Class Project
Georgia Tech Atlanta
FALL 2010

Turbo codes Encoding/Decoding & EXIT charts

Walid Boumerdassi, Etienne Collange & Team Space Busters

December 9, 2010

Abstract

TURBO codes are the channel coding scheme used in wireless cellular networks as they are able to reach nearer to the Shannon limit. In this Section we introduce the basic definitions and notions in turbo codes, using different research articles, we were able to to implement a turbo encoder/decoder. The decoding scheme relying on the BCJR algorithm, the maximum aposteriori algoithm (MAP). Also, one of the most important part is the analysis of the code, using an implementation of the EXIT charts [4] we were able to study the behaviour of our turbo codes: The evolution of the information exchanged in the structure of the decoder at each iteration. All the important simulations and result are analysed in the last part of this document.

1 Introduction

Over the years, there has been a spectacular growth in digital communications especially in the fields of cellular/PCS, satellite, and computer communication. In these communication systems, the information is represented as a sequence of binary bits. This information is then mapped, modulated and transported in communication channels, which introduce noise and losses. For these reasons it is important to conceive efficient coding algorithms. In the last few decades, important breakthroughs have been made, using LDPC and Turbo codes enabling telecommunication channels to approach nearer to the Shannon channel capacity limit.
We will organize this paper, explaining our step by step approach, introducing at each parts the important notions and materials, which helped our understanding of Turbo codes. Hence, we first describe the encoding process relying on recursive convolution and including the interleaving process. Then, we will describe the BCJR algorithm relating [1][2], which enabled us to implement the decoding circuit relying on a loop and using two MAP block decoders. Finally, we will use EXIT charts to analyse the behaviour of Turbo Codes.

2 Turbo Encoding

The fundamental turbo encoding is built using two identical recursive systematic convolutional (RSC) code with parallel concatenation. The encoder used in our experimentation is an RSC with $r = \frac{1}{2}$. The two block encoders are separated by an interleaver. Only one of the systematic outputs from the two component encoders is used, because the systematic

output from the other component encoder is just a permuted version of the chosen systematic output. Figure 1 shows the structure of our turbo encoder using two RSC blocks and a random interleaver.

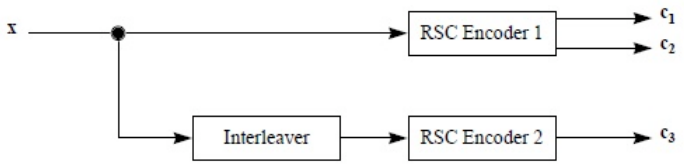

Figure 1: Turbo Encoder

In this figure we can see three outputs, two of them issued by RSC encoder 1 where c_1 is the systematic sequence and c_2 the recursive convolutional sequence. Also, it is important to notice that from the RSC encoder 2; we only issue the convolutional part c_3, the other output is discarded.

2.1 Recursive Systematic Convolutional (RSC) Encoder

In this section, we will explain in more detail the RSC encoding process. As we already mentioned, the turbo encoder uses convolutional encoder and feedback one of its encoded output to the input. Figure 2 shows the convolutional encoder used in our experimentation. This code is represented by the

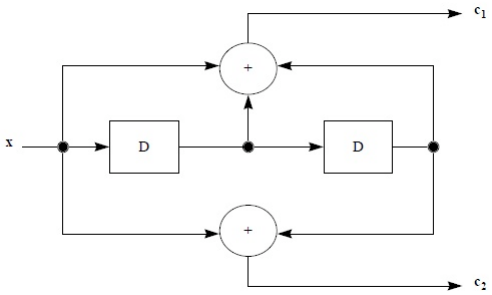

Figure 2: convolutional encoder G= [111 ; 101]

generator matrix $G = [111; 101]$. Relying on the convolutional structure the RSC encoder is obtained by feeding back the output c_1 form the previous figure to the input. Thus the final structure of our RSC encoder is obtained on Figure 3 the output c_1 is the systematic sequence and c_2 the encoded sequence.

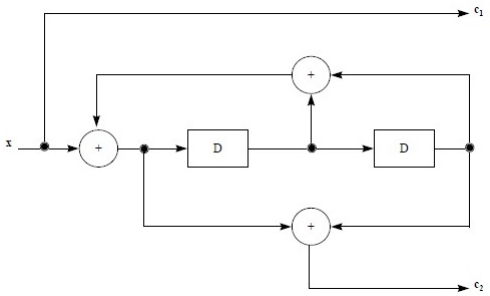

Figure 3: RSC encoder, G= [111 ; 101]

2.2 interleaving

Interleaving is a process of rearranging the ordering of a data. The inverse of this process is called deinterleaving which restores the received sequence to its original order (used in the turbo decoding). Interleaving is a practical technique to enhance the error correcting capability of coding. In turbo coding, interleaving is used before the information data is encoded by the second component encoder (see figure[1]). The basic role of an interleaver is to construct a long block code from small memory convolutional codes, as long codes can approach the Shannon capacity limit. Secondly, it spreads out burst errors [3]. The interleaver provides a scrambled data to the RSC encoder 2 and decorrelates the inputs to the two decoder blocks, so that the BCJR algorithm based on uncorrelated information exchange between the two component decoders can be applied. The final role of the interleaver is to break low weight input sequences, and hence increase the code free Hamming distance or reduce the number of codewords with small distances in the code distance spectrum. The interleaver plays a major role in the performance of turbo codes. There are different interleaving types, which could be implemented in the process of coding and decoding of turbo codes. In our experiment we used a random interleaver that uses a fixed random random permutation in order to scramble the data.

2.3 The Trellis Diagram

The construction of the trellis diagram enables us to describe the behaviour of the turbo encoder and is a key element in the process of decoding. This Trellis representation is obtained from its state diagram. (see figure 4)

For the turbo encoder, the trellis is terminated by inserting m = K-1 additional bits after the input sequence, where K is the number of columns in the generator matrix G. These

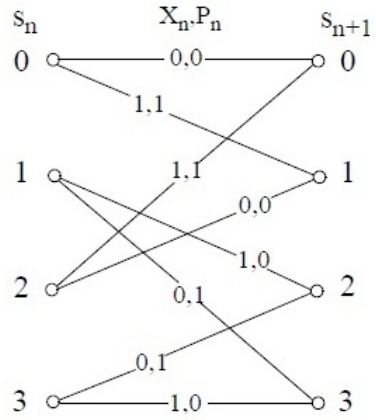

Figure 4: (State Diagram of the encoder

additional bits drive the encoder to the all-zero state (trellis termination).
Using the different described blocks: RSC encoder, Interleaver and Trellis Termination, we were able to implement our Turbo Encoder shown on Figure 1.

3 Turbo Decoding

In this section, we will describe the process of decoding using the BCJR algorithm.
Actually, The structure of a parallel turbo decoder is show on Figure 5. It consists of a pair of decoders which work cooperatively in order to refine and improve the estimate of the original bits. The decoder are based on the BCJR algorithm, also called MAP (Maximum aposteriori probability) algorithm. The operation relies on soft decision information learned from each other.

After initialization the soft decision of one decoder noted Le and named the extrinsic information (described in next section 3.2) is used to initialize the other decoder. The decoded information is cycled around the loop until the soft decisions converge on stable set of values. In this case, we use the last extrinsic information issued from the first decoder to compute the estimate values of the message.

3.1 BCJR algorithm

The BCJR algorithm is an algorithm for MAP decoding of error correcting codes defined on trellises (principally convolutional codes). The algorithm is named after its inventors: Bahl, Cocke, Jelinek and Raviv . This algorithm is critical to modern iteratively-decoded error-correcting codes including

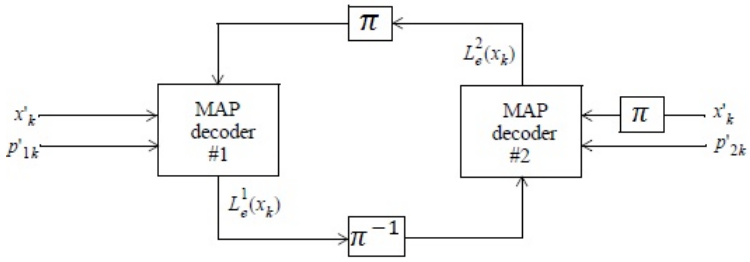

Figure 5: Turbo decoder circuit

turbo codes and low-density parity-check codes. In this section we will define the BCJR algorithm using [1][2]. Let us first introduce the different notation:

- The transmitted message elements is $y_{k,i} = \{x_k, p_k^i\}$ depending on the encoder output $i \in \{1, 2\}$, where x_k is the systematic code, p_k^1 the parity check code from RSC encoder 1 and p_k^2 RSC encoder 2.
- The received code elements is $y'_{k,i} = \{x'_k, p'_{i,k}\}$ $i \in 1, 2$ and the vector on the received code depending on the encoder's outputs is **y'** $= y'_{1,N} = \{y'_1, y'_2, ..., y'_N\}$, where N is the length of the message
- The Extrinsic information $Le_{i,j}$ form decoder i to decoder j, where $(i, j) \in \{1, 2\}^2$
- The Apriori information $L_{a,k}$ form decoder $k \in \{1, 2\}$
- The MAP soft decision :

$$L_{map}(x_k) = log\{\frac{Pr(x_k=+1|\mathbf{y'})}{Pr(x_k=-1|\mathbf{y'})}\}$$

Hence, using the Trellis notation introduced in Figure 4, where $s_k \in S$ is the state of the encoder at time k, S^+ is the set of pairs (s', s) which corresponds to the transition $(s_{k-1} = s') \rightarrow (s_k = s)$ caused by the data input $x_k = +1$. Similarly, S^- is defined for data input $x_k = -1$. We remark that by using the Baye's theorem at the numerator and denominator, we can simplify this expression by suppressing $P(\mathbf{y'})$. Then we obtain this expression:

$$\frac{Pr(x_k = +1|y')}{Pr(x_k = -1|y')} = \frac{\sum_{(s',s)\in S^+} Pr(s_{k-1} = s', s_k = s, y')}{\sum_{(s',s)\in S^-} Pr(s_{k-1} = s', s_k = s, y')}$$

Furthermore, the development of the element in this expression will introduce other function (γ_k, α_k and β_k) such that:

$$Pr(s_{k-1} = s', s_k = s, y'_{1,N}) = Pr(s_{k-1} = s', y'_{1,k-1}).$$

$$Pr(s_k = s, y'_k | s_{k-1} = s').Pr(y'_{k+1,N} | s_k = s) = \alpha_{k-1}(s').\gamma_k(s', s).\beta_k(s)$$

In the following, we will introduce the meanings of these functions, the different steps and calculations used to obtain these expressions are given in reference [1][2]. Hence, the function $\alpha_k(s')$ is the probability of arriving at a branch in a particular state with the sequence of the noisy observations $y'_{1,k} = \{y'_1, y'_2, ..., y'_k\}$, which lead to that state. Using forward recursion we can express this function in terms $\gamma_k(s', s)$:

$$\alpha_k(s) = Pr(s_k = s, y'_{1,k})$$
$$= \sum_{s'} Pr(s_{k-1} = s', y'_{1,k-1}).Pr(s_k = s, y'_k | s_{k-1} = s')$$
$$= \sum_{s'} \alpha_{k-1}(s').\gamma_k(s', s)$$

However, in the implementation of the BCJR we observed using MATLAB some instabilities, with further reading and using [2], we were able to correct this, by normalizing these functions, thus we used:

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

The function $\beta_k(s)$ corresponds to the probability of an existing branch via a particular state s and a sequence of noisy observations $y'_{k+1,N} = y'_{k+1}, y'_{k+2}, ..., y'_N$ which finish off the trellis. This time, using a backward recursion we can express these in terms of $\gamma_k(s', s)$:

$$\beta_k(s') = Pr(y'_{k+1,N} | s_k = s)$$
$$= \sum_{s} Pr(s_{k+1} = s, y'_{k+1} | s_k = s').Pr(y'_{k+2,N} | s_k = s')$$
$$= \sum_{s'} \gamma_{k+1}(s', s).\beta_{k+1}(s)$$

For the same reason of stability, we normalize these coefficient and use:

$$\beta_k(s') = \frac{\sum_{s'} \beta_{k+1}(s).\gamma_{k+1}(s', s)}{\sum_s \sum_{s'} \beta_{k+1}(s').\gamma_{k+1}(s', s)} \quad (2)$$

Finally, the function $\gamma_k(s', s)$ represents the probability of the transmitted signal an parity bit y'_k knowing the previous state. This function can be approximated by:

$$exp\{\frac{1}{2}(x_k L_a(x_k) + x_k L_c x'_k + p_k L_c p'_k)\} \quad (3)$$

Where L_a is the apriori information issued from the previous decoder, $L_c = \frac{2}{\sigma^2}$ depends on the SNR used. We also use the Trellis matrix expressing x_k and p_k for a certain state transition. Finally x'_k and p'_k are respectively the noisy message and parity bit at the reception and p'_k depends on the decoder.

3.2 implementation of the BCJR Algorithm

In this section we will describe the algorithm implemented on MATLAB in order to decode turbo codes. We will explain step by step and using the notation introduced at the beginning of this section and also referring to[2]:

=======Initialisation==========

Decoder 1:

- $\alpha_0^1(s) = \delta_{s,0}$
- $\beta_N^1(s) = \delta_{s,0}$
- $Le_{2,1}(x_k) = 0$ for $k = 1, 2..., N$ There were no exchange from D2-> D1 we are in the first iteration

Decoder 2:

- $\alpha_0^2(s) = \delta_{s,0}$
- $\beta_N^2(s) = \alpha_N^2(s)$ for all s (set after computation of $alpha_N^2(s)$ in the first iteration)
- $Le_{1,2}(x_k)$ for $k = 1, 2..., N$ we obtain the extrinsic information form D1 after the first iteration, and use this interleaved value as the apriori information L_a^2

==========

======The n^{th} iteration===

Decoder 1:

for $k = 1 : N$

- Get $y'_k = (x'_k, p'_{1,k})$
- First compute $\gamma_k(s's)$ from eq(3) for all allowable state transitions where the apriori information is $L_a^1 = \pi^{-1}(Le_{2,1})$ the deinterleaved extrinsic information from the Decoder 2
- Compute $\alpha_k^1(s)$ for all s using eq(1)

end

for $k = N : -1 : 2$

- Compute $\beta_{k-1}^1(s)$ for all s using eq(2)

end

for $k = 1 : N$

- Compute $Le_{1,2}(x_k) = log\{\frac{\sum_{s \in S^+} \alpha_{k-1}^1(s').\gamma_k(s',s).\beta_k^1(s)}{\sum_{s \in S^-} \alpha_{k-1}^1(s').\gamma_k(s',s).\beta_k^1(s)}\}$ using the Trellis function see Figure 6 for summing on respectively on S^+ and S^-

end

Decoder 2:

for $k = 1 : N$

- Get $y'_k = (\pi(x'_k), p'_{2,k})$, the interleaved message and noisy parity check 2 form encoder 2.
- Compute $\gamma_k(s's)$ from eq(3) for all allowable state transitions where the apriori information is $L_a^2 = \pi(Le_{1,2})$ the interleaved extrinsic information from the Decoder 1, becomes the apriori information.
- Compute $\alpha_k^2(s)$ for all s using eq(1)

end
for $k = N : -1 : 2$

- Compute $\beta_{k-1}^2(s)$ for all s using eq(2)

end
for $k = 1 : N$

- Compute
  $Le_{2,1}(x_k) = log\{\frac{\sum_{s\in S^+} \alpha_{k-1}^2(s').\gamma_k(s',s).\beta_k^2(s)}{\sum_{s\in S^-} \alpha_{k-1}^2(s').\gamma_k(s',s).\beta_k^2(s)}\}$

end
==========
==== After the last iteration==
for $k = 1 : N$

- compute the L_{map1} issued form decoder 1
  $L_{map}(x_k) = L_a(x_k) + L_c x'_k + Le_{1,2}$
  Using the values obtained at the last iteration of the code.
- if $L_{map}(x_k) > 0$
  decide $x_k = +1$
  else
  decide $x_k = -1$ end

==========

4 EXIT charts

An EXIT chart, which stands for Extrinsic information transfer chart, can be seen as a tool to aid the construction of good iteratively-decoded error-correcting codes, especially Turbo Codes and Low-Density Parity-Check Codes. EXIT

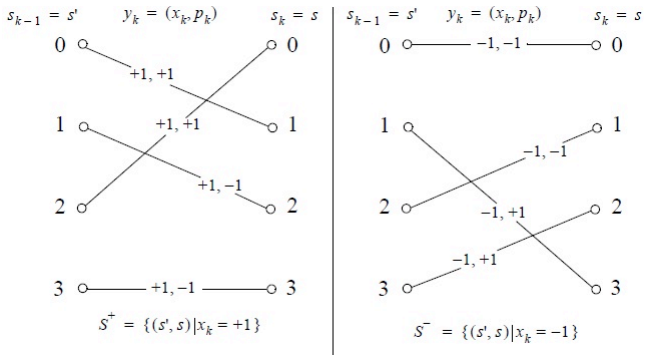

Figure 6: Transition states for $G = [111; 101]$ used in the calculation of the coefficientγ, where S^+ corresponds to the transition (s', s) with $x_k = +1$ in the Trellis Diagram

charts were developed by Stephan ten Brink, who used the concept of extrinsic information developed in the Turbo coding community.
Basically, in the context of turbo codes, an EXIT chart is the reunion of two curves that characterize the two decoders used in a Turbo decoder. Each curve represents a relation between the input and the output of one decoder, this relation is the mutual information between the output of the decoder (Le: the extrinsic information) and the initial message that was encoded before passing through the channel, with respect to the mutual information between the input of the decoder(La: the a priori information) and the message :

$$I(x, L_e) = F(I(x, L_a))$$

As we have seen before, in a Turbo decoder the extrinsic information of the first decoder (Le_1) is used as the a priori information of the second decoder (La_2) and vice versa. Hence, by plotting on the same graph:

$$I(x, Le_1) = F(I(x, La_1))$$
$$I(x, La_2) = F(I(x, Le_2))$$

We will be able to evaluate the minimal number of iterations that will be needed for the Turbo decoder to converge. The two curve necessarily cross for a mutual information of 1 because if the a priori information is entirely correlated with the initial message ($I(x, La_1) = 1$) then the decoder will necessarily outputs an extrinsic information entirely correlated with the initial message to ($I(x, Le_1) = 1$). Therefore, by jumping from one curve to the other until we reach a mutual information near from the unity, we obtain the trajectory of the iterative decoder and the number of

jumps is an estimation of the minimal number of iteration in which the Turbo decoder converges. On this graph, we can

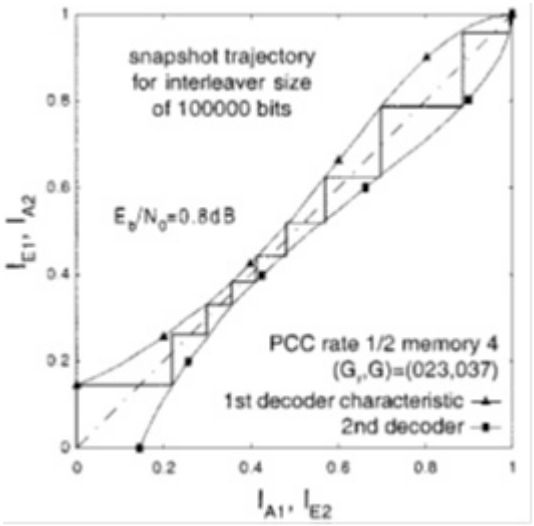

Figure 7: Example of an EXIT chart [4]

find a typical EXIT charts with the trajectory of the decoder between the two curves of $Ie_1(Ia_1)$ and $Ia_2(Ie_2)$. In order to trace EXIT charts of the turbo codes that we implemented, we used a method described by Stephan ten Brink himself in the paper "Convergence Behaviour of Iteratively Decoded Parallel Concatenated Codes" [4], where he explains how to trace the curve $Ie(Ia)$ for a single convolutional decoder. The idea is to try to decode a encoded message with a range of different a priori information that is chosen specifically to obtain a range of Ia that is well spread between 0 and 1. A good repartition for La is obtained with a variation of μ in the following expression:

$$La = \mu.x + n$$

Where x is the initial message and n is a Gaussian noise with variance $\sqrt{(}2.\mu)$.For each La, we have to compute Ia and Ie with the following formulas:

$$Ia = \frac{1}{2}.\sum_{x=-1,+1}\int_{-\infty}^{+\infty} P_a(\varepsilon|X=x) \times log2\{\frac{2P_a(\varepsilon|X=x)}{P_a(\varepsilon|X=1)+P_a(\varepsilon|X=+1)}\}d\varepsilon$$

$$Ie = \frac{1}{2}.\sum_{x=-1,+1}\int_{-\infty}^{+\infty} P_e(\varepsilon|X=x) \times log2\{\frac{2P_e(\varepsilon|X=x)}{P_e(\varepsilon|X=1)+P_e(\varepsilon|X=+1)}\}d\varepsilon$$

Where P_a and P_e are respectively the probability density functions of La and Le, which we will estimate with Monte Carlo simulation. Note that x is a block of thousands of bits so La has also a thousands values and we can estimates P_a by histogram measurements, Selecting bits of La where x is 1 and computing an histogram of this values, we get an estimations of $P_a(\varepsilon|X=+1)$. Similarly, by selecting bits of La where x is -1, we get an estimations of $P_a(\varepsilon|X=-1)$. The same method is used to compute P_e, thus we can plot the EXIT chart.

4.1 Simulation

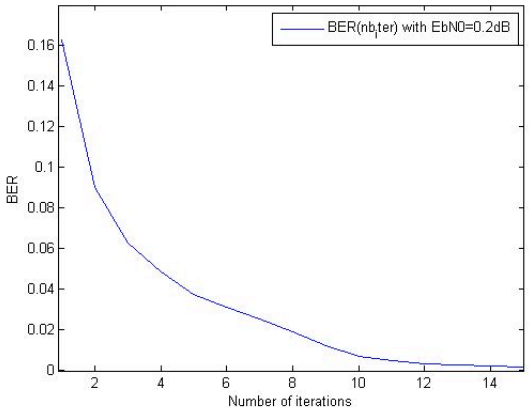

Figure 8: The evolution of the BER in function of the number of iterations

We realized several simulations the implemented Turbo code. The first simulation on Figure 8, was realised to verify if the decoder was well implemented by looking at the evolution of the Bit Error Rate after each iterations. To this end, we just had to perform soft decoding at the end of each iteration to see that the decoded message is getting closer and closer to the initial message when the number of iteration is increasing.
On the previous curve, we can see that the BER decreases exponentially and approach 0 as the number of iteration increases.

The second simulation consists in monitoring the evolution of the Bit Error Rate at the end of the decoder when the Signal to Noise Ratio evolves. In order to obtain significant values for the BER, we had to generate as many input messages as needed to reach a certain amount of errors for each values of the SNR. If we had just sent one message, the number of errors would have been too low to consider the

BER as a significant value. The used code with the generator matrix [1 1 1; 1 0 1], is unpunctured on Figure 9 we have an example of the evolution of the BER with only 6 iterations:

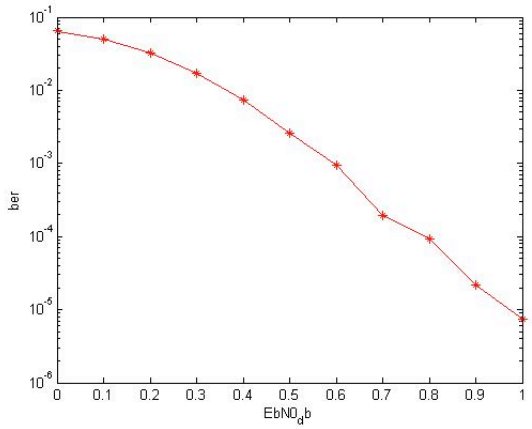

Figure 9: BER in function of different values of the SNR expressed in dB

The BER decreases regularly from 0 to 0.7dB but we can notice that after 0.7dB, the BER is decreasing more slowly. Finally, fur further studies of our implementation, we realized an EXIT chart analysis of our decoder. In order to do that, we had to write another code to implement the Ten Brink's method that was previously presented. This code allows us to plot the mutual information (Ie) between the extrinsic information at the output of convolutional decoder and the message, with respect to the mutual information (Ia) between the a priori information at the input of convolutional decoder and the message for a certain range of signal to noise ratios: We can see on Figure 10 that when the SNR increases, the extrinsic information Ie augments, in addition we can obtain high values of extrinsic information even when Ia is null. This is consistent because when the channel is good, the input of the decoder contains few errors so even with an apriori information totally uncorrelated with the initial message, the decoder can output an estimation of the message that make sense, but when the channel introduces more noise, the decoder cannot estimate the message without an efficient a priori information that has a good apriori informationIa.
We can also notice that hopefully all curve converge to the point (1,1), this is because when Ia equals 1 then the a priori information of the decoder is perfectly correlated to the initial message so the extrinsic information at the output

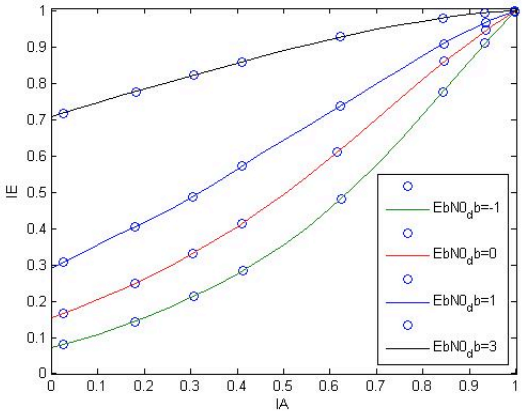

Figure 10: This Graph represents the Extrinsic information expressed in function of the Apriori information

must be perfectly correlated to the initial message too, if it was not the case, it would mean the decoder has a negative effect because it would lose information about the message. Then we choose to plot the EXIT chart for a signal to noise ratio of 0.2dB just as an example because it gives a good shape to understand the trajectory of the decoder during a simulation Figure 11.

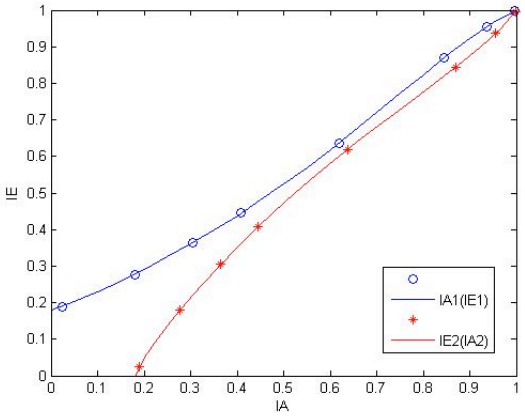

Figure 11: Theoretical EXIT chart with one decoder at SNR=0.2dB

The next graphs on Figure 12,13 and 14 show the EXIT charts for different values of SNRs. Also, it is important to mention that the black curve represents the trajectory of our decoder when we perform a simulation with the same SNR. We insist on the fact that the EXIT chart is predicted outside and before the simulation whereas the trajectory is computed during each iteration within the simulation. In the following graphs we can that the two match quite well.
Actually, we can observes that the decoder is always near one of the two curves of the EXIT chart before they converge to a central point, when the decoder converges.

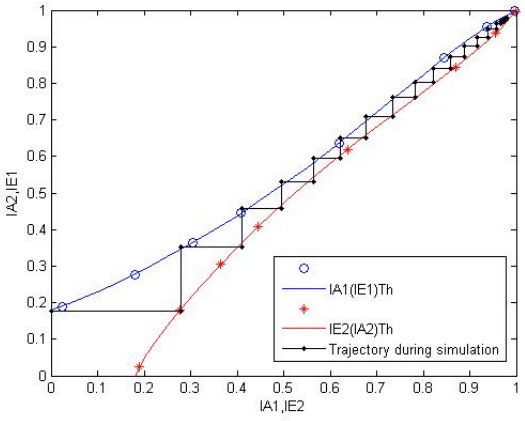

Figure 12: EXIT chart and trajectory of the decoder for EbN0= 0.2 dB

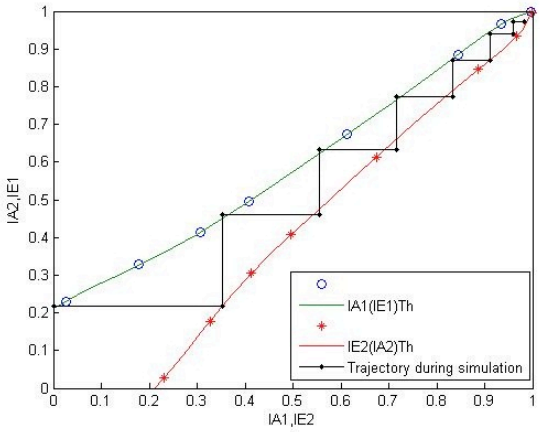

Figure 13: EXIT chart and trajectory of the decoder for EbN0= 0.5 dB

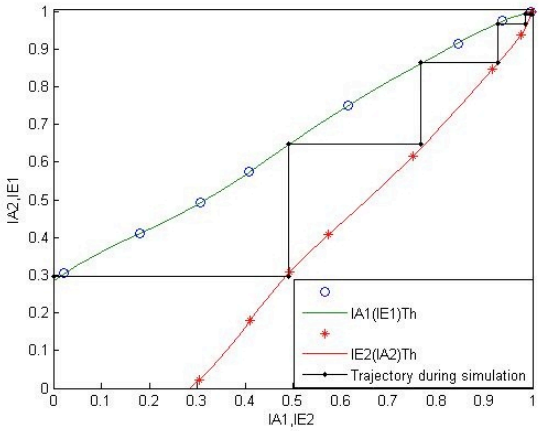

Figure 14: EXIT chart and trajectory of the decoder for EbN0=1 dB

5 Conclusion

The understanding of the BCJR algorithm, enabled us to succeed in implementing a Turbo Code on MATLAB. We realized several analyses using the EXIT charts representation, which show similar results compared to the original Ten Brink's work. Further studies could be realized using different mapping at the level of the encoder and the interleaver(diagonal interleavers, circular-shifting interleavers, etc.). We tried to propose a scalable structure to our source code on MATLAB for this purpose.

Appendix: Instructions to run the code

The source codes includes files:

- ExitchartsPredicted.m
- trellis.m
- TurboCode_ Simulation.m
- TurboDecode1.m
- TurboEncode.m

Instructions to run the code: At first you have to unzip our archive, keep all the matlab files in the same folder and run the programs from this folder.
There are two main programs that you can launch:

1. TurboCode_ Simulation.m is the main program of our Turbo code simulation. If you run it, it will launch an analysis of the BER with respect to a range of SNR and plot the result. You can easily modify some parameters at the beginning of this file to adapt the simulation:

   - The number of iterations that will be performed in the turbo decoder. (nb_ iter at line 9)
   - The range of SNRs that will be used to evaluate BER's evolution. (EbN0_ db_ vect at line 18)
   - Be careful, with a SNR too high or too many iterations, the simulation can become very long!

   You can also modify two parameters in order to run the simulation for different purposes:

   - EXIT at line 2: Put the value 1 if you want to draw the trajectory of the decoder in the EXIT chart during the simulation. You will have to run the other program (EXIT chartsPredicted.m) and merge the curves to have graphs similar to those in our report.
   - BER_ IT at line 6: Put this the value 1 if you want to draw the BER with respect to the number of iterations.

   When EXIT=1 or BER_ IT=1, EbN0_ db_ vect becomes only one value of SNR at which we want to evaluate the EXIT chart, you have to set up this value in the (if EXIT==1 | BER_ IT==1) at line 15.

2. EXIT chartsPredicted.m is the program we used to plot EXIT charts. The only parameter that you can change is the range of SNR (EbN0_ db_ vect at line 5) on which you want to plot the EXIT charts. Usually we put only a value in EbN0_ db_ vect because it can become messy when several EXIT charts are plotted in the same graph. To obtain an EXIT chart with the corresponding trajectory of our decoder for a certain SNR, proceed as follows:

   - Launch EXIT chartsPredicted.m with the wanted SNR (e.g.: EbN0_ db_ vect= 0.5)
   - A figure with the EXIT chart will appear, do not close it, and run TurboCode_ Simulation.m with EXIT=1 and the same SNR (e.g.: EbN0_ db_ vect= 0.5).
   - The trajectory will appear on the previous figure. If the trajectory does not reach the point of convergence, you have to start again from 1 and increase the number of iterations in TurboCode_ Simulation.m so that there are enough iterations to achieve convergence.

References

[1] *EXIT Chart Analysis for Compressive Turbo Codes* - Bilal Riaz Department of Electrical and Computer Engineering McGill University Montréal,Canada, 2009, 126 pages

[2] *A turbo Code Tutorial* - William E. Ryan, New mexico State university

[3] *Optimal choice of interleaver for turbo codes.* - Shobha Rekh, Dr.S.Subha rani, Dr.A.Shanmugam, Academic Open internet Journal Volume 15, 2005

[4] *Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes* - Stephan ten Brink, Member IEEE, IEEE Transaction on Communications, VOL. 49, No. 10, October 2001

[5] *Evaluation of Soft Output Decoding for Turbo Codes* - Huang Fu-hua, Master's Thesis, 1997-05-29

[6] *Bit Error Rate Estimation for Turbo Decoding* - Nick Letzepis, Member, IEEE, and Alex Grant, Senior Member, IEEE, IEEE Transaction on Communications, Vol. 57, NO. 3, March 2009

[7] *The EXIT Chart Ű Introduction to extrinsic information transfer in iterative processing* - Joachim Hagenauer, Institute of Communications Engineering (LNT), Munich University of Technology (TUM); 2005