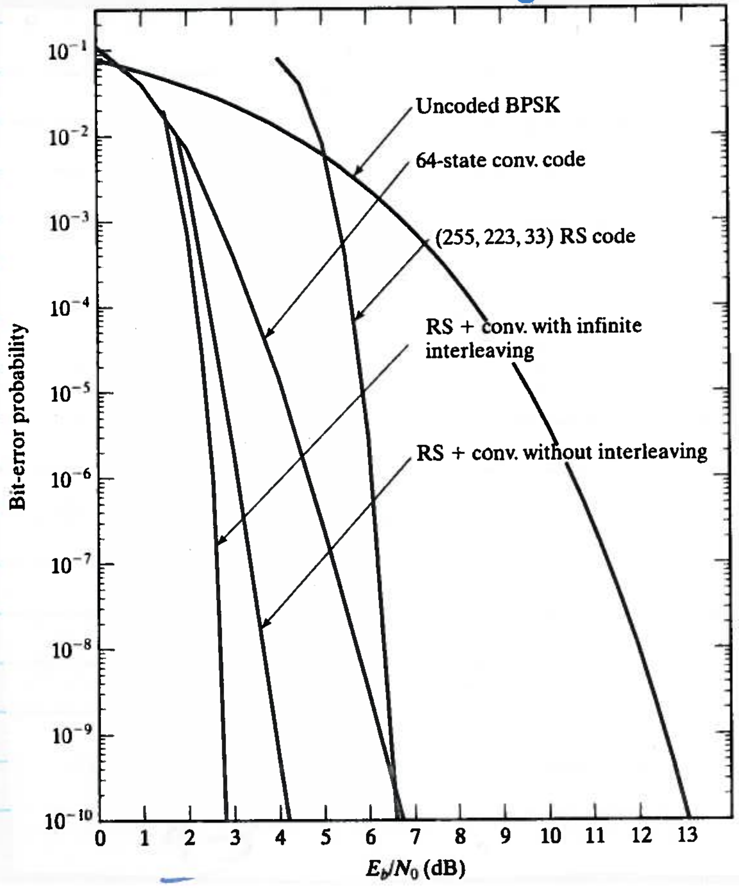
One example is NASA's Tracking Data Relay Satellite System (TDRSS) that uses a (255,223) RS Code as the outer Code and a 64 state Convolutional Code with

$g_1(D) = 1 + D + D^3 + D^4 + D^6$ and $g_2(D) = 1 + D^3 + D^4 + D^5 + D^6$.

This the industry standard (133, 171) Code.

The overall rate is $\frac{223}{255} \times \frac{1}{2} = 0.437$.

Performance of this scheme (with and without interleaving) is compared with uncoded case as well as just Convolutional or just RS Coding.

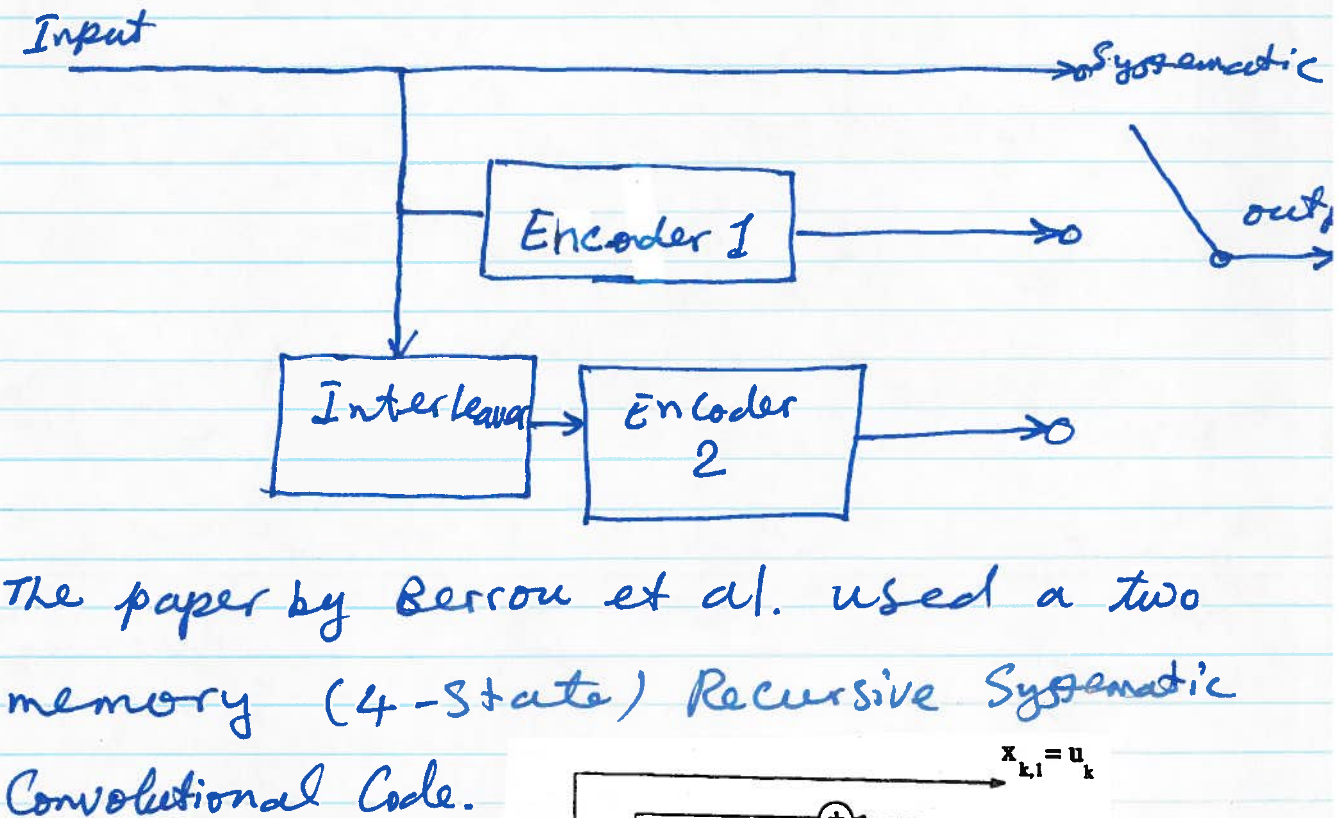


Turbo Codes

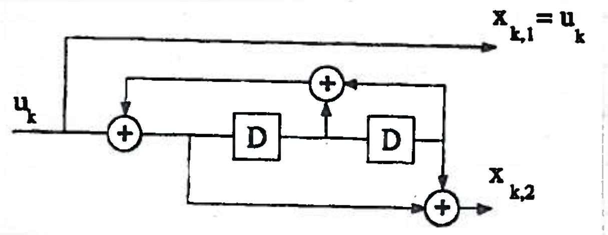
One possibility for improving the performance of concatenated codes is through iterative cooperation of inner and outer code to improve the decoding accuracy. In such as situation, insted of decoding the inner code and giving the hard decision to the outer code, one can extract soft values from the inner code, i.e., likelihood ratios and pass it to outer decoder. Then outer decoder instead of starting from no prior information, that is, assuming $P(0) = P(1) = \frac{1}{2}$ can start from somewhere closer to actual situation. The order decoder can then use this informatio to extract new likelihood ratios and passing them to the inner code. This process can be iterated until a steddy-state is reached. While this is possible with a serial concatenatio and any combination of codes, the best result can be achieved by parallel concatenation of two convolutional codes.

In 1993 Berrou, Glavieux and Thitimajshima showed that using two very simple convolutional codes working on two copies of the input data (one being a permutation of the other), one can get results very close to Shannon's bounds.

The encoder has the following structure



The paper by Berrou et al. used a two memory (4-state) Recursive Systematic Convolutional Code.



Realization of the systematic convolutional encoder with feedback for the rate-1/2 code with memory 2. The generator polynomials are $g_0(D) = 1 + D + D^2$ and $g_1(D) = 1 + D^2$.

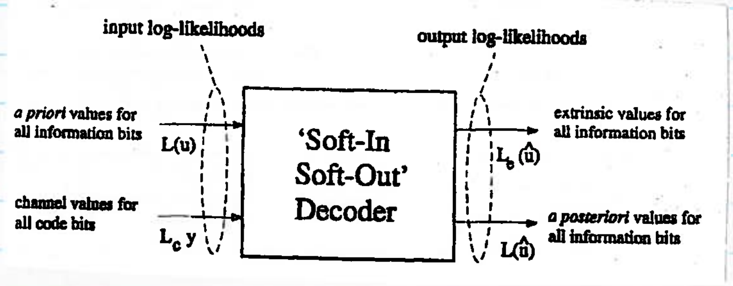
The above code sends the systematic part, only from one encoder, plus two parity streams. So, the rate of the code is $\frac{1}{3}$. However, by puncturing the parity streams, one can get higher rates. For example, if we send one systematic bit and a parity from one of the two on each output, i.e.,

$$u, p_1, u, p_2, u, p_1, u, p_2, \ldots$$

we have a rate $\frac{1}{2}$ code.

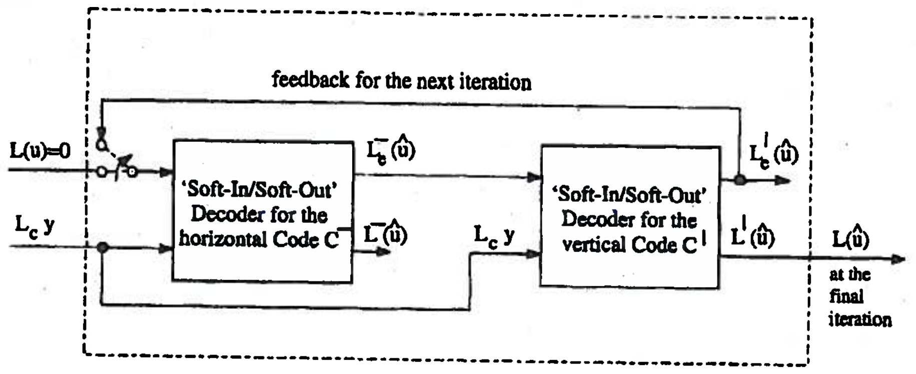
While the constituent encoders each have only four states, the interleaver causes the output stream have large minimum free distance.

To <u>decode</u> the turbo code, we need two SISO (Soft-input-Soft-output) decoders


input log-likelihoods
output log-likelihoods
a priori values for all information bits
$L(u)$
channel values for all code bits
$L_c y$
'Soft-In Soft-Out' Decoder
extrinsic values for all information bits
$L_e(\hat{u})$
a posteriori values for all information bits
$L(\hat{u})$

Each SISO decoder works on one of the two streams. It takes as input the channel value, y, and the probability of the corresponding systematic bit being one (or equivalently, likelihood ratio) and generates its own likelihood ratio.

It then passes its result, i.e., likelihood ratio (or log-likelihood ratio) or the improvement it has added $L_e(\hat{u})$ called extrinsic log-likelihood ratio to the other decoder.


feedback for the next iteration
L(u)=0
L_c y
'Soft-In/Soft-Out' Decoder for the horizontal Code C⁻
L_e⁻(û)
L⁻(û)
L_c y
'Soft-In/Soft-Out' Decoder for the vertical Code C|
L_e|(û)
L|(û)
L(û) at the final iteration

Iterative decoding procedure with two "soft-in/soft-out" decoders with initial $L(u) = 0$, i.e., equally likely source (information) bits.

At the beginning $L(u)=0$, i.e., we assume that $P(0)=P(1)=\frac{1}{2}$ so $\log \frac{P(u=0)}{P(u=1)} = \log \frac{1/2}{1/2} = 0$.

But in other iterations, $L^{-}(\hat{u})$ and $L^{|}(\hat{u})$ are evolved, iteratively.

Decoding of Turbo Codes

The Soft-value of a bit is given as its log-likelihood ratio:

$$L(u) = \log \frac{P(u=0)}{P(u=1)}$$

Assume that we encode a sequence of k bits using an (n, k) Systematic Code. k of the encoded bits of the Code sequence say x are equal to the values of u. The other n−k are parity bits.

Now assume that we assign ±1 to bits of x, i.e., $x=0 \rightarrow +1$ and $x=1 \rightarrow -1$.

Assume that we transmit x's over an additive White Gaussian Noise (AWGN) Channel with possibly multiplying the Symbols by a fade factor a.

The Conditional soft-value or log-likelihood ratio of x given y (the matched filter output) is:

$$L(x|y) = \log \frac{P(x=+1|y)}{P(x=-1|y)} = \log\left[\frac{P(x=+1)\, P(y|x=+1)}{P(x=-1)\, P(y|x=-1)}\right]$$

$$p(y|x=+1) = \frac{1}{\sqrt{\pi \frac{N_0}{E_s}}} e^{-\frac{E_s}{N_0}(y-a)^2}$$

and

$$p(y|x=-1) = \frac{1}{\sqrt{\pi \frac{N_0}{E_s}}} e^{-\frac{E_s}{N_0}(y+a)}$$

So:

$$L(x|y) = \log \frac{e^{-\frac{E_s}{N_0}(y-a)^2}}{e^{-\frac{E_s}{N_0}(y+a)^2}} + \log \frac{P(x=+1)}{P(x=-1)}$$

or

$$L(x|y) = 4a\frac{E_s}{N_0} \cdot y + L(x)$$

$$= L_c \cdot y + L(x)$$

for an AWGN Channel $a=1$.

A <u>maximum</u> <u>a posteriori</u> (MAP) decoder outputs an <u>a</u> <u>posteriori</u> (i.e., after observing the total output of the matched filter $\underline{y} = (y_1, y_2, \ldots)$) log-likelihood ratio for transmitted $+1$ and -1:

$$L(\hat{u}) = L(u|\underline{y}) = \log \frac{P(u=+1|\underline{y})}{P(u=-1|\underline{y})}$$

A MAP decoder uses a priori values $L(u)$ for all information bits and the channel values $L_c \cdot y$. It generates for each information bit a soft value $L(\hat{u})$ that in addition to $L_c \cdot y$ an $L(u)$ has $L_e(\hat{u})$ called the extrinsic information that depends on the structure of the encoder.

$$L(\hat{u}) = L_c \cdot y + L(u) + L_e(\hat{u})$$

The first decoder starts with $L_c \cdot y$ only (at the beginning) and generate the extrinsic information

$$\bar{L_e}(\hat{u}) = \bar{L}(\hat{u}) - L_c \cdot y$$

The second decoder uses $\bar{L_e}(\hat{u})$ and generates

$$L_e'(\hat{u}) = L'(\hat{u}) - (L_c \cdot y + \bar{L_e}(\hat{u})).$$

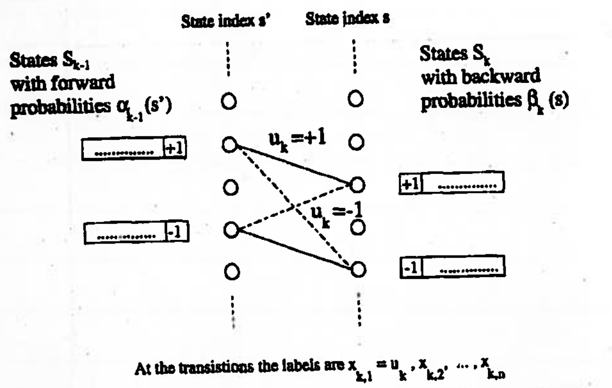
This continues iteratively. At the end

$$L(\hat{u}) = L_c \cdot y + \bar{L_e}(\hat{u}) + L_e'(\hat{u})$$

is output and is used for making the final (hard) decision).

At this point we discuss the generation of $L(\hat{u}_k)$ for all k for convolutional codes.

Consider the trellis diagram of a Systematic Convolutional Code given as:


State index s'
State index s
States S_{k-1} with forward probabilities $\alpha_{k-1}(s')$
States S_k with backward probabilities $\beta_k(s)$
$u_k=+1$
$u_k=-1$
+1
-1
At the transitions the labels are $x_{k,1}=u_k, x_{k,2}, \dots, x_{k,n}$

Trellis structure of systematic convolutional codes with feedback encoders.

Then

$$L(\hat{u}_k) = \log \frac{P(u_k=+1|\underline{y})}{P(u_k=-1|\underline{y})} = \log \frac{\sum_{s,s',u_k=+1} p(s',s,\underline{y})}{\sum_{\substack{s,s' \\ u_k=-1}} p(s',s,\underline{y})}$$

where states s' and s define bit u_k and the coded bit $x_{k,\nu}$, for $\nu=2,\dots,n$.

The sums of probabilities $p(s',s,\underline{y})$ are taken over all existing transitions from s' to s with transition label $u_k=+1$ (in numerator) and with label $u_k=-1$ (in the denominator).

$$p(s', s, \underline{y}) = p(s', \underline{y}_{j<k}) \cdot p(s, \underline{y}_k | s') \cdot p(\underline{y}_{j>k} | s)$$

$$= p(s', \underline{y}_{j<k}) \cdot P(s|s') \cdot p(\underline{y}_k | s', s) \, p(\underline{y}_{j>k} | s)$$

$$= \alpha_{k-1}(s') \cdot \gamma_k(s', s) \cdot \beta_k(s)$$

$\underline{y}_{j<k}$ is the sequence of received symbols from the beginning up to time $k-1$, $\underline{y}_{j>k}$ is the sequence from $k+1$ to the end.

α_k's can be found using the forward recursion

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

and β_k's are found using backward recursion.

$$\beta_{k-1}(s') = \sum_{s} \gamma_k(s', s) \cdot \beta_k(s)$$

The branch transition probabilities ar given by,

$$\gamma_k(s', s) = p(\underline{y}_k | u_k) \cdot P(u_k)$$

We start recursion with

$$\alpha_{start}(0) = 1 \quad \text{and} \quad \beta_{end}(0) = 1.$$

$$L(u_k) = \log \frac{P(u_k=+1)}{P(u_k=-1)} \Rightarrow \frac{P(u_k=+1)}{1-P(u_k=+1)} = e^{L(u_k)}$$

So,

$$P(u_k=+1)=\frac{e^{L(u_k)}}{1+e^{L(u_k)}}$$

Similarly

$$P(u_k=-1)=\frac{e^{-L(u_k)}}{1+e^{-L(u_k)}}$$

Putting these two Together:

$$P(u_k=\pm 1)=\frac{e^{\pm L(u_k)}}{1+e^{\pm L(u_k)}}=\frac{e^{-\frac{1}{2}L(u_k)}}{1+e^{-L(u_k)}}e^{\frac{L(u_k)u_k}{2}}$$

$$=A_k\cdot e^{L(u_k)u_k/2}$$

Similarly,

$$p(\underline{y}_k|u_k)=B_k\cdot\exp\left[\frac{1}{2}L_c y_{k,1}u_k+\frac{1}{2}\sum_{\nu=2}^{n}L_c y_{k,\nu}x_{k,\nu}\right]$$

So,

$$\gamma_k(s',s)=\exp\left[\frac{1}{2}u_k(L_c y_{k,1}+L(u_k))\right]\gamma_k^{(e)}(s',s)$$

[after ignoring factors not depending on u_k and $y_{k,\nu}$.]

where

$$\gamma_k^{(e)}(s',s)=\exp\left[\frac{1}{2}\sum_{\nu=2}^{n}L_c y_{k,\nu}x_{k,\nu}\right]$$

Since $\exp\left[\frac{1}{2}u_k(L_c y_{k,1} + L(u_k))\right]$ is common in all terms of the sum $\sum p(s', s, \underline{y})$, we have

$$L(\hat{u}_k) = L_c y_{k,1} + L(u_k) + \log \frac{\sum\limits_{\substack{(s',s)\\u_k=+1}} \gamma_k^{(e)}(s',s)\,\alpha_{k-1}(s')\,\beta_k(s)}{\sum\limits_{\substack{(s',s)\\u_k=-1}} \gamma_k^{(e)}(s',s)\,\alpha_{k-1}(s')\,\beta_k(s)}$$

<u>log-max algorithm</u>:

Since the summations in the above formula are difficult to perform, the following fact is used to reduce the complexity:

$$\log(x+y) = \log x\left(1+\frac{y}{x}\right) \simeq \log x \quad \text{if } \frac{y}{x} \ll 1$$

or $y \ll x$. That is

$$\log(x+y) = \log \max(x, y).$$

So, the approximate version of MAP or Max-log MAP is:

$$L(\hat{u}) = L_c \cdot y_k + L(u_k) + \max_{\substack{s',s\\u_k=+1}}\left[\log \alpha_{k-1}(s') + \log \beta_k(s) + \log \gamma_k^{(e)}(s',s)\right]$$
$$- \max_{\substack{s',s\\u_k=-1}}\left[\log \alpha_{k-1}(s') + \log \beta_k(s) + \log \gamma_k^{(e)}(s',s)\right].$$

Performance results:

Following shows the performance of rate $\frac{1}{2}$ Turbo Code with 1, 2, ..., 18 iterations. Each iteration involves two decodings, one by each decoder and exchange of informatio

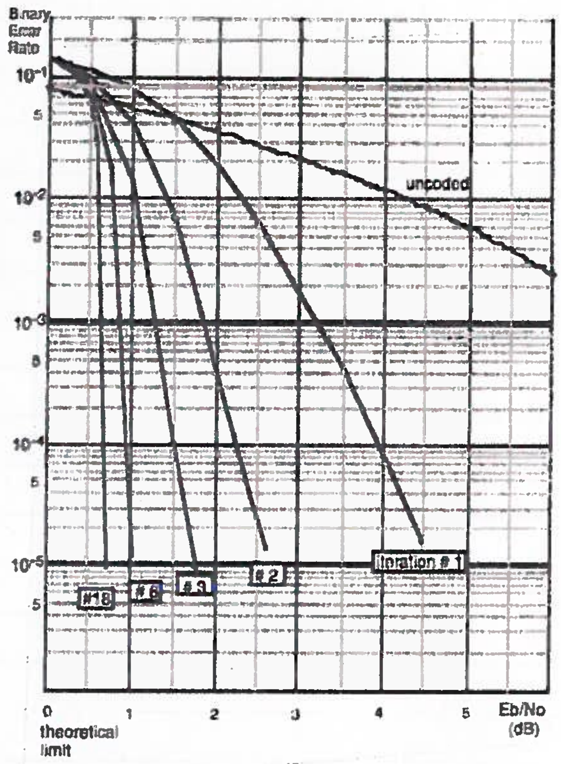


Fig. 9. BER given by iterative decoding ($p = 1, \cdots, 18$) of a rate $R = 1/2$ encoder, memory $\nu = 4$, generators $G_1 = 37, G_2 = 21$, with interleaving 256×256.