THE EFFECT OF DECODING ITERATIONS ON THE PERFORMANCE OF TURBO CODE

H. H. Abbas and Ibtesam Raheem Karhiy
College Of Engineering University Of Technology
University Of Baghdad
Baghdad, Iraq

ABSTRACT:

This paper introduces the effect of decoding iterations on the performance of a new class of convolutional codes called Turbo Code. Turbo Code encoder is built using a parallel concatenation of two Recursive Systematic Convolutional (RSC) codes. The associated decoder is implemented using a feedback-decoding rule in an iterative manner.

1. INTRODUCTION:

A useful tool in the design of reliable digital communication systems is channel coding. Channel coding provides improved error performance for digital communication systems by mapping input sequences into code sequences, which inserts redundancy and memory to the transmission. Information theory states that arbitrarily small error rates are achievable provided the rate of transmission, is less than the capacity of the channel.

In 1993, a very powerful channel coding scheme was developed by Berrou, Galvieux and Thitmajshima, which used ideas related to both block and trellis codes. The encoding scheme uses simple convolutional codes separated by interleaving stages to produce generally low rate block codes. The decoding is done by decoding the convolutional encoder separately using soft-output Viterbi Algorithm and sharing bit reliability information in an iterative manner. This coding scheme is called Turbo Code and is found...
capable of achieving near Shannon capacity performance, the theoretical limit.
The objective of this paper is discussing the effect of decoding iterations on the performance of Turbo Codes.

2. ENCODING OF TURBO CODE:
Turbo Code is the parallel concatenation of two or more systematic codes. A generalized turbo encoder is shown in the figure below.

![Generalized turbo encoder](image)

In the figure above, a data block \( u \), which is \( k \) bits long enters the coders. The PAD block appends \( n-k \) tail bits to the data block, which yields the sequence \( x_0 \). This \( n \) bit sequence is then fed in parallel into \( M \) sets of interleavers \( \alpha \) and encoders ENC\( i \). Each interleaver scrambles the \( x_0 \) sequence in a pseudo-random fashion and feeds its output into a constituent encoder [1,2]. Each of the \( M \) constituent encoders presents a parity sequence \( x_i \) at its output. The information sequence \( x_0 \) together with the \( M \) parity sequences are concatenated to form the code word.

From the above discussion, it is clear that the Turbo Code encoder consists of two main parts:
1. A set of classical convolutional encoders.
2. A set of interleavers. The basic role of the interleaver is to spread the residual error blocks of rectangular form, which makes the decoding process stronger [1,2,3,4].

2.1 ENCODER:

The common practice of encoder is to use Recursive Systematic Convolutional (RSC) encoders. By using a convolution encoder, it is possible for the decoder to utilize a modified version of the Viterbi algorithm. Recursive encoder is used, as nonrecursive encoder will result in output codes with poor distance properties.

2.2 PUNCTURING THE OUTPUT:

It is a common practice to puncture the output of the encoder in order to increase the code rate to 1/2. For a rate 1/2 punctured turbo code, the first output stream is the input stream itself (plus the necessary padding), while the second output stream is generated by multiplexing the M non-systematic outputs of the RSC encoders.

2.3 AN EXAMPLE OF ENCODING TURBO CODE:

Because turbo codes are linear block codes, the encoding operation can be viewed as the modulo-2 matrix multiplication of an information vector with a generator matrix. Here the encoding of a sequence by Turbo Code with the help of matrix representation is demonstrated.

If the code generator matrix $g = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$, the encoder will become:
The encoder has constraint length of 3 with memory of 2 only. The encoding algorithm is similar to that of convolutional encoding. In order to generate a block code using a parallel concatenation of convolutional encoders, it is desirable for encoders to start and end in the all-zero state. To ensure this will happen, extra bits are need to "clear" the memory of encoder. Thus, the number of extra bits needed equals to the memory of encoder. In this example, 2 extra bits are needed.

For an input $u = [1 \ 0 \ 1]$, the following steps must be performed:

1. The feedback variable must be calculated first as given below:

   $$ r_k = u_k + \sum_{i=1}^{M_c} g_{1i} \times r_{k-i} \quad \text{Mod.2} \quad g_{1i} = 0,1 $$  \hspace{1cm} (1)

2. The output code word is calculated by the equation:

   $$ c_k = \sum_{i=0}^{M_c} g_{2i} \times r_{k-i} \quad \text{Mod.2} \quad g_{2i} = 0,1 $$  \hspace{1cm} (2)

3. The tail bits is calculated by:
\[ d_k = \sum_{i=1}^{M_c} g_{li} \times r_{k-i} \]  
\[ \text{Mod.2 } g_{1i} = 0,1 \]  

Where

\( M_c \): Encoder memory size.

Table I presents the complete results of encoding process.

<table>
<thead>
<tr>
<th>Input ( u_k )</th>
<th>State ( T_1 )</th>
<th>State ( T_2 )</th>
<th>Feedback ( r_k )</th>
<th>Output ( c_k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table I Encoding process

Therefore, output code \( c = [1 \ 1 \ 0 \ 1 \ 1] \)

For a pseudo-random interleaver \( \alpha \) with the mapping table as follow:

<table>
<thead>
<tr>
<th>( l )</th>
<th>( \alpha (l) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Table II Interleaver mapping

Where the interleave function \( \alpha (l) \) means the \( l \)th bit will take the \( \alpha (l) \)th bit of the original code.

The above table can also be represented by an interleaver matrix \( \alpha = [2 \ 5 \ 4 \ 1 \ 3] \). With input \( u = [1 \ 0 \ 1 \ (0 \ 1)] \), interleaver output = \( [0 \ 1 \ 0 \ 1 \ 1] \).
Then the turbo encoder becomes:

![Turbo Encoder Diagram](image)

**Fig. 3 Turbo Code encoder example**

With input $u = [1 \ 0 \ 1 \ 0 \ 1]$,

1. The first output stream $x_0$ is the input stream $u$ itself with padding and equals to $[1 \ 0 \ 1 \ 0 \ 1]$.
2. The encoder 1 output $c_1$ equals to $[1 \ 1 \ 0 \ 1 \ 1]$.
3. The encoder 2 input equals to $[0 \ 1 \ 0 \ 1 \ 1]$.
4. The encoder 2 output $c_2$ equals to $[0 \ 1 \ 1 \ 0 \ 0]$.

The turbo encoder output is therefore the multiplexing of the above three codes.

There are two different ways of multiplexing: output puncturing or not.

If output puncturing is not implemented, the output codes are simply multiplexed together by taking a bit from each stream alternatively. The resulting code becomes

- $x_0: [1 \ 0 \ 1 \ 0 \ 1]$
- $c_1: [1 \ 1 \ 0 \ 1 \ 1] \Rightarrow y: [1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0]$
If output puncturing is implemented, then the encoder outputs will multiplex into the second channel only, achieving code rate = 1/2. The code for the second channel will become the multiplexing of the encoder outputs c1 and c2 in the example:

\[ c1: [1 \ 1 \ 0 \ 1 \ 1] \]
\[ c2: [0 \ 1 \ 1 \ 0 \ 0] \Rightarrow x1: [1 \ 1 \ 0 \ 0 \ 1] \]

And the final turbo encoder output is the multiplexing of the systematic data x0 and the multiplexed encoder stream x1:

\[ x0: [1 \ 0 \ 1 \ 0 \ 1] \]
\[ x1: [1 \ 1 \ 0 \ 0 \ 1] \Rightarrow y: [1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1] \]

3. DECODING OF TURBO CODES:

It is proposed that an iterative decoding scheme should be used. The decoding algorithm is similar to Viterbi algorithm in the sense that it produces soft outputs. While the Viterbi algorithm outputs either 0 or 1 for each estimated bit, the Turbo Code decoding algorithm outputs a continuous value of each bit estimate. While the goal of the Viterbi decoder is to minimize the code word error by finding a maximum likelihood estimate of transmitted code word, the soft output decoding attempts to minimize bit error by estimating the posterior probabilities of individual bits of the code word. This decoding algorithm is called Software Decision Viterbi Decoding.

The turbo decoder consists of \( M \) elementary decoders, one for each encoder in turbo encoding part. Each elementary decoder uses the Software Decision Viterbi Decoding to produce a software decision for each received bit. After an iteration of the decoding process, every elementary decoder shares its soft decision output with the other \( M - 1 \) elementary decoders.
In theory, as the number of these iterations approaches infinity, the estimate at the output of decoder will approach the maximum a posteriori (MAP) solution.

The Turbo Code decoder can be described in the following diagram:

Assuming zero decoder delay in the turbo-decoder, the decoder 1 computes a soft-output from the systematic data \( (x_0) \), code information of encoder 1 \( (y_1) \) and a-priori information \( (L_{a2}) \). From this output, the systematic data \( (x_0) \) and a-priori information \( (L_{a2}) \) are subtracted. The result is multiplied by the scaling factor called channel reliability \( L_c \) to compensate the distortion. The result is uncorrelated with \( x_k \) and is denoted as \( L_{e1} \), for extrinsic data from decoder 1.

Decoder 2 takes as input the interleaved version of \( L_{e1} \) (the a-priori information \( L_{a1} \)), the code information of second encoder \( (y_2) \) and the interleaved version of systematic data \( (\alpha(x_0)) \). Decoder 2 generates a soft output, from which the systematic data \( (\alpha(x_0)) \) and a-priori information \( (L_{a1}) \)
was subtracted. The result is multiplied by the scaling factor called channel reliability $L_c$ to compensate the distortion. The extrinsic data from decoder 2 ($L_{e2}$) is interleaved to produce $L_{a2}$, which is fed back to decoder 1. And the iterative process continues [5,6,7].

4. THE EFFECT OF DECODING ITERATIONS ON THE PERFORMANCE OF TURBO CODE:

One distinctive characteristic of Turbo Code is iterative decoding, in which the results of one decoder will passed to the another decoder for the next decoding iterations. The intuition of iteration is that an decoder only get part of the information of the decoding bits (the first decoder gets the systematic output and also the first encoder output, while the second decoder gets the information of the systematic output and also the second encoder output, etc.)

Iteration is useful in sharing the information from one decoding to another. In this paper, the first decoder does not have the information of the second encoder output in the first iteration. After the first iteration, the output of the second decoder will feed back into the input of the first encoder. Thus first decoder have more information in the second iteration and the decoding performance should be improved [8,9].

From the above, the performance of the Turbo Code increases as the number of iterations increases. However, the time used will also increases linearly as the number of iterations. The increase in decoding time per bits will lead to increase in latency.

Therefore, designers have to justify the number of iterations to accommodate the performance/time ratio.
5. SIMULATION RESULTS:

To analyze how number of decoding iterations affects the Turbo Code; a simulation test is done with the implemented programs. A Turbo Code with frame size = 100 and with output puncturing is used. The following figure shows the performance of Turbo Code for different number of iterations:

![Frame size = 100](image)

Fig.5 BER versus decoding iterations for a block of 100 bits long

From this figure, we have the following interpretations:

1. When the decoding iterations increases from 1 to 2, the performance of Turbo Code is improved dramatically (BER is less by an order of magnitude). This is because after the information is shared between decoders, the decoders have more information about the input and thus make more accurate decision.

2. As the decoding iterations increase, the performance of Turbo Code improves. However, the rate of improvement decreases. This is
because after some iterations, the decoders already get the picture of the input code and further exchange of output does not provide as many new information as those in the first iteration. The performance line can show that there is a threshold after some iterations.

3. If the decoding iterations further increased beyond the threshold, we will found that the performance of Turbo Code will degrade. After the threshold, more iterations does not give any more information to the decoders. However, in the process of decoding, the SOVA algorithm suffers from two distortions. The first one is overoptimistic soft outputs, which is almost compensated by introducing a scale factor $L_c$. The second one is correlation between intrinsic and extrinsic information. Although the overall distortion is very small, but after a number of iterations, these distortions will accumulate, which may affect the performance of the Turbo Code.

REFERENCES:


THE EFFECT OF DECODING ITERATIONS ON THE PERFORMANCE OF TURBO CODE

(TURBO CODE) 

(CONVOLUTIONAL CODE)

(PARALLAL CONTANATION)

(RHC) 

- (dB, 3dB & 4dB2)

.1

.2