Military Technical College
Kobry El-Kobba
Cairo, Egypt

12-th International Conference
on
Aerospace Sciences &
Aviation Technology

Illuminating the Mechanism of Iterative Turbo Code Decoding Process

Moataz M. Salah*, Salah S. El-Agooz*, Ashraf I. Mahroos*

ABSTRACT

The most exciting and potentially important development in coding theory in recent years has been the dramatic announcement of "Turbo codes". Turbo codes are constructed by applying two or more component codes to different interleaved versions of the same information sequence. Then, the encoded bits are decoded through an iterative decoding algorithm of relatively low complexity. Turbo code decoder consists of two soft-input/soft-output component convolutional decoders that work together in an iterative fashion. Turbo code decoding process is the most complex part of the turbo coding decoding process. Turbo decoding process has a lot of ambiguity and scarce of decoding details in the literature publication. Due to of these, the goal of this paper is shading the light to the mechanism of the turbo decoder to illuminate the principles of the iterative decoding process of the turbo code decoder through a numerical example applied to the actual turbo decoder.

KEYWORDS: Turbo Codes, Concatenated Codes, Iterative decoding.

* Egyptian Armed Forces

I. Introduction

Parallel concatenated codes, turbo codes [1], have been shown 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 a Bit Error Rate (BER) of 10^{-5} and code rate of 1/2.

The encoder block diagram of the turbo encoder is shown in Figure 1. Two identical recursive systematic convolutional (RSC) component codes of rate 1/2 which are separated by an interleaver form the turbo encoder. Thus, the same information sequence is encoded twice but in different orders. The output sequence from the upper encoder and from the lower encoder are multiplexed and optionally punctured.

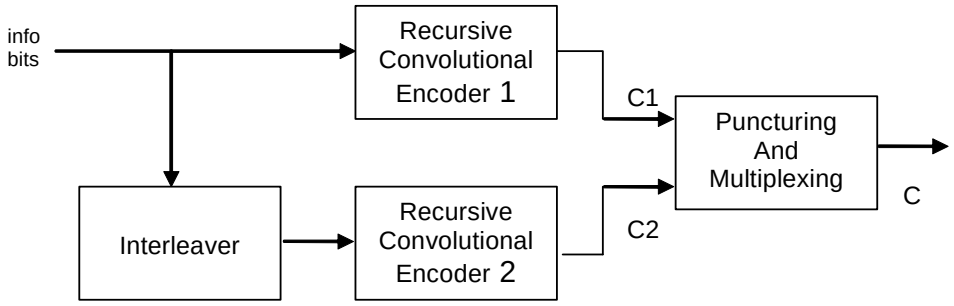
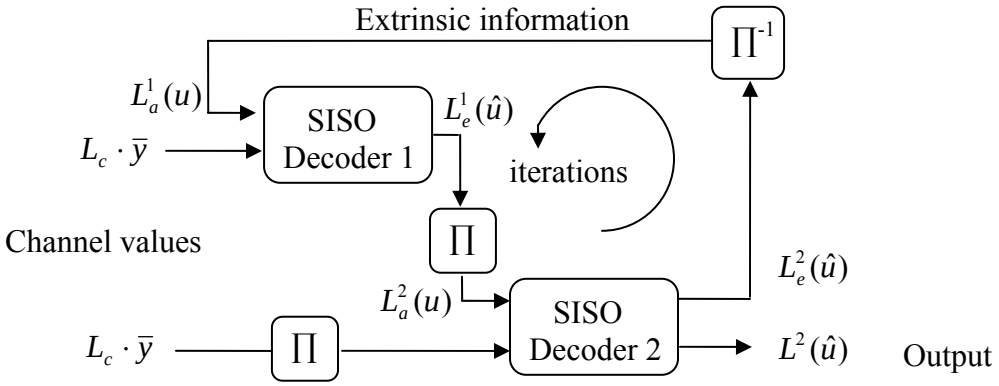


Figure 1. Turbo Code Encoder.

Note that turbo decoder does not perform maximum likelihood decoding directly, but attempts to achieve maximum likelihood decoding in an iterative way. The original turbo decoder [1] used two maximum aposteriori (MAP) algorithm decoders. There are other less complex algorithms that can be used in place of the MAP algorithm for each decoder such as SOVA (soft output Viterbi algorithm) [2] and Max-log MAP [3].
An iterative turbo decoder consists of two component decoders, identical to the one in the encoder, concatenated serially via an interleaver as shown in Figure 2, these two component decoders are soft input soft output (SISO) decoders.
SISO decoder accepts soft inputs and give soft outputs for the decoded sequence. These soft inputs and outputs provide not only an indication of whether a particular bit was a 0 or a 1, but also a *likelihood ratio* which gives the *probability* that the bit has been correctly decoded. The turbo decoder operates *iteratively*. In the first iteration the first SISO decoder provides a soft output giving an estimation of the original data sequence based on the soft channel inputs alone. It also provides an *extrinsic* output. The *extrinsic* output for a given bit is based not on the channel input for that bit, but on the information for surrounding bits and the constraints imposed by the code being used. This extrinsic output from the first decoder is used by the second RSC decoder as *a-priori information*, and this information together with the channel inputs are used by the second SISO decoder to give its soft output and extrinsic information. In the second iteration the extrinsic information from the second decoder in the first iteration is used as the a-priori information for the first decoder, and using this a-priori information the decoder can hopefully decode *more* bits correctly than it did in the first iteration. This

cycle continues, with at each iteration both SISO decoders producing a soft output and extrinsic information based on the channel inputs and a-priori information obtained from the extrinsic information provided by the previous decoder. After each iteration the Bit Error Rate (BER) in the decoded sequence drops, improvements obtained with the number of iterations increases.
Decoding can be stopped, and a final decoding estimate declared, after some fixed number of iterations (usually on the order of 2-12) [1], or based on a stopping criterion which is designed to detect when the estimate is reliable with very high probability.



$\prod$ interleaver

$\prod^{-1}$ De-interleaver

Figure 2. Iterative decoding procedure with two SISO decoders.

For illustration the performance enhancement of turbo codes due to its repetitive iterations, Figure 3 shows the performance of a turbo decoder using the MAP algorithm versus the number of decoding iterations which were used. As the number of iterations used by the turbo decoder increases, the turbo decoder performs significantly better. However after 8 iterations there is little improvement achieved by using further iterations. It can be seen from Figure 3 that using 16 iterations rather than 8 gives an improvement of only about 0.1 dB. Hence for complexity reasons usually only between about 4 and 12 iterations are used.

The paper is organized as following. Section II presents the principle of the iterative decoding process of two-dimensional systematic convolutional codes using any soft-input/soft-output decoder. Section III addresses the problem of estimating the state sequence of a Markov process observed through noise using the trellis based decoding algorithms, which is known as MAP algorithm. Section IV presents a numerical example to illustrate the mechanism of the turbo code decoding process. Section V summarizes the paper.

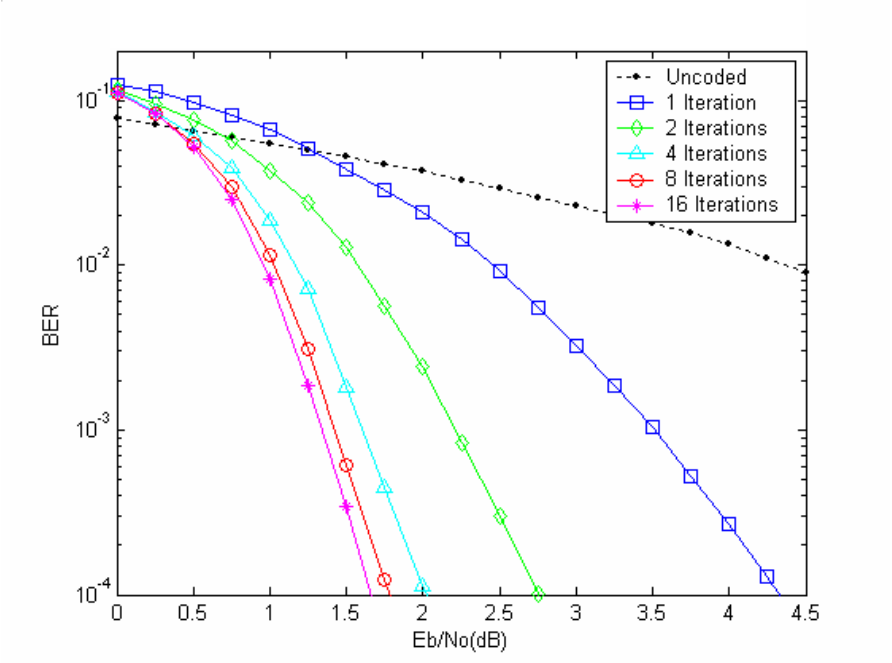


Figure 3. The effect of the number of iterations.

II. ITERATIVE DECODING PRINCIPLES

SISO (Soft-in Soft-output) algorithms are well suited for iterative decoding because they accept *a priori* information at their input and produce *a posteriori* information at their output. The "Soft-in Soft-output" decoder [4] shown in Figure 4 is used for decoding of the component codes.

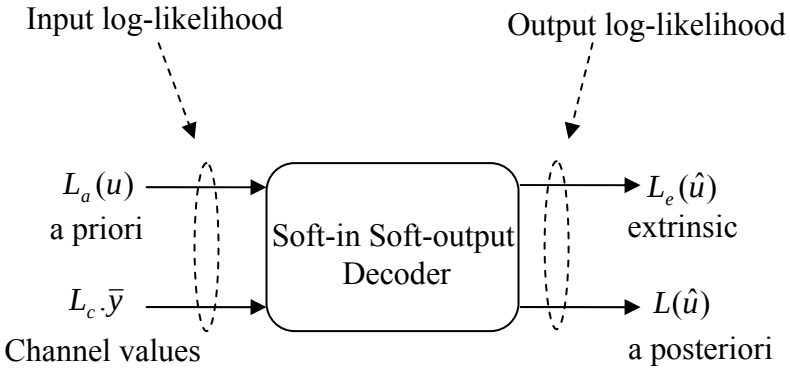


Figure 4. "Soft-in Soft-output" decoder.

The input of the decoder is the *a priori* values $L_a(u)$ for all information bits *u*, (initially set to zero), and the *channel values* $L_c.\bar{y}$ for all coded bits,

$$L_c = 4a\left(\frac{E_s}{2\sigma^2}\right) \tag{1}$$

where L_c is the channel reliability, $\boldsymbol{E_s = r\,E_b}$ is the energy per code symbol, ***r*** is the *code rate*, $\boldsymbol{E_b}$ is energy per information bit, σ^2 noise variance, and ***a*** is the fading amplitude, and is constant (equal one) for AWGN channel.

The output of the decoder $L(\hat{u})$ is defined as the *a posteriori* log-likelihood ratio, that is, the logarithm of the ratio of the probabilities of a given bit being +1 or -1, for binary phase shift key (BPSK), given the observation $\bar{y}$ (received channel values).

$$L(\hat{u}) = L(u \mid \bar{y}) = \ln\left(\frac{P(u=+1 \mid \bar{y})}{P(u=-1 \mid \bar{y})}\right) \tag{2}$$

The decoder also delivers extrinsic information $L_e(\hat{u})$, which will be used by the second decoder as *a priori* information. For systematic codes, the soft output for the information bit u will be represented as the sum of three terms [4]:

$$L_{ouput} = L_{channel} + L_{apriori} + L_{extrinsic} \tag{3}$$

$$L(\hat{u}) = L_c.y_s + L_a(u) + L_e(\hat{u}) \tag{4}$$

This means there are three *independent* estimates for the log-likelihood ratio of the information bits:

The channel values $L_c.y_s$, where y_s received systematic bits, the a priori $L_a(u)$, and the extrinsic $L_e(\hat{u})$ (output of the decoder).

The whole procedure of iterative decoding with two "Soft-in Soft-output" decoders is shown in Figure 2. In the first iteration of the iterative decoding algorithm, decoder 1 computes the extrinsic information as follows:

$$L_e^1(\hat{u}) = L^1(\hat{u}) - [L_c.y_s + L_a^1(u)] \tag{5}$$

Assuming equally likely information bits, thus by initializing $L_a^1(u) = 0$ (corresponding to *P(u) = 0.5*) for the first iteration. The extrinsic information from the first decoder is passed to the decoder 2, which uses $L_e^1(\hat{u})$ as the *a priori* information. Hence the extrinsic information value computed by Decoder 2 is

$$L_e^2(\hat{u}) = L^2(\hat{u}) - [L_c.y_s + L_e^1(\hat{u})] \tag{6}$$

Then, decoder 1 will use the extrinsic information values $L_e^2(\hat{u})$ as *a priori* information in the second iteration. The computation is repeated in each iteration, the iteration process is usually terminated after a predetermined number of iterations or when the soft-output value $L^2(\hat{u})$ *stabilizes* and changes little between successive iterations. In the final iteration decoder 2 makes the hard decision based on the sign of $L^2(\hat{u})$

$$\tilde{u} = sign\left[L^2(\hat{u})\right] \tag{7}$$

The *sign* of the LLR $L^2(\hat{u})$ of a bit ***u*** will indicate whether the bit *is more likely* to be +1 or -1, and the *magnitude* of the LLR $L^2(\hat{u})$ gives an indication of *how likely* the LLR $L^2(\hat{u})$ gives the *correct* value of the information bit.

III. MAXIMUM APOSTERIORI (MAP) ALGORITHM

The MAP algorithm examines *every* possible path through the convolutional decoder trellis and provides not only the estimated bit sequence, but also the *probabilities* for each bit that it has been decoded correctly.
Given the received sequence $\bar{y}$ can be split up into three sections as shown in Figure 5:
The received codeword associated with the present transition $\bar{y}_k$
The received sequence prior to the present transition $\bar{y}_{j<k}$
The received sequence after the present transition $\bar{y}_{j>k}$

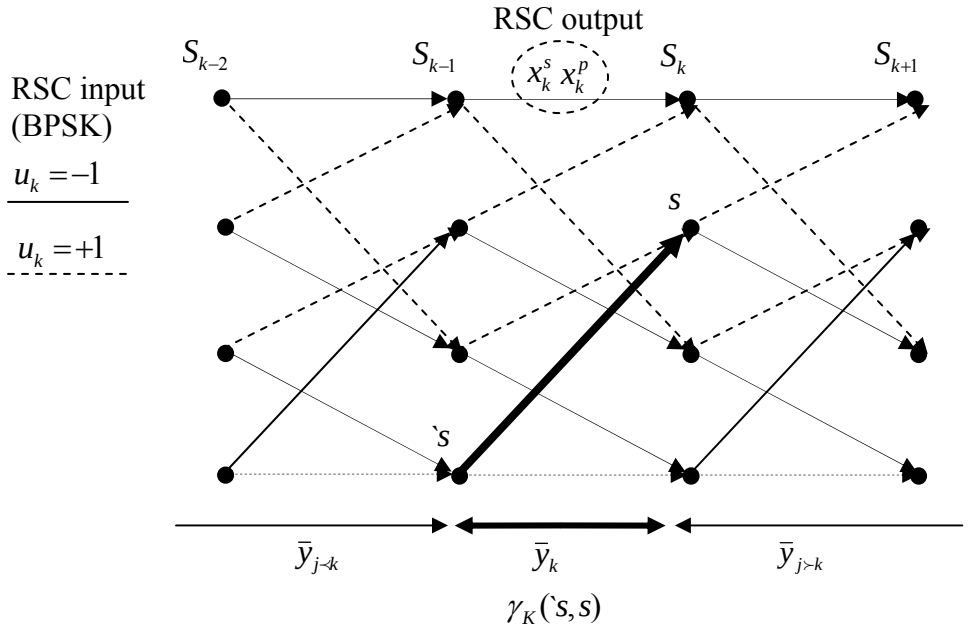


Figure 5. MAP decoder trellis for k = 3 RSC code.

The MAP algorithm gives, for each decoded bit $\boldsymbol{u_k}$, the probability that this bit was +1 or -1 (for BPSK modulation), given the received symbol sequence $\bar{y}$. This is equivalent to finding the *a-posteriori log likelihood ratio LLR* $L(u_k \mid \bar{y})$, where

$$L(u_k \mid \bar{y}) = \ln\left(\frac{P(u_k = +1 \mid \bar{y})}{P(u_k = -1 \mid \bar{y})}\right) \tag{8}$$

Assume the previous state $S_{k-1} = `s$ and the present state $S_k = s$ are known in a trellis then the input bit $u_k$ which caused the transition between these states will be known. This, along with Bayes' rule and the fact that the transitions between the previous state $`s$ to the present state s in the trellis are mutually exclusive (i.e., only one of them could have occurred at the encoder), allow us to rewrite equation (8) as:

$$L(u_k \mid \bar{y}) = \ln\left(\frac{\sum_{u_k=+1}^{(`s,s)} P(`s, s, \bar{y})}{\sum_{u_k=-1}^{(`s,s)} P(`s, s, \bar{y})}\right) \tag{9}$$

Where the summation ($\sum_{u_k=+1}^{(`s,s)}$) is the set of transitions from previous state $`s$ to the present state s in the trellis that can occur if the input bit $u_k = +1$ and similarly for the summation ($\sum_{u_k=-1}^{(`s,s)}$).

Where $P(`s, s, \bar{y})$ can be written as following form [5]:

$$P(`s, s, \bar{y}) = \alpha_{k-1}(`s).\gamma_K(`s, s).\beta_K(s) \tag{10}$$

Where,

- $\alpha_{k-1}(`s)$ is the probability that the trellis is in state $`s$ at time $k-1$ and the received channel sequence up to this point is $\bar{y}_{j \prec k}$.
- $\beta_K(s)$ is the probability that given the trellis is in state s at time k the future received channel sequence will be $\bar{y}_{j \succ k}$.
- $\gamma_K(`s, s)$ is the *branch* transition probability that given the trellis was in state $`s$ at time $k-1$, it moves to state s and the received channel sequence for this transition is $\bar{y}_k$, $\gamma_K(`s, s)$ referred to the *branch metric* in the decoding algorithm.

From equation (10) substitute in equation (11) the conditional LLR of u_k, given the received sequence $\bar{y}$ is written as follows:

$$L(u_k \mid \bar{y}) = \ln\left(\frac{\sum_{u_k=+1}^{(`s,s)} \alpha_{k-1}(`s).\gamma_K(`s, s).\beta_K(s)}{\sum_{u_k=-1}^{(`s,s)} \alpha_{k-1}(`s).\gamma_K(`s, s).\beta_K(s)}\right) \tag{11}$$

The MAP algorithm finds $\alpha_k(s)$ and $\beta_K(s)$ for all states s throughout the trellis, i.e., for $k = 0,1,...N-1$, and $\gamma_K(`s, s)$ for all possible transitions from state $S_{k-1} = `s$ to state $S_k = s$, again for $k = 0,1,...N-1$. These values are then used with equation (11) to give the conditional LLRs $L(u_k \mid \bar{y})$ that the MAP decoder delivers as its output.

- **Branch metric Calculations, $\gamma_K(`s, s)$:**

After mathematical manipulation presented in [5], $\gamma_K(`s, s)$ can be written as follows:

$$\gamma_K(`s, s) = C \cdot e^{\frac{u_k}{2}\left(L_a(u_k) + L_c \cdot y_k^s\right)} \cdot e^{\frac{L_c}{2}\sum_{i=2}^{n} y_{ki}^p \cdot x_{ki}^p} \tag{12}$$

Where, C is a constant and $L_a(u_k)$ input from other decoder. Figure 6 shows the recursive calculation of $\alpha_k(s)$ and $\beta_K(s)$ from the knowledge of $\gamma_K(`s, s)$ for *s = 1*.

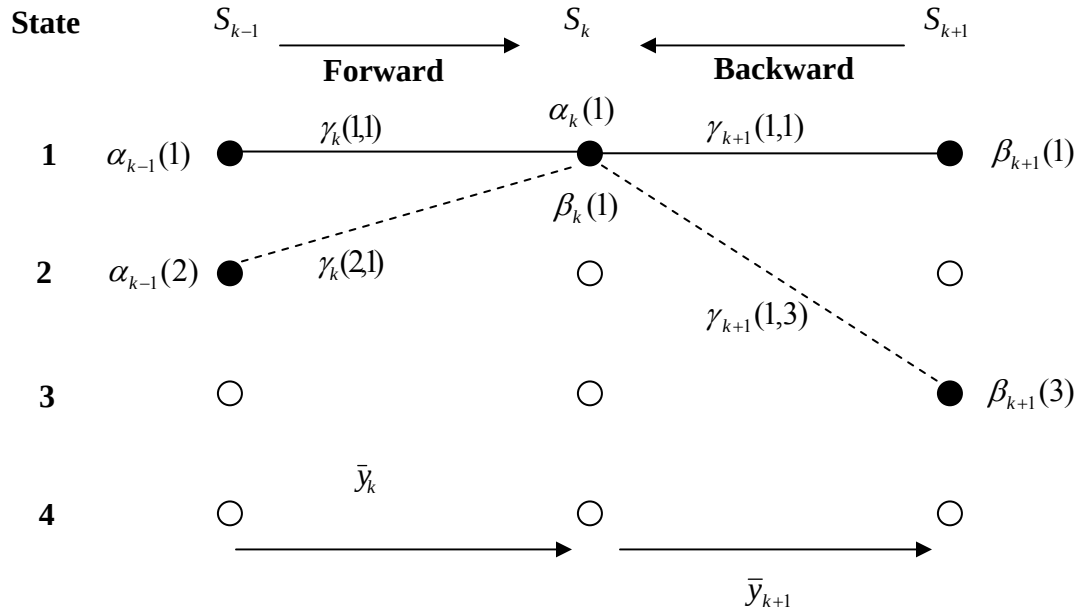
- **Forward recursive calculations, $\alpha_k(s)$:**

Once the branch metrics values $\gamma_K(`s,s)$ are known for every path through the trellis, the $\alpha_k(s)$ values can calculated recursively in *forward* direction.

$$\alpha_k(s) = \sum_{all \to `s} \alpha_{k-1}(`s) \cdot \gamma_k(`s,s) \tag{13}$$

Since the RSC encoder starts in all-zero state, the $\alpha_k(s)$ probabilities are initialized as follows:

$$\alpha_0(s) = \begin{cases} 1 & s = 1 \\ 0 & s \neq 1 \end{cases} \tag{14}$$



$$\alpha_k(1) = \alpha_{k-1}(1) \cdot \gamma_k(1,1) + \alpha_{k-1}(2) \cdot \gamma_k(2,1)$$
$$\beta_k(1) = \beta_{k+1}(1) \cdot \gamma_{k+1}(1,1) + \beta_{k+1}(3) \cdot \gamma_{k+1}(1,3)$$

Figure 6. Recursive calculation of $\alpha_k(1)$ and $\beta_k(1)$.

- **Backward recursive calculations, $\beta_K(s)$:**

The $\beta_K(s)$ values can be calculated recursively in *backward* direction.

$$\beta_{k-1}(`s) = \sum_{all \to s} \beta_k(s) \cdot \gamma_k(`s,s) \tag{15}$$

Since the RSC encoder terminates in all-zero state, the $\beta_K(s)$ probabilities are initialized as follows:

$$\beta_N(s) = \begin{cases} 1 & s = 1 \\ 0 & s \neq 1 \end{cases} \tag{16}$$

If the RSC encoder is left unterminated, then

$$\beta_N(s) = \frac{1}{2^{M_c}} \qquad \forall\ s \text{ (Equally likely Probabilities)} \tag{17}$$

Where, *N* frame length, 2^{M_c} number of trellis states, $M_c = K_c - 1$, K_c constrain length of the convolutional encoder.

- **Summary of the MAP algorithm**

The algorithm can be easily computed by using equations (12), (13) and (15), then substituted in equation (11) as shown in Figure 7.

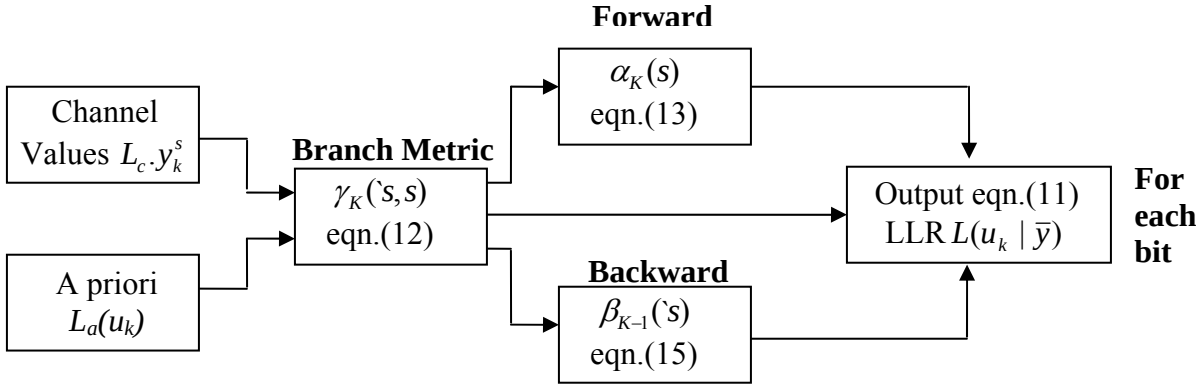


Figure 7. Summary of the MAP algorithm.

IV. NUMERICAL EXAMPLE OF TURBO ENCODING AND DECODING

In this section, an example of Turbo Decoding procedure is illustrated step by step. Figure 8 shows a RSC encoder (n,k,m) with generator matrix $G = [1, \frac{g_0}{g_1}]$, where g_0 forward polynomial, g_1 feedback polynomials, memory size $m = 1$, Constrain length $l = 2$,number of states = 2 (Only 2 states for simplicity), and rate $r = \frac{k}{n} = \frac{1}{2}$. The encoder output x_k^s , x_k^p (1 information bit [systematic] + 1 parity bit).

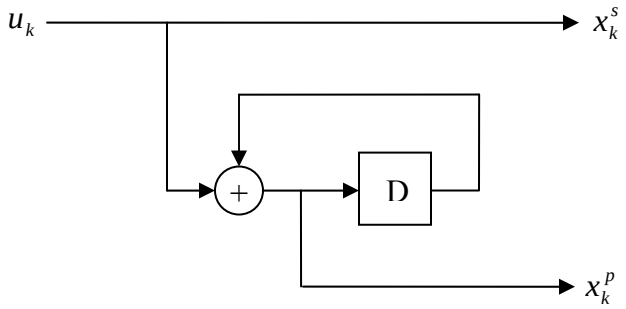


Figure 8. A RSC encoder (2, 1, 1).

Figure 9 shows the Turbo encoder with rate (1/3), consists of two RSC encoder connected in parallel (PCCC). For rate 1/3, The output of the encoder will be as follows

$\bar{x}_k = x_1^s \quad x_1^{p1} \quad x_1^{p2} \quad \ldots\ldots\ldots \quad x_L^s \quad x_L^{p1} \quad x_L^{p2}$. *L … frame length*

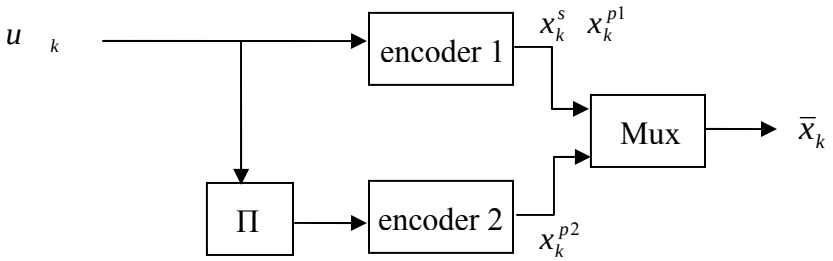


Figure 9. The Turbo encoder with rate (1/3).

Let the input to the *encoder 1* is:

$$\bar{u} = [u_1 \quad u_2 \quad u_3 \quad u_4] = [0 \quad 1 \quad 0 \quad 1]$$

It consists of three *Information* bits, plus one *Tail* bit for terminate *encoder 1* in all-zero state; *(frame length L=4).*

The output of *encoder 1* will be:

$$\bar{x}_1 = [0\,0 \;\; 1\,1 \;\; 0\,1 \;\; 1\,0]$$

Assume we use *block* interleaver, which write row-wise $\begin{pmatrix} u_1 & u_2 \\ u_3 & u_4 \end{pmatrix}$.read column-wise

$\bar{u}_\Pi = (u_1 \quad u_3 \quad u_2 \quad u_4)$

The input to *encoder 2* will be:

$$\bar{u}_\Pi = [u_1 \quad u_3 \quad u_2 \quad u_4] = [0 \quad 0 \quad 1 \quad 1]$$

Encoder 2 left open without termination.

The output of *encoder 2* will be:

$$\bar{x}_2 = [0\,0 \;\; 0\,0 \;\; 1\,1 \;\; 1\,0]$$

The output of the *turbo encoder* will be:

(One *information* bit **+** One *parity* from encoder 1 **+** One *parity* from encoder 2) to produce rate 1/3.

$$\bar{x} = [0\,0\,0 \;\; 1\,1\,0 \;\; 0\,1\,1 \;\; 1\,0\,0]$$

Assume a BPSK modulation is used, the transmitted symbols will be:

$$\bar{v} = [-1\,-1\,-1 \quad +1\,+1\,-1 \quad -1\,+1\,+1 \quad +1\,-1\,-1]$$

Assuming the signal to noise ratio $\frac{E_b}{N_0} = 1.5 \;\; (1.76\ dB)$.

Assume the *channel* is AWGN with variance $\sigma_n^2 = \frac{N_0}{2}$.

$$\bar{y} = \bar{v} + \bar{n}$$

Assume the corrupted *received* symbols are:

$$\bar{y} = [0.38 \;\; 0.32 \;\; -1.90 \;\; -1.30 \;\; 2.78 \;\; -0.60 \;\; -0.98 \;\; 0.59 \;\; -0.54 \;\; 1.22 \;\; -2.37 \;\; -1.84]$$

The received *information* bits will be:

$$\bar{y}_s = [0.38 \;\; -1.30 \;\; -0.98 \;\; 1.22]$$

If we make a *Hard Decision*, then $\hat{u} = [1 \;\; 0 \;\; 0 \;\; 1]$ $(we\ have\ 2\ bits\ with\ error)$

The channel reliability will be, L_c , 2.

Now decoder 1 and decoder 2 can start using the MAP algorithm to correct the errors in the received symbols, following the Iterative decoding procedure with two SISO decoders, described in section III.

Figure 10 shows the *trellis* section of the RSC decoder with the details of $\alpha_k(s)$, $\beta_K(s)$ and $\gamma_K(`s,s)$, which will be calculated for each iteration.

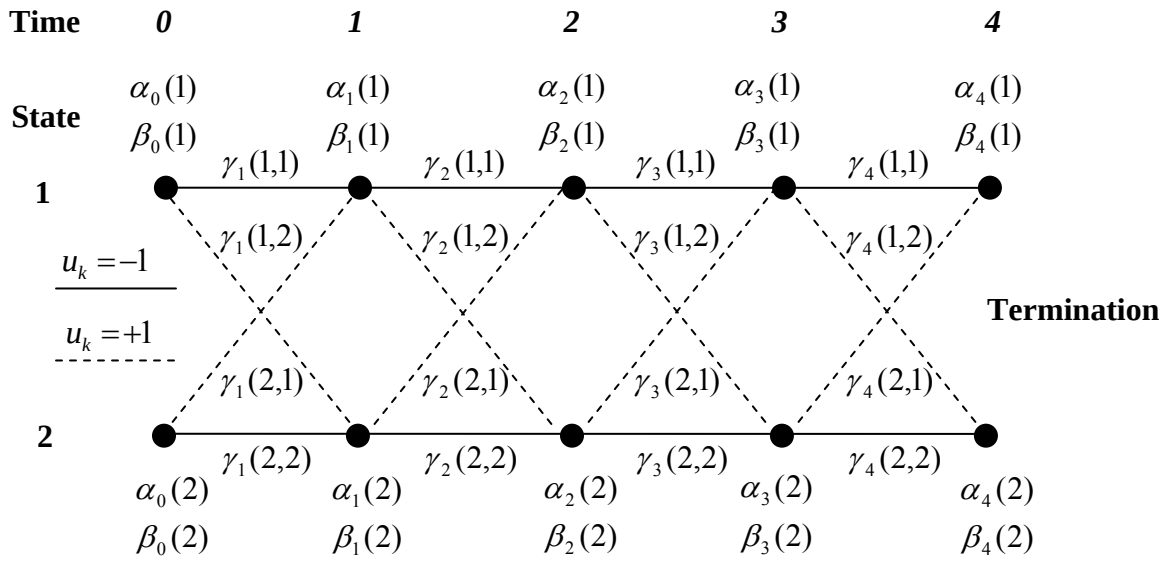


Figure 10. Trellis section of the RSC decoder (2, 1, 1).

- ***Iteration (1) of Decoder (1)***

Decoder 1 will start the *1st* iteration with the inputs from *Channel* and the *apriori* probability as show in Table 1.

Table 1. Input of decoder (1) in iteration (1).

k	1	2	3	4
y_k^s	0.38	-1.30	-0.98	1.22
y_k^{p1}	0.32	2.78	0.59	-2.37
$L_a^1(u_k)$	0	0	0	0

Where, $L_a^1(u_k)=0$ initially set to zeros, starting all-zero state and terminating at all-zero state.

Computation of the Branch Metric $\gamma_K(`s,s)$

$\gamma_K(`s,s)$ is computed by using the equation (12). For example, at time $k=2$,

$$\gamma_2(2,2) = e^{\frac{(-1)}{2}((0)+(2)\cdot(-1.30))} \cdot e^{\frac{(2)}{2}((2.78)\cdot(+1))} = e^{1.3} \cdot e^{2.78} = 59.15$$

$$\gamma_2(2,1) = e^{\frac{(+1)}{2}((0)+(2)\cdot(-1.30))} \cdot e^{\frac{(2)}{2}((2.78)\cdot(-1))} = e^{-1.3} \cdot e^{-2.78} = 0.02$$

Table 2 shows the results of $\gamma_K(`s,s)$ at $u_k=-1$ or $u_k=+1$.

Table 2. Branch metrics of decoder (1) in iteration (1).

k	1	2	3	4
$\gamma_k(1,1)$	0.50	0.23	1.48	3.16
$\gamma_k(1,2)$	2.01	4.39	0.68	0.32
$\gamma_k(2,1)$	1.06	0.02	0.21	36.23
$\gamma_k(2,2)$	0.94	59.15	4.81	0.03

Computing the Forward Trace $\alpha_k(s)$

$\alpha_k(s)$ is computed using the equation (13). For example, at time $k=3$,

$$\alpha_3(1) = \alpha_2(1)\cdot\gamma_3(1,1) + \alpha_2(2)\cdot\gamma_3(2,1)$$

$$\alpha_3(1) = (0.15)\cdot(1.48) + (121.09)\cdot(0.21) = 25.65$$

Table 3. Forward Trace of decoder (1) in iteration (1).

k	0	1	2	3	4
$\alpha_k(1)$	1	0.50	0.15	25.65	21186.48
$\alpha_k(2)$	0	2.01	121.09	582.54	25.68

Computing the Backward Trace $\beta_K(s)$

$\beta_K(s)$ is computed by using the equation (15). For example, at time $k=2$,

$$\beta_2(1) = \beta_3(1)\cdot\gamma_3(1,1) + \beta_3(2)\cdot\gamma_3(1,2)$$

$$\beta_2(1) = (3.16)\cdot(1.48) + (36.23)\cdot(0.68) = 29.31$$

Table 4. Backward Trace of decoder (1) in iteration (1).

k	0	1	2	3	4
$\beta_k(1)$	21186.20	774.68	29.31	3.16	1
$\beta_k(2)$	10547.98	10347.69	174.93	36.23	0

Computing the aposteriori probability $L(u_k)$

$L(u_k)$ is computed by using the equation (11). For example, at time $k=3$,

$$L(u_3) = \ln\left(\frac{\alpha_2(1).\gamma_3(1,2).\beta_3(2) + \alpha_2(2).\gamma_3(2,1).\beta_3(1)}{\alpha_2(1).\gamma_3(1,1).\beta_3(1) + \alpha_2(2).\gamma_3(2,2).\beta_3(2)}\right)$$

$$L(u_3) = \ln\left(\frac{(0.15).(0.68).(36.23) + (121.09).(0.21).(3.16)}{(0.15).(1.48).(3.16) + (121.09).(4.81).(36.23)}\right) = \ln\left(\frac{84.05}{21102.6}\right) = -5.52$$

$$L_e^1(\hat{u}_3) = L^1(\hat{u}_3) - [L_c.y_3^s + L_a^1(u_3)]$$

$$L_e^1(\hat{u}_3) = (-5.52) - [(2).(-0.98) + (0)] = -3.56$$

Table5. Output of decoder (1) in iteration (1).

k	1	2	3	4
$L_e^1(u_k)$	3.22	-1.39	-3.56	3.12
$L^1(u_k)$	3.98	-3.99	-5.52	5.56
Hard Decision	1	0	0	1

$\tilde{u} = sign\left[L^1(\hat{u})\right] = [1 \quad 0 \quad 0 \quad 1]$

Number of errors: 2

- ***Iteration (1) of Decoder (2)***

Decoder (2) will start the *1st* iteration with the inputs shown in Table 5.

Table 6. Input of decoder (2) in iteration (1).

k	1	2	3	4
y_k^s	0.38	-0.98	-1.30	1.22
y_k^{p2}	-1.90	-0.60	-0.54	-1.84
$L_a^2(u_k)$	3.22	-3.56	-1.39	3.12

Where, $L_a^2(u_k) = \Pi\left\{L_e^1(u_k)\right\}$ Interleaving of the extrinsic LLR of decoder (1), and assuming starting at all-zero state, and terminating as following:

$\beta_4(s) = \frac{1}{2} \quad \forall\ s$ i.e. decoder (2) left open, according to equation (17) Table 7 shows the results of $\gamma_K(`s, s)$.

Table 7. Branch metrics of decoder (2) in iteration (1).

k	1	2	3	4
$\gamma_k(1,1)$	0.91	28.79	12.62	0.39
$\gamma_k(1,2)$	1.09	0.03	0.08	2.56
$\gamma_k(2,1)$	48.91	0.11	0.23	101.49
$\gamma_k(2,2)$	0.02	8.67	4.28	0.01

Table 8. Forward Trace of decoder (2) in iteration (1).

k	0	1	2	3	4
$\alpha_k(1)$	1	0.91	26.32	334.34	4461.98
$\alpha_k(2)$	0	1.09	9.48	42.68	856.34

Table 9. Backward Trace of decoder (2) in iteration (1).

k	0	1	2	3	4
$\beta_k(1)$	2656.91	657.47	22.61	1.47	0.5
$\beta_k(2)$	32194.63	1888.64	217.55	50.75	0.5

Table 10. Output of decoder (2) in iteration (1).

k	1	2	3	4
$L_e^2(u_k)$	-2.75	-0.2	0.85	-1.88
$L^2(u_k)$	1.23	-5.72	-3.14	3.68
Hard Decision	1	0	0	1

Then, $\hat{u} = \Pi^{-1}\left\{ L^2(u_k) \right\} = [1.23 \quad -3.14 \quad -5.72 \quad 3.68]$

$\tilde{u} = sign\left[L^2(\hat{u})\right] = [1 \quad 0 \quad 0 \quad 1]$

Number of errors: 2

- ***Iteration (2) of Decoder (1)***

Decoder (1) will start the *2nd* iteration with the input shown in Table 10.

Table 11. Input of decoder (1) in iteration (2).

k	1	2	3	4
y_k^s	0.38	-1.30	-0.98	1.22
y_k^{p1}	0.32	2.78	0.59	-2.37
$L_a^1(u_k)$	-2.75	0.85	-0.2	-1.88

Where, $L_a^1(u_k) = \Pi^{-1}\left\{ L_e^2(u_k) \right\}$ Deinterleaving of the extrinsic LLR of decoder (2). Table 12. shows the results of $\gamma_K(`s,s)$.

Table 12. Branch metrics of decoder (1) in iteration (2).

k	1	2	3	4
$\gamma_k(1,1)$	1.96	0.15	1.63	8.08
$\gamma_k(1,2)$	0.51	6.72	0.61	0.12
$\gamma_k(2,1)$	0.27	0.02	0.19	14.15
$\gamma_k(2,2)$	3.72	38.67	5.31	0.07

Table 13. Forward Trace of decoder (1) in iteration (2).

k	0	1	2	3	4
$\alpha_k(1)$	1	1.96	0.3	6.74	2528.3
$\alpha_k(2)$	0	0.51	32.89	174.83	13.05

Table 14. Backward Trace of decoder (1) in iteration (2).

k	0	1	2	3	4
$\beta_k(1)$	2528.52	518.49	21.8	8.08	1
$\beta_k(2)$	11170.76	2965.26	76.67	14.15	0

Table 15. Output of decoder (1) in iteration (2).

k	1	2	3	4
$L_e^1(u_k)$	2.39	1.34	-1.68	3.26
$L^1(u_k)$	0.4	-0.41	-3.84	3.82
Hard Decision	1	0	0	1

$\tilde{u} = sign\left[L^1(\hat{u})\right] = [1 \quad 0 \quad 0 \quad 1]$

Number of errors: 2

- ***Iteration (2) of Decoder (2)***

Decoder (2) will start the *2nd* iteration with the input show in Table 15.

Table 16. input of decoder (2) in iteration (2).

k	1	2	3	4
y_k^s	0.38	-0.98	-1.30	1.22
y_k^{p2}	-1.90	-0.60	-0.54	-1.84
$L_a^2(u_k)$	2.39	-1.68	1.34	3.26

Where, $L_a^2(u_k) = \Pi\left\{ L_e^1(u_k) \right\}$ Interleaving of the extrinsic LLR of decoder (1)

Table 17 shows the results of $\gamma_K(`s,s)$.

Table 17. Branch metrics of decoder (2) in iteration (2).

k	1	2	3	4
$\gamma_k(1,1)$	1.38	11.24	3.22	0.36
$\gamma_k(1,2)$	0.72	0.09	0.31	2.74
$\gamma_k(2,1)$	32.30	0.29	0.91	108.85
$\gamma_k(2,2)$	0.03	3.39	1.09	0.01

Table 18. Forward Trace of decoder (2) in iteration (2).

k	0	1	2	3	4
$\alpha_k(1)$	1	1.38	15.72	52.95	852.85
$\alpha_k(2)$	0	0.72	2.56	7.66	145.16

Table 19. Backward Trace of decoder (2) in iteration (2).

k	0	1	2	3	4
$\beta_k(1)$	499.43	251.17	21.86	1.55	0.5
$\beta_k(2)$	8119.16	212.25	60.74	54.43	0.5

Table 20. Output of decoder (2) in iteration (2).

k	1	2	3	4
$L_e^2(u_k)$	-3.97	-0.05	1.41	-1.77
$L^2(u_k)$	-0.82	-3.69	0.15	3.93
Hard Decision	0	0	1	1

Then, $\hat{u} = \Pi^{-1}\left\{ L^2(u_k) \right\} = \left[-0.82 \quad 0.15 \quad -3.69 \quad 3.93\right]$

$\tilde{u} = sign\left[L^2(\hat{u})\right] = [0 \quad 1 \quad 0 \quad 1]$, which is the same as the input to the turbo encoder.

Number of errors: 0 (after 2 iterations)

V. CONCLUSIONS

The concept of iterative decoding is discussed in detail and a numerical example of turbo code system with two iterations was presented. The maximum aposteriori probability (MAP) algorithm is used as a soft input soft out (SISO) component decoder. The MAP algorithm examines *every* possible path through the convolutional decoder trellis and provides not only the estimated bit sequence, but also the *probabilities* for each bit that it has been decoded correctly. Care must be taken to avoid numerical overflow problems in the recursive calculation of $\alpha_{k-1}(`s)$ and $\beta_K(s)$, but such problems

can be avoided by normalization of these values. The choice of the numbers in the numerical example were based many trials to converge the iterative process to only two iterations to fix all the errors encountered.

REFRENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: TURBO-CODES," *in Proceeding of 1993 IEEE International Conference On Communication* (Geneva, Switzerland, 1993), pp. 1064-1070.

[2] J. Hagenauer and P. Hoeher, "A Vitrbi Algorithm with Soft-Decision Outputs and its Applications," *Proc. of Global Communications Globecom'89*, Dallas, Vol. 3, pp. 47.1.1-47.1.7, Nov. 1989.

[3] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, pp. 284-287, March 1974.

[4] J. Hagenauer, E. Offer, and L. Papke, "Decoding Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, Vol. 42, No. 2, pp. 429-445, Mar. 1996.

[5] A. I. Mahroos, "Turbo Codes for Mobile and Satellite Communication Systems," *M.Sc. thesis, Military Technical College, Cairo, 2006*.