Turbo codes: from first principles to recent standards
Catherine Douillard and Michel Jezequel
Telecom Bretagne, UMR6285, Lab-STICC, F29200 Brest, France

Abstract
This chapter is a general introduction to the original turbo codes discovered in the early 1990s and also known as convolutional turbo codes or parallel concatenated convolutional codes. It presents the main concepts of coding theory introduced with the invention of turbo codes, put in an historical perspective. The overall structures of the encoder and decoder are analyzed and some fundamental guidelines for the design of turbo codes with good performance are provided. Then, the basics of turbo decoding are introduced and the main component decoding algorithms are briefly described. Finally, the very first proof-of-concept implementations are described and the pioneer telecommunication applications and current transmission standards using turbo codes are reviewed.

Keywords: Turbo codes, historical perspective, general notations and concepts, industrial impact, standards.

1. Introduction
This chapter is a general introduction to the original turbo codes, proposed and patented by Claude Berrou in 1991 [1][2]-[3] and also known as convolutional turbo codes or parallel concatenated convolutional codes. Turbo codes are an outcome of the research activity of Telecom Bretagne¹ (formerly École Nationale des Télécommunications de Bretagne), a French graduate engineering school in the field of information technologies. The Electronics Department of Telecom Bretagne has been involved in research in the field of algorithm-silicon interaction for more than 25 years. Its activity mainly consists in jointly devising new algorithms and innovative hardware architectures, digital and analog, for digital communications.

The chapter describes the main concepts of coding theory introduced with the invention of turbo codes, provides the fundamental guidelines for the design of turbo codes with good performance, gives the basics of turbo decoding and briefly reviews the industrial impacts of this new generation of error-correcting codes. Most of the sections are introduced from an historical point of view. The chapter is organized as follows. Section 2 describes the experimentations, observations and

reflections which led to turbo codes and the ensuing concepts of iterative decoding, extrinsic information and parallel concatenation. Section 3 focuses on the different constituents of the turbo encoder and analyzes their effect on the code performance. Section 4 provides the basics of the turbo principle and soft-input soft-output decoding of convolutional codes. Section 5 presents the very first hardware turbo codecs, some pioneer telecommunication applications having adopted these codes and an overall picture of the current telecommunication standards including turbo codes. Section 6 concludes the chapter.

2. History of turbo codes

The invention of turbo codes is the outcome of an intuitive experimental approach, inspired by the work of some European researchers, Gerard Battail, Joachim Hagenauer and Peter Hoeher who, at the end of the 1980s [4]-[7], highlighted the interest of soft-output decoding for concatenated coding. The present section presents a chronology describing the successive ideas that led to the development of the first turbo codes, which publication in 1993 [8] shook the coding community. With a performance at 0.5 dB from the Shannon limit, they showed a gain of almost 3 dB compared to solutions existing at that time.

2.1. The origins of turbo codes

The origins of turbo codes are to be found in the late 1980s at Telecom Bretagne. Every year, a group of third year students directed by Claude Berrou and Patrick Adde was assigned to implement a digital communication function into a CMOS logic integrated circuit. In 1989, Alain Glavieux suggested the Soft-Output Viterbi Algorithm (SOVA) proposed by G. Battail in [4] for implementation. Therefore, the beginning of the work on error-correcting codes at Telecom Bretagne was marked by essential references on Viterbi decoding such as [9] and [10] and by the two main papers describing modifications to the Viterbi decoder to make it provide soft decisions, [4] and [7]. After two years, a suitable hardware architecture was finally proposed [11]. Meanwhile, this work led the researchers to a deep understanding of probabilistic decoding. Following G. Battail, J. Hagenauer and P. Hoeher, it was observed that a soft-input and soft-output (SISO) decoder could be regarded as a signal-to-noise ratio (SNR) amplifier. This observation stimulated C. Berrou to consider techniques commonly used with electronic amplifiers, especially negative feedback. However, this analogy with amplifiers is meaningful only if the input and output of the decoder represent the same signal – that is to say, only if the code is systematic.
The next phases went faster. The development of turbo codes passed through several stages and led to the introduction of neologisms, such as parallel concatenation and extrinsic information, now part of the coding theory jargon. Here in a few words are the reflections that marked out this work, as related in [12].

2.2. Concatenation

Using the version of the SOVA in [11], it was possible to cascade SNR amplifiers and do the experiments described in [6], which was the initial plan: decoding a classical – i.e., serial – concatenation of two or more ordinary – i.e., non-systematic, non-recursive – convolutional codes. Concatenation is a simple way to obtain large asymptotic gains [13], but the performance at low SNR is debased due to the sharing of the redundancy energy between the component codes.

The concatenated coding and decoding scheme that served as a starting point to develop turbo codes is described in Figure 1.

![Diagram of Concatenated Coding Scheme](image)

*Figure 1: Serial (conventional) concatenation of two convolutional codes with coding rates 3/4 (outer code) and 2/3 (inner code). Global coding rate is 1/2.*

2.3. Negative feedback in the decoder and recursive systematic convolutional codes

Analyzing the scheme in Figure 1, one can notice a dissymmetry in the use of received information: the inner decoder benefits only from redundancy symbols $y_1$ whereas the outer decoder benefits...
from redundancy symbols $y_2$ and from the work of the inner decoder. This observation gave C. Berrou the idea of re-injecting the result of outer decoding into the inner decoder. As the different levels of the composite decoder do not represent the same pieces of information – the codes are not systematic, it was necessary to build an estimate of symbols $x_2$ and $y_2$ at the output of decoder DEC2. At first, it was a great surprise to observe that the bit error rate (BER) of these reconstructed symbols after decoding was lower than the BER of decoded information $\hat{a}$. However, an intense search did not make it possible to find any explanation for this strange behavior in the literature. It was then a straightforward task to (re)invent recursive systematic convolutional (RSC) codes in order to have information data carried by the encoder output instead of its state and take advantage of this property not covered in other works. An insight into the distance properties of these codes is given in section 3.1. The detailed analysis can be found in [14] (in French). The resulting serial concatenated coding scheme is shown in Figure 2.

![Figure 2: Serial (conventional) concatenation of two recursive systematic convolutional (RSC) codes with coding rates 3/4 (outer code) and 2/3 (inner code). Global coding rate is 1/2.](image)

The idea of re-injecting the result of outer decoding into the inner decoder, similar to the principle of the turbo engine, gave its prefix to turbo codes, although it would have been more rigorous to mention only turbo decoding, since no feedback is implemented at the encoder side.

### 2.4. Extrinsic information and iterative decoding

The soft-output Viterbi decoder of a systematic code provides a good estimate of the log likelihood ratio (LLR) relative to its input symbols. It can be shown that each computed LLR can be expressed as the sum of two contributions. The first, intrinsic information, is available at the channel output before any decoding stage; the second, extrinsic information, is provided by exploiting the dependencies (due to convolution, parity, ...) existing between the symbol being processed and the other symbols processed by the decoder. As the intrinsic information is used by both decoders (at
different instants), it is the extrinsic information produced by each of the decoders that must be passed to the other as new information, to ensure joint convergence. In any decoding construction, extrinsic information must not be used by the processor which produced it. C. Berrou finally came up with the scheme depicted in Figure 3. The decoder is built in such a way that no information produced by a component decoder can feed its input, either directly or indirectly.

![Figure 3: Structure of the very first turbo decoding scheme](image-url)

Others, mainly in the United States, Peter Elias [15], Robert Gallager [16][17], Michael Tanner [18], etc. had earlier imagined procedures for coding and decoding that were the forerunners of turbo codes.

The digital processing of information has at least one major drawback: it does not lend itself easily to feedback techniques. An iterative procedure has to be used because of the delays (trellis, interleaving, ...). This procedure increases the decoder latency, but ever-continuing progress in microelectronics makes possible today what was unimaginable not so long ago.

Two hardware solutions can be contemplated, depending on the information rate. For low data rates, a single decoding processor working at a high frequency can make all the required iterations with a tolerable added delay. For high rates, a cascade of identical decoding modules can be implemented monolithically to enable high-speed pipe-line processing (in this case, typically that of broadcasting, the latency problems are generally less crucial).

### 2.5. Parallel concatenation

The idea of the so-called parallel concatenation paradigm did not germinate by analogy with product codes, as sometimes reported. It came actually from the team in charge of the design of the very first pipe-lined turbo decoding integrated circuit, intended to validate the concept of iterative decoding.
Serial concatenation requires different clock signals for the inner and outer decoders. Due to the symmetric structure of the encoder, parallel concatenation simplifies the architecture of the system since both component encoders and decoders can work with the same clock signal, which is also the data clock. The first studied parallel turbo encoder and its decoding scheme are shown in Figure 4. Turbo codes are sometimes also called parallel concatenated convolutional codes (PCCC) in contrast with the conventional serial concatenation.

Figure 4: Parallel concatenation of two RSC codes and associated (asymmetrical) decoder

Later on, a symmetrical structure was also devised for the turbo decoder (Figure 5), which is more natural in the sense that it reflects the symmetry of the encoding process.
It was also observed that, for a given coding rate, parallel concatenation yields more redundant symbols from the outer code than serial concatenation does. A parallel concatenation of two elementary codes $C_1$ and $C_2$ with coding rates $R_1$ and $R_2$ has a global coding rate:

$$R_p = \frac{R_1 R_2}{R_1 + R_2 - R_1 R_2} = \frac{R_1 R_2}{1 - (1 - R_1)(1 - R_2)}$$

(1)

whereas the global coding rate of the serially concatenated code is $R_s = R_1 R_2$. For instance, a global coding rate 1/2 can be obtained with the parallel concatenation of two codes with elementary rates 2/3 or with the serial concatenation of two codes with elementary rates 2/3 and 3/4, as in Figure 1. Thanks to a greater number of redundant symbols, the parallel structure can benefit from a higher diversity effect. This explains why the convergence threshold, *i.e.* the minimum SNR at which the iterative decoder starts to correct most of the errors, is lower when the concatenation is parallel. In return, serially concatenated convolutional codes (SCCC) show lower changes of slope in the bit error probability curves than their parallel counterpart, due to higher minimum Hamming distances. The distance properties of PCCCs and SCCCs are analyzed in [19] and [20].
3. Fundamentals of turbo coding

The turbo encoder involves a parallel concatenation of at least two elementary Recursive Systematic Convolutional (RSC) codes separated by an interleaver ($\Pi$).

Figure 6: The turbo encoder structure, a parallel concatenation of two RSC encoders separated by an interleaver.

Figure 6 represents a turbo code in its most conventional version [8][21]. The input binary message of length $K$ is encoded in its natural order and in a permuted – or interleaved – order by two RSC encoders ENC1 and ENC2. In the figure, the RSC encoders are identical but this property is not essential. The overall coding rate $R$ of the turbo code is $1/3$. To obtain higher rates, the most common technique involves puncturing, i.e. not transmitting, part of coded symbols, most often redundancy symbols $Y_1$ and $Y_2$. Adopting $m$-binary codes is another way of increasing the code rate [22].

3.1. Recursive Systematic Convolutional (RSC) component codes

The convolutional codes used in the experiments described in Figure 1 was a non-systematic code. However, due to the observation mentioned in section 2.3 that the BER of the reconstructed symbols after decoding was lower than the BER of decoded information, C. Berrou, A. Glavieux and their PhD student Punya Thitimajshima investigated the ways of building systematic convolutional codes with
good performance [14]. It was then observed that recursive systematic codes could offer good performance both in low and high SNR regions. Let us illustrate this statement with the memory-2 systematic and non-systematic codes given in Figure 7.

![Figure 7: An example of memory-2 convolutional codes with their trellis](image)

(a) non-systematic convolutional (NSC) code, polynomials \([1+D^2, 1+D+D^2]\)

(b) systematic convolutional (SC) code, polynomials \([1, 1+D+D^2]\).

The free distance of the NSC code is equal to 5, yielding better asymptotic performance than the SC code whose free distance is equal to 4. This is illustrated by Figure 8, which compares the BER of both codes as a function of \(E_b/N_0\), where \(E_b\) denotes the energy received per information bit and \(N_0\) is the noise power spectral density.
Figure 8: Comparison of simulated performance of the memory-2 NSC and SC codes.

However, one can observe that, for low values of $E_b/N_0$, the systematic code performs better than the non-systematic code. This behavior can be predicted by the observation of the distance spectrum of these codes which is given in Table 1. To compare the performance of these codes at low SNRs, all the terms of the distance spectrum have to be considered. Thus, the better performance of the systematic code can be explained by the lower “density” of its spectrum for $d > 10$.

<table>
<thead>
<tr>
<th>$d$</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n(d)$</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>89</td>
<td>0</td>
<td>233</td>
</tr>
<tr>
<td>$n(d)$</td>
<td>3</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>58</td>
<td>0</td>
<td>201</td>
<td>0</td>
<td>655</td>
<td>0</td>
<td>2,052</td>
</tr>
<tr>
<td>$n(d)$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>$n(d)$</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>12</td>
<td>32</td>
<td>80</td>
<td>192</td>
<td>448</td>
<td>1,024</td>
<td>2,304</td>
<td>5,120</td>
</tr>
</tbody>
</table>

Table 1: First terms of the distance spectrum of the NSC and SC codes of Figure 7. $n(d)$ represents the number of sequences with Hamming weight $d$ and $w(d)$ is the cumulative input weight of these sequences.

To build parallel concatenated codes, C. Berrou was seeking component codes that were systematic, due to their good behavior at low SNRs, but also with free distances as high as the conventional
(NSC) codes. They found such codes with the rediscovery of recursive convolutional codes in their systematic version.

Introducing recursivity into the NSC code of Figure 7 (a) can provide two recursive systematic convolutional (RSC) codes with transfer functions \(1, \frac{1+D+D^2}{1+D^2}\) and \(1, \frac{1+D^2}{1+D+D^2}\). Let us consider the first one, represented in Figure 9 with the corresponding trellis.

![Figure 9: Recursive systematic convolutional (RSC) code with transfer function \(1, \frac{1+D^2}{1+D+D^2}\).](image)

The free distance of this code is identical to the free distance of the NSC mother code, since the trellis labels are identical. The difference with the trellis of Figure 7 (a) lies in the value of the input associated with the encoder state transitions (red transitions in the figure). Consequently, the distance spectrum of the RSC code only differs from the spectrum of the NSC code in the cumulative weight values. The first terms of the distance spectrum of this code are given in Table 2. As expected, the cumulative weights are lower than those of the NSC codes for \(d \geq 9\), thus yielding better performance at low SNRs. Examining the first term of Table 2, one can notice that the input weight of sequence with minimum Hamming weight \(d = 5\) is 2 for the RSC code instead of 1 for the NSC code. Consequently, for high SNRs the probability of error of the RSC code is twice higher than that of the NSC code, which is a negligible degradation in practice. Figure 10 compares the BER curves of the RSC code of Figure 9 and the SC and NSC codes of Figure 7.

<table>
<thead>
<tr>
<th>(d)</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n(d)) RSC</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>(w(d)) RSC</td>
<td>2</td>
<td>16</td>
<td>14</td>
<td>32</td>
<td>72</td>
<td>160</td>
<td>352</td>
<td>768</td>
<td>1,664</td>
<td>3,584</td>
</tr>
</tbody>
</table>

Table 2: First terms of the distance spectrum of the RSC code of Figure 9.
Figure 10: Comparison of simulated performance of the memory-2 RSC code with NSC and SC codes.

Furthermore, when rates higher than the natural code rate are required, it has been observed that it is possible to find punctured RSC codes whose performance, in terms of probability of error, is always better than the non-recursive form of the code [14]. Figure 11 compares the BER curves of the RSC code of Figure 9 and the NSC code of Figure 7 when both codes are punctured to achieve a coding rate of 3/4.
In general, periodic puncturing patterns performed on the redundancy symbols yield the best performance for the resulting turbo code. However, puncturing systematic symbols can also be considered for high coding rates, in order to increase the minimum Hamming distance of the code. In this case, the convergence threshold can be slightly degraded (increased) since puncturing information data common to both component codes penalizes both SISO decoders.

In practice, the value of the component RSC code memory is taken lower than or equal to 4. Increasing the component code memory increases the minimum Hamming distance of the resulting turbo code but also increases decoding complexity, as it is proportional to the number of states in the trellis. The generator polynomials are generally those previously used for conventional convolutional codes that can be found in the extensive literature on channel coding in the 1980s and 1990s.

Recursive convolutional codes have been scarcely discussed in the literature before the invention of turbo codes since they were considered by coding specialists not to have particular advantages compared to conventional non-systematic (non-recursive) codes. This is indeed true when high SNR regions are considered. However, for coding schemes approaching channel capacity, performance at low SNR becomes crucial. This is the reason why RSC codes came back to prominence with the invention of turbo codes. Later, another essential property of RSCs was observed that makes them necessary to build PCCCs with high minimum Hamming distances.
A turbo code with component encoders ENC1 and ENC2 that encodes a weight-$w$ information sequence provides a weight-$d$ coded sequence with

$$d = w + p_1 + p_2$$  \hspace{1cm} (2)

where $p_1$ and $p_2$ are the weights of the parity sequences provided by ENC1 and ENC2 respectively.

If $w = 1$, a non-recursive encoder produces a finite-weight parity sequence whereas a recursive encoder produces an infinite-weight parity sequence. Consequently, if ENC1 and ENC2 are non-recursive encoders, a weight-1 input sequence results in two low-weight parity sequences and thus in a low-weight overall codeword. A turbo code built with non-recursive component codes has a very low minimum Hamming distance, *whatever the permutation II.*

If ENC1 and ENC2 are RSC encoders, the minimum Hamming distance of the turbo code is not limited by weight-1 input sequences, since these sequences result in infinite-weight codewords. Low weight parity sequences can only result from information sequences with $w \geq 2$. For such sequences, the permutation can help to increase the concatenated codeword weight. This property was demonstrated by Benedetto and Montorsi in [19] using the so-called *uniform interleaving* approach [23]. The uniform interleaver is a theoretical device that yields average performance concatenated codes. In practice, uniform interleavers can be simulated by successively generating a random permutation for each encoded sequence. Under the uniform interleaving assumption, it was shown in [19] that, for a PCCC, the probability of error after decoding is proportional to $K^{1-w_{\text{min}}}$, where $K$ is the interleaver size and $w_{\text{min}}$ is the minimum number of information bits in a finite-weight non-zero encoded sequence. For non-recursive component codes, $w_{\text{min}} = 1$ and no *interleaving gain* can be obtained whereas for recursive component codes, $w_{\text{min}} = 2$, and the interleaving gain goes as $1/K$.

**3.2. Block coding with turbo codes**

Convolutional codes are naturally adapted to the encoding of very long data sequences, for broadcasting applications for instance. Therefore, the very first turbo decoders were designed to encode and decode continuous data sequences (see section 5.1). However, most telecommunication systems use independent block transmissions, with very short lengths – a few dozen of bits – in some cases. In the decoding process of convolutional codes (see section 4.2), the decoder needs to know the initial and final state of the encoder during the decoding of the block. Since the convolutional encoder structure calls for a shift register, the initial state can be easily fixed by resetting the register.

---

2 In the event of an infinite-length information sequence.
Knowing the final state is not so easy since its value depends on the encoded message. The various techniques allowing this problem to be solved are called \textit{trellis termination} techniques. For turbo codes, the trellises of both component encoders have to be terminated. Several solutions can be considered:

\textbf{Direct truncation}: The most obvious possibility is to stop the encoding process when all the information data has been applied to the encoder input. Thus, the final state is unknown at the decoder side, both in the natural and in the interleaved order and the overall error correction performance of the code is degraded due to a greater density of decoding errors at the block ends. This degradation is stronger for short blocks than for long blocks and could be admissible is some applications. It should be noted that the direct truncation of the trellises has a more negative impact on the block/packet error rate (PER or BLER) than on the BER.

\textbf{Zero termination}: The standard solution to avoid this performance degradation at the end of the data block consists in adding \( v \) additional bits or \textit{tail bits} to the original message so that the encoder retrieves the zero state, \( v \) being the code memory. In the case of non-recursive convolutional codes, injecting \( v \) zero bits at the end of the message makes the encoder to be forced to zero. The problem is slightly more difficult in the case of a recursive code. Nevertheless, the trellis termination can be achieved by injecting zero bits at the input of the shift register, as shown in Figure 12 for the RSC encoder of Figure 9. For turbo codes, the trellis of one or both component codes can be terminated using tail bits. The CCSDS [24], UMTS [25][26] and LTE [27] standards have adopted this technique. A first drawback when used by turbo code is that the tail bits used to terminate the trellis of one component code are not encoded by the other component code: in other words, they are not turbo coded. They are therefore less protected than the regular data bits. This leads, but to a lesser degree, to similar drawbacks as direct truncation. Moreover, the tail bits have to be transmitted. This causes a decrease in the coding rate and in the spectral efficiency, which is fortunately not significant, except for very short information block lengths.
Figure 12: Termination of the RSC encoder of Figure 9. Regular encoding is performed when the switch is in position 1. Switching to position 2 makes the encoder retrieve state zero after \( v = 2 \) encoding steps.

**Automatic trellis termination:** The turbo code interleaver can be designed so that the encoder always retrieves state zero, provided that self-concatenation is applied: the interleaved data sequence has to be directly encoded behind the non-interleaved sequence, without initializing the encoder state in between. This property is shown in [28]. Unlike the zero termination technique, this method does not introduce any side effect and does not require the transmission of additional bits. Unfortunately, it imposes strong constraints on the permutation design that turned out to be hardly compatible with high minimum Hamming distances and low error rates.

**Tail-biting:** This technique was introduced in the eighties to solve the problem of trellis termination without introducing any side effect [29]-[31]. It allows any state of the encoder as the initial state and the information sequence is encoded so that the final state and the initial state are identical. Among the different techniques aiming at transforming a convolutional code into a block code, tail-biting, also called *circular encoding*, is the best termination method for turbo codes. First, no extra bits have to be added and transmitted; thus, there is no rate loss and the spectral efficiency of the transmission is not reduced. Next, tail-biting does not induce any side effect in the message. Consequently, all the information bits are protected in the same way by the turbo code and the circular property prevents the occurrence of low-weight truncated codewords since the trellis can be considered as with infinite length. Moreover, since no peculiar positions in the trellis have to be considered, the permutation design is made simpler.

As for RSC codes, circular encoding requires a two-step encoding process [32], chap. 5, [30], [31], illustrated by Figure 13 in the case of the \((1, 1+D2+D3/1+D+D3)\) RSC code shown in Figure 13(a):

- The first step determines the initial/final state (sometimes called *circulation state* \( S_c \)): the information message is encoded from initial state zero as shown in Figure 13(b). The resulting final state \( S_K \) allows the value of the circulation state to be calculated from Table 3.
- The second step is the actual encoding stage. The encoder starts in the correct initial state \( S_c \) as illustrated in Figure 13(c) and the corresponding valid codeword is delivered.
Figure 13: Circular encoding of message “01101101” with the $(1, 1+D^2+D^3/1+D+D^3)$ RSC code: (a) $(1, 1+D^2+D^3/1+D+D^3)$ RSC encoder; (b) First encoding step: the message is encoded from state 0. The final state is $S_k = 3$. (c) Actual encoding step: the message is encoded from circulation state $S_c = 2$ and the final state is also $S_c = 2$. 

<table>
<thead>
<tr>
<th>$S_k$ mod 7</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>
Table 3: Table providing the circulation states of the $\{1, 1+D^2+D^3/1+D+D^3\}$ RSC code as a function of $K \mod 7$ ($K$ is the message length) and of the final state obtained from the first encoding step. In the example of Figure 13, $K \mod 7 = 1$, $S_a = 3$ and $S_c = 2$.

The contents of the circulation state table depends on the code memory and on the recursion polynomial. The computation principle is described in [32], chap. 5, [30] and [31]. Note that for some block sizes, the circulation state does not exist: in the example described in Figure 13, the circulation exists if and only if $K$ is nor a multiple of 7.

When tail-biting is adopted, the code trellis can be viewed as a circle (see Figure 14): the iterative decoding of such codes involves repeated and continuous loops around the circular trellis. The number of loops performed is equal to the required number of iterations. The state probabilities or metrics, according to the chosen decoding algorithm, computed at the end of each loop are used as initial values for the next loop.
This elegant and efficient method of transforming a convolutional code into a block code has a disadvantage compared to the other termination techniques: the two-step encoding stage introduces latency. However this is not a major handicap since, due to the very simple structure of the encoder, encoding can be performed at a frequency much higher than the data rate. Due to its special convenience for turbo codes, this termination technique has been adopted in several recent telecommunication standards such as DVB-RCS [33][34], DVB-RCT [35] and WiMAX [36].

3.3. The permutation

Whatever we call it interleaving or permutation, the technique that involves spreading the data over time has always been very useful in digital communications. The initial and basic idea for introducing an interleaver in the parallel concatenated structure of turbo codes was to efficiently fight against the occurrence or error burst, on at least one of the dimensions of the composite code. However, in the very first developments related in section 2, the results of the first simulations carried out with parallel concatenation were encouraging and frustrating at the same time; encouraging because exceptional performance was obtained at low SNR; frustrating, because the BER could hardly fall below $10^{-4}$. This was due to the observation of the – now usual – shape of the bit error rate (BER) curve of a concatenated code under iterative decoding, shown in Figure 15, divided into two regions: the waterfall region, which appears at low signal-to-noise ratios (SNR), has a steep slope, especially for long blocks of information bits. The error-floor region, which appears at higher SNRs has a flatter slope caused by low-weight codewords.
The very first turbo code simulations called for a row-column regular interleaver: data are written row-wise in a rectangular matrix while they are encoded with ENC1 (natural order) and are read column-wise before being encoded by ENC2 (interleaved order). It was then observed that most of the residual error patterns the turbo decoder could not correct were also regular, at the four corners of a rectangle for instance. It was then realized that the way the permutation \( \Pi \) was defined, in close relation with the properties of the component codes, had an impact on the minimum Hamming distance of the concatenated code and governed its behavior at low BER (< \( 10^{-5} \)).

A permutation with good scattering properties avoids that two bits which are close to each other before being interleaved (i.e. in the natural order) remain close after interleaving. These undesirable configurations cause a performance degradation because they introduce some correlation in the iterative decoding process [37][38]. The scattering properties of a permutation can be measured through its minimum spread, defined as [39][40]:

\[
S_{\min} = \min_{j_1, j_2 | j_1 \neq j_2} (|j_1 - j_2|_K + |\Pi(j_1) - \Pi(j_2)|_K), \text{ for } j_1, j_2 = 1 \ldots K
\]  

(3)

where \( K \) is the interleaver length and \( \Pi \) denotes the permutation function that associates index \( \Pi(i) \) in the interleaved order to an index \( i \) in the natural order. If the code is tail-biting, \( |i - j|_K = \min(|i - j|, K - |i - j|) \), otherwise \( |i - j|_K = |i - j| \).

This parameter measures the minimum cumulated spatial distance between two bits before and after interleaving. The maximum achievable value for \( S_{\min} \) is upper bounded by \( \sqrt{2K} \) [39][40].

\[20\]
Besides introducing correlation in the decoding process, permutations with a low minimum spread may lead to low weight codewords that can limit the minimum Hamming distance of the turbo code. In order to obtain turbo codes with large minimum Hamming distances, the trick is to match low-weight codewords before permutation with high-weight codewords after permutation, and vice versa.

Another important factor that has to be carefully considered for the design of the turbo code interleaver is its implementation, especially when the encoding of long data blocks is required. There are mainly two ways to specify and implement a permutation: describe the link between the addresses before and after permutation with equations or store the correspondence between addresses in a look-up table. The first one is easier to specify and implement but the second can lead to higher minimum Hamming distances, since the permutation search area is generally larger.

### 3.3.1. Regular permutation

The conventional implementation of regular permutation involves writing the data in a memory row-wise while encoding them with encoder ENC1 and reading them column-wise for the second encoding stage with ENC2. This permutation process assumes that the block of $K$ data to be encoded can be organized as table of $N_r$ rows and $N_c$ columns, such as $k = N_r \times N_c$ (see Figure 16).

![Figure 16: Regular permutation representation in rectangular form.](image)

When tail-biting is adopted in the encoding and decoding process, a circular representation of the data block is more appropriate than the rectangular representation.
Then, another form of regular permutation calls for circular shifting [41] or the co-prime [42] principle. It consists in writing the data in a linear memory from address $i = 0$ to address $i = N - 1$ and to read them at addresses

$$i = \Pi(j) = Pj \mod K \text{ for } j = 0 \ldots K - 1$$

(4)

In this form, shown in Figure 17, the data block is viewed as a circle, the two extremities being adjacent in both natural order ($i = 0$ and $i = K - 1$) and interleaved order ($j = 0$ and $j = K - 1$). This makes the circular representation well suited for tail-biting convolutional codes. The constraint design for the circular permutation is that $P$ and $K$ have to be relatively prime.

Regular permutation is able to achieve the maximum value for the minimum spread $S_{\text{min}, \sqrt{2K}}$ [40]. It is therefore appropriate for turbo codes in the sense that they minimize the correlation between the extrinsic information provided by each decoder.

According to (2), determining the minimum Hamming distance of a turbo code involves identifying the low-weight input sequences that produce low-weight parity sequences. These sequences are those which make the component encoders leave the all-zero state and then to return to the all-zero state. They are sometimes called return to zero (RTZ) sequences [32]. Since component codes are considered are RSC codes, only input sequences with weight greater than or equal to two have to be considered.

Two cases have to be considered for regular permutation:
- Input sequences only contain one return to zero (simple RTZ sequence),
- Input sequences contain several returns to zero (composite RTZ sequence).

For the turbo code of Figure 18(a) and adopting the rectangular form of the regular permutation, the first type of RTZ sequences is illustrated by Figure 18(b) and (c) whereas the second type is illustrated by Figure 18(d) and (e).

Let us analyze the behavior of the turbo encoder in the presence of such input RTZ sequences.

When the pattern described in Figure 18(b) is encoded row-wise by encoder ENC1, this encoder receives a length-8 RTZ pattern, whose corresponding redundancy pattern is “11001111”, providing a parity weight \( p_1 = 6 \). When the interleaved sequence is encoded column-wise by ENC2, the pattern delimited by the two ones is again an RTZ pattern but its length is \( 7N_r + 1 \). If \( N_r \) is large enough, the weight of the parity sequence produced by ENC2, \( p_2 \), is much larger than \( p_1 \), thus yielding a high total weight \( d \) for the encoded sequence. A similar observation can be made for the weight-3 input sequence of Figure 18(c): the short RTZ pattern for ENC1 provides redundancy pattern “1011” and \( p_1 = 3 \) but the corresponding RTZ pattern is much longer for ENC2. The same argument can be given for simple sequences with such short RTZ patterns for ENC2. Simple RTZ sequences provide low redundancy weight on one encoding dimension and higher redundancy weight on the other – provided the values of \( N_r \) or \( N_c \) are large enough. In general, when \( K \) tends towards infinity via the values of \( N_r \) or \( N_c \), the total weight of such simple RTZ sequences also tends towards infinity. **One can conclude that the regular permutation is appropriate to simple RTZ sequences.**
The RTZ sequence presented in Figure 18(d) consists of two length-8 RTZ patterns for both encoders ENC1 and ENC2. Row-wise encoding produces two redundancy patterns equal to “11001111”, yielding parity weight \( p_1 = 2 \times 6 = 12 \). The same holds for column-wise encoding with ENC2 and \( p_2 = p_1 = 12 \). The resulting total weight of this sequence is then \( d = w + p_1 + p_2 = 6 + 12 + 12 = 30 \). For the input-9 triple RTZ sequence of Figure 18(e), the parity weight is equal to \( 3 \times 3 = 9 \) on each encoding dimension, thus resulting in a total weight \( d = w + p_1 + p_2 = 9 + 9 + 9 = 27 \).

As early observed, the minimum distances associated with these patterns are generally not sufficient to ensure good performance at low error rate, especially for long information blocks. For instance, Figure 19 shows that a minimum Hamming distance of at least 50 is necessary to achieve a frame error rate of \( 10^{-7} \) in a Gaussian channel for an information block of 1000 bits when the code is not punctured (coding rate 1/3).

Furthermore, the distances associated to the composite RTZ sequences do not depend on the block size and cannot be increased by increasing the value of \( K \) via \( N_r \) or \( N_c \). The regular permutation is therefore definitely not appropriate for the composite RTZ sequences.
3.3.2. Irregular permutations

In the light of the performance of turbo codes with regular permutation, the idea emerged that some disorder had to be instilled into the permutation in order to break the regularity of the composite error patterns, while keeping high distances for simple RTZ sequences and a reasonably high spread for avoiding problems of correlation in the decoding process. But the disorder must be carefully managed! Numerous non-regular turbo code interleavers have been imagined so far that have generated numerous publications, see for instance [42]-[57].

We first present in this chapter two families of interleavers that are both efficient and simple to implement. The basic common idea behind these permutations is that some disorder has to be instilled into the permutation in order to break the regularity of the composite error patterns, while keeping high distances for simple RTZ sequences and a reasonably high spread for avoiding problems of correlation in the decoding process. They can be applied to the rectangular form of the permutation or to its circular form.

The first one is called dithered relatively prime (DRP) permutation [44][49][51]. Two levels of controlled disorder are introduced before and after the regular permutation, in groups of $W$ and $R$ bits respectively. In practice $W$ and $R$ can be identical values, typically 4 or 8. This way of introducing disorder locally does not significantly decrease the minimum spread value of the interleaver. However, it allows some rectangular error patterns to be removed, provided that the dimensions of the rectangle are multiples of $W$ and $R$.

Figure 20: Dithered relatively prime (DRP) permutation (example taken from [49])
Another permutation concept close to DRP, called almost regular permutation (ARP) was developed independently by C. Berrou’s team [50]. The disorder is introduced by means of a set of shift values that are applied when reading the data in the interleaved order. Considering circular permutation, the ARP model is an extension of (4):

\[ i = \Pi(j) = Pj + Q(j) \mod K \quad \text{for } j = 0 \ldots K - 1 \]  

(5)

where \( Q(j) \) is an integer. The number of different shift values \( Q(j) \) used in the ARP is called the disorder cycle and is denoted \( C \). \( C \) must be a divider of \( K \). DVB-RCS [33], DVB-RCT [35] and DVB-RCS2 [34] standards have adopted the ARP model with \( C = 4 \) for turbo code interleaving.

For example, in DVB-RCS/RCT the four values for \( Q(j) \) are:

- If \( j \mod 4 = 0 \) then \( Q(j) = 1 \)
- If \( j \mod 4 = 1 \) then \( Q(j) = K/2 + P_1 + 1 \)  
(6)
- If \( j \mod 4 = 2 \) then \( Q(j) = P_2 + 1 \)
- If \( j \mod 4 = 3 \) then \( Q(j) = K/2 + P_3 + 1 \)

Where the values of \( P_1, P_2 \) and \( P_3 \) depend on the block size, that is on \( K \).

In these standards, the block sizes under consideration are rather small, ranging from a few dozen bits to a few thousands bits. Therefore, a disorder cycle equal to 4 is sufficient to ensure good performance. For longer block lengths, up to 10,000 or 100,000, larger values of \( C \), up to 8 or 16, may be necessary in order to achieve acceptable minimum Hamming distances.

Figure 21 shows some PER performance curves obtained with the DVB-RCS [33] (red curves) and DVB-RCS2 [34] (blue curves) turbo codes that both use an ARP interleaver. The simulation results take actual implementation constraints (input quantization and simplified decoding algorithm) into account. We can observe good average performance for the memory-3 DVB-RCS code, whose decoding complexity is very reasonable. The performance improves predictably with block size and coding rate in relation to the theoretical limit. The reported limits on PER are derived from the Gallager’s random coding bound on the error probability for binary-input channels, as described in [52]. To improve the performance of this code family at PER below \( 10^{-4} \), the more powerful but more complex memory-4 component code has to be selected in order to increase the overall minimum Hamming distance of the concatenated code. Then the simulated curves lie within less than 1 dB from the limit, regardless of block size and coding rate.
Figure 21: PER performance of double-binary memory-3 and memory-4 turbo codes with ARP interleaving of size 1504 for coding rates 1/2, 2/3 and 3/4. Max-Log MAP decoding algorithm (see section 4.2.3) with 4-bit input samples and 8 iterations. QPSK modulation and Gaussian channel.

A major advantage of the ARP model is that it naturally offers a parallelism degree of $C$ in the decoding process, for both the natural and interleaved order and can therefore suitable for high-speed hardware implementations of turbo decoders. According to the technique depicted in Figure 22 for $C = 4$, four processors are assigned to the four quadrants of the circle under process. At the same time, the four units deal with data that are located at addresses $i$ corresponding to the four possible values of $i \mod C$. For instance, at the beginning of the process, the first processor deals with data located at an address $i$ with congruence 0 (i.e. such as $i \mod C = 0$), the second with data at an address with congruence 1, and so on. The next decoding step, the first processor handles data at an address with congruence 2, the second process handles data at an address with congruence 3, etc. Finally, a parallelism with degree 4 is easily feasible by means of a barrel shifter directing the four processors towards four distinct memory pages. For any value of $C$, larger parallelism degree $pC$ is possible, provided that $K$ is a multiple of $pC$. Later on, the ARP interleaver was also shown to be contention-free for many other values of parallelism degrees [53].
Another class of deterministic irregular interleaver for turbo codes is worth mentioning, due to its adoption in 3GPP Long Term Evolution (LTE) [27] and in its further evolution LTE-advanced (beyond release 9 of LTE). This interleaver is based on quadratic permutation polynomials (QPP) over integer rings [54]-[57]. The design of such interleavers is reduced to the selection of polynomial coefficients. Moreover, QPP interleavers have been shown to have high minimum Hamming distances [56] and spread [57]. Besides, QPP interleavers can be designed in order to have the contention-free property for all window sizes dividing the interleaver length, thus allowing a high degree of freedom for parallel processing [55].

4. Fundamentals of turbo decoding

4.1. The turbo principle

Conventional decoding of turbo codes involves two constituent SISO decoders that exchange extrinsic information through an iterative process. Originally, the first iterative decoders for turbo codes were based on an asymmetrical structure, where both component decoders had slightly different roles as related in section 2 and shown in Figure 4. The successive operations carried out by this turbo decoder are described in Figure 23. Each SISO decoder computes the LLR of information data $d$, in natural order for SISO₁ and in interleaved order for SISO₂. At the first iteration (see Figure 23 (a)), the SISO decoders are simply concatenated and extrinsic information is obtained by subtracting the systematic input of SISO₂ to its output. At iteration $I_t$ (see Figure 23 (b)), extrinsic information computed at iteration $I_{t-1}$, $z^{I_{t-1}}$ is added to the systematic channel data after having been
properly de-interleaved. The decoding result is then passed to SISO\(_2\) in the interleaved order after having subtracted input extrinsic information \(z^{\text{It}-1}\), in order to prevent SISO\(_2\) from reusing a piece of information provided by itself. This process makes both SISO decoders benefit from the redundancy provided by both codes.

Figure 23: Turbo decoding principle; (a) Processing first iteration; (b) Processing iteration \(\text{It}\).

In contrast to the turbo encoding process, this turbo decoder structure is not quite symmetrical with respect to the SISO decoders. Therefore, later on a symmetrical structure was also devised for the turbo decoder, which is more natural as it better reflects the structure of the encoding process. Figure 24 shows the operations carried out by the symmetrical form of the turbo decoder at the first iteration (see Figure 24 (a)) and at iteration \(\text{It}\) (see Figure 24 (b)). The main difference with the serial structure is that systematic data \(x\) is added before and subtracted after each SISO decoding process, thus providing extrinsic information after each SISO decoder or, in other words, after each half
iteration. In this structure, both SISO decoders provide extrinsic information whereas in Figure 23, only the second SISO decoder does. These are two somewhat different points of view for the same actual behavior.

Figure 24: Another view of turbo decoding principle; (a) Processing first iteration; (b) Processing iteration $l_t$. 
From the decoding structure of Figure 24, the LLR computed by the SISO decoders at the first iteration can be expressed as

\[
\begin{align*}
\text{LLR}_1^1(d) &= x_1 + z_1^1 \\
\text{LLR}_2^1(d) &= (x_2 + \Pi[z_1^1]) + z_2^1
\end{align*}
\]

(7)

At iteration \( I_t \), these expressions become

\[
\begin{align*}
\text{LLR}_1^{I_t}(d) &= (x_1 + \Pi^{-1}[z_2^{I_t-1}]) + z_1^{I_t} \\
\text{LLR}_2^{I_t}(d) &= (x_2 + \Pi[z_1^{I_t}]) + z_2^{I_t}
\end{align*}
\]

(8)

where \( \Pi \) and \( \Pi^{-1} \) refer to the interleaving and de-interleaving functions. The next subsection presents an insight in the SISO decoding algorithms that justifies these equations and the overall decoding principle.

Equations (7) and (8) assume that each decoding iteration starts with SISO\(_1\) and ends with SISO\(_2\). Due to the symmetry property, the role of the SISO decoders can be swapped, starting the first half-iteration with SISO\(_2\) and processing the second half-iteration with SISO\(_1\), without any difference in the final result. Some parallel turbo decoding scheduling can also be considered, where SISO\(_1\) and SISO\(_2\) process data simultaneously and continuously exchange extrinsic information (see for instance [58][59]).

When the iterative process converges towards a stable solution, \( z_1^{I_t} - z_1^{I_t-1} \) and \( z_2^{I_t} - z_2^{I_t-1} \) tends towards zero when \( I_t \) tends towards infinity. Thus, with every passing iteration, both SISO decoders merge towards the same probabilistic decision. Figure 25 shows the BER performance curves of the double-binary memory-3 turbo code (derived from the DVB-RCS code [33]) as a function of the number of iterations. This code approaches within about 0.35 dB the theoretical limit introduced by Shannon for a BER of \( 10^{-5} \). One can conjecture that most of the residual loss is due to the use of repeated elementary decoding stages instead of a global one-shot decoding.

This result is obtained with a large number of iterations, a floating point MAP decoder (see section 4.2.2) and a large interleaver. For such long blocks (greater than 10,000 information symbols), the performance gain no longer improves beyond 20 iterations. For shorter blocks, the required number of iterations tends to be lower: practical turbo decoders usually implement from 6 to 10 iterations, depending on the block size and the SISO decoding algorithm used.
Figure 25: BER performance of a rate 1/2 double-binary memory-3 turbo code with pseudo-random interleaving of size 20,000. MAP decoding algorithm, with 1, 5, 10, 15 and 20 iterations. Floating point input data. BPSK modulation and Gaussian channel.

The proof of convergence of turbo decoders is not a trivial matter. Some tentative explanations can be found in [60] and [61] for instance. Nevertheless, there exist useful tools dedicated to the analysis of the iterative process convergence. The most popular one is the extrinsic information transfer (EXIT) chart introduced by Stephan ten Brink [62][63]. The flow of extrinsic information through the constituent SISO decoders is analyzed from a mutual information point of view. The extrinsic information transfer characteristic of each SISO decoder is plotted and the exchange of extrinsic information can be visualized through a decoding trajectory. This tool is particularly useful to predict the position of the waterfall region (see Figure 15 in section 3.3) for a given turbo code structure.

4.2. Soft-Input Soft-Output decoding

In the early nineties, Viterbi decoding architectures were widely known and recent publications related to a modified Viterbi algorithm delivering the \textit{a posteriori} probability or a reliability value for each bit [4][7] directed the very first hardware implementations of turbo decoders [64][65] towards a simplified version of the soft Viterbi algorithm (SOVA) [11]. At the same time, A. Glavieux advocated the use of the Ball-Cocke-Jelinek-Raviv (BCJR) algorithm [66], also known as symbol-by-
symbol maximum *a posteriori* (MAP) algorithm. This algorithm is optimal for estimating the states or outputs of a Markov process observed in white noise and it was used for the simulations results published in the reference papers describing turbo codes [8][21]. Although the MAP algorithm yields 0.7 dB gain over SOVA when used for turbo decoding [67], it was at that time likely to be considered too complex for implementation in a real system, due to the numerical representation of probabilities, the use of non-linear functions (exponential for transmission in Gaussian channel) and the high number of multiplications and additions. However, due to the existence of simplified versions of the MAP algorithm operating in the logarithmic domain [68][69], it quickly became the reference algorithm for turbo decoding in software as well as in hardware implementations.

This next section presents the MAP algorithm and it variants in the logarithmic domain: Log-MAP and Max-Log-MAP [67]. They are illustrated with the memory-2 RSC code described in Figure 26.

![Recursive systematic convolutional (RSC) code with transfer function](image)

**4.2.1. Definitions**

Let us consider the transmission chain shown in Figure 27. The systematic and redundancy symbols $X_i$ and $Y_i$, $i = 1 \cdots K$ are transmitted over the transmission channel using a binary antipodal modulation: $X_i$ or $Y_i = 0$ is transmitted as -1 and $X_i$ or $Y_i = 1$ is transmitted as +1. For the sake of simplicity in the notations, we use notations $X_i$ and $Y_i$ indifferently to refer to the encoded bits taking values 0 or 1 and to the transmitted modulated symbols, taking values +1 or -1.
At the receiver side, the received sequence is \( R^K_i = \{R_1, \ldots, R_K\} \), with \( R_i = (x_i, y_i) \), for \( i = 1 \cdots K \).

In the context of SISO decoding of convolutional codes, the MAP algorithm and its derivatives aims at calculating the logarithm of the likelihood ratio (LLR) related to each source data \( d_i \) for \( i = 1 \cdots K \):

\[
\Lambda(d_i) = \ln \frac{\Pr(d_i = 1 | R^K_i)}{\Pr(d_i = 0 | R^K_i)}
\]

where \( \Pr(d_i = j | R^K_i), j = 0,1 \) is the a posteriori probability (APP) of bit \( d_i \).

The hard decision related to \( d_i \) is given by the sign of \( \Lambda(d_i) \):

\[
\hat{d}_i = 1 \text{ if } \Lambda(d_i) \geq 0
\]

\[
\hat{d}_i = 0 \text{ if } \Lambda(d_i) < 0
\]

and its magnitude provides the reliability of the decision.

### 4.2.2. The MAP algorithm

We do not repeat the complete derivation of the MAP algorithm here, but only state the main results, illustrated with the code of Figure 26. More detailed derivations can be found in [66], [1], [21], [70] and [71].

The code trellis is introduced in the derivation of the APP of \( d_i \) with the joint probability \( \lambda^i_j(m) \) defined as

\[
\lambda^i_j(m) = \Pr(d_i = j, S_i = m | R^K_i)
\]

where \( S_i \) is the encoder state after the encoding of bit \( d_i \).

The LLR of bit \( d_i \) can then be expressed as

\[
\Lambda(d_i) = \ln \frac{\sum_m \lambda^i_j(m)}{\sum_m \lambda^i_0(m)}
\]

For the code of Figure 26, \( \Lambda(d_i) \) is written as

\[
\Lambda(d_i) = \frac{\lambda^i_1(0) + \lambda^i_1(1) + \lambda^i_1(2) + \lambda^i_1(3)}{\lambda^i_0(0) + \lambda^i_0(1) + \lambda^i_0(2) + \lambda^i_0(3)}
\]

The principle of the MAP algorithm consists in processing separately data encoded between time 1 and time \( i \) (past data) and data encoded between time \( i + 1 \) and time \( K \) (future data) to compute...
probabilities $\lambda_i^f(m)$. This dichotomous processing is achieved by introducing probability functions $\alpha_i(m), \beta_i(m)$ and $\gamma_i(m)$ defined by

Forward state probabilities: $\alpha_i(m) = \Pr\{S_i = m | R^i_j\}$ (13)

Backward state probabilities: $\beta_i(m) = \frac{\Pr[R^K_{i+1} | S_i = m]}{\Pr[R^K_{i+1} | R^i_j]}$ (14)

State transition probabilities: $\gamma_j(R_i, m', m) = \Pr\{d_i = j, S_i = m, R_i | S_{i-1} = m', j = 0, 1\}$ (15)

One can show that

$$\Lambda(d_i) = \ln \frac{\sum_m \sum_{d(m', m) = 1} \gamma_j(R_i, m', m) \alpha_{i-1}(m') \beta_i(m)}{\sum_m \sum_{d(m', m) = 0} \gamma_j(R_i, m', m) \alpha_{i-1}(m') \beta_i(m)}$$ (16)

The application of (16) to the example of Figure 26 yields

$$\Lambda(d_i) =$$

$$\ln \left[ \frac{\gamma_j(R_i,1,0) \alpha_{i-1}(1) \beta_i(0) + \gamma_j(R_i,2,1) \alpha_{i-1}(2) \beta_i(1) + \gamma_j(R_i,0,2) \alpha_{i-1}(0) \beta_i(2) + \gamma_j(R_i,3,3) \alpha_{i-1}(3) \beta_i(3)}{\gamma_j(R_i,0,0) \alpha_{i-1}(0) \beta_i(0) + \gamma_j(R_i,1,2) \alpha_{i-1}(1) \beta_i(2) + \gamma_j(R_i,2,3) \alpha_{i-1}(2) \beta_i(3) + \gamma_j(R_i,3,1) \alpha_{i-1}(3) \beta_i(1)} \right]$$

4.2.2.1. Computation of state probabilities $\alpha_i(m)$ and $\beta_i(m)$

Forward probabilities can be computed recursively through a forward recursion in the trellis:

$$\alpha_i(m) = \sum_{m'} \sum_{j=0,1} \alpha_{i-1}(m') \gamma_j(R_i, m', m)$$ (17)

$\alpha_i(m)$ values can then be all computed from initial values $\alpha_0(m)$. If $m_0$ is the initial state of the encoder, then $\alpha_0(m_0) = 1$ and $\alpha_0(m) = 0$ for $m \neq m_0$. If the initial state is unknown, $\alpha_0(m) = \frac{1}{2^n}$ for all $m$, where $n$ is the code memory. When circular encoding is implemented, one can use the value of $\alpha_K(m)$ as initial value for $\alpha_0(m)$, for every trellis state $m$, since the initial and final states are identical. This initialization method is particularly convenient for iterative decoding, since the values of $\alpha_K(m)$ obtained at the end of iteration $lt$ are simply handed to $\alpha_0(m)$ before starting the forward recursion of the $(lt+1)^{th}$ decoding iteration.

To avoid numerical precision problems when implementing the algorithm, it is highly recommended to normalize the $\alpha_i(m)$ values regularly. Since $\Lambda(d_i)$ is a ratio, it does not make any difference in the LLR calculation.

Figure 28 shows an example of forward state probability computation using the trellis of the code of Figure 26.
\( \alpha_i(0), \alpha_i(1) \) and \( \alpha_i(2) \) are computed by:

\[
\alpha_i(0) = \alpha_{i-1}(0) \gamma_0(R_i, 0, 0) + \alpha_{i-1}(1) \gamma_1(R_i, 1, 0)
\]

\[
\alpha_i(1) = \alpha_{i-1}(3) \gamma_0(R_i, 3, 1) + \alpha_{i-1}(2) \gamma_1(R_i, 2, 1)
\]

\[
\alpha_{i+1}(2) = \alpha_i(1) \gamma_0(R_i, 1, 2) + \alpha_i(0) \gamma_1(R_i, 0, 2)
\]

In a similar way, backward probabilities can be computed recursively through a backward recursion in the trellis:

\[
\beta_i(m) = \sum_{m'} \sum_{j=0,1} \beta_{i+1}(m') \gamma_j(R_{i+1}, m, m')
\]

\( \beta_i(m) \) values can then be all computed from final values \( \beta_K(m) \). If \( m_K \) is the final state of the encoder, then \( \beta_K(m_K) = 1 \) and \( \beta_K(m) = 0 \) for \( m \neq m_K \). If the final state is unknown, \( \beta_K(m) = \frac{1}{2^{\nu}} \) for all \( m \), where \( \nu \) is the code memory. When circular encoding is implemented, one can use the value of \( \beta_0(m) \) as initial value for \( \beta_K(m) \), for every trellis state \( m \). Then, during the iterative decoding process, the values of \( \beta_0(m) \) obtained at the end of iteration \( lt \) are passed to \( \beta_K(m) \) as initial values for the backward recursion of the \((lt+1)^{th}\) decoding iteration.

Regular normalization of the \( \beta_i(m) \) values is also recommended for the same reason as mentioned for the computation of forward state probabilities.

Figure 29 shows an example of forward state probability computation using the trellis of the code of Figure 26.
Figure 29: Example of computation of $\beta_i(0)$, $\beta_i(2)$ and $\beta_{i-1}(1)$ for the code of Figure 26

$\beta_i(0)$, $\beta_i(2)$ and $\beta_{i-1}(1)$ are computed by:

$\beta_i(0) = \beta_{i+1}(0)\gamma_0(R_{i+1}, 0, 0) + \beta_{i+1}(2)\gamma_1(R_{i+1}, 0, 2)$

$\beta_i(2) = \beta_{i+1}(3)\gamma_0(R_{i+1}, 2, 3) + \beta_{i+1}(1)\gamma_1(R_{i+1}, 2, 1)$

$\beta_{i-1}(1) = \beta_i(2)\gamma_0(R_i, 1, 2) + \beta_i(0)\gamma_1(R_i, 1, 0)$

**4.2.2.2. Computation of state transition probabilities $\gamma_j(R_i, m', m)$**

The state transition probabilities $\gamma_j(R_i, m', m)$ can be expressed as the product of three terms:

$\gamma_j(R_i, m', m) = \Pr\{d_i = j|S_i = m, S_{i-1} = m'\}\Pr\{d_i = j\}\Pr\{R_i|d_i = j, S_i = m, S_{i-1} = m'\}$ (19)

The first term is equal to one if there is a transition between states $m'$ and $m$ in the trellis corresponding to $d_i = j$ and is equal to zero otherwise. The second term is the *a priori* information related to $d_i = j$: in a non-iterative process or at the first turbo decoding iteration, it is given by the source statistics; in an iterative process, it is provided by the output extrinsic information computed by the other component decoder. The third term is given by the transition probability of the transmission channel.

Therefore, if we consider a transmission in the discrete Gaussian memoryless channel, with noise variance $\sigma^2$, the non-zero terms $\gamma_j(R_i, m', m)$ can be written as

$\gamma_j(R_i, m', m) = \Pr\{d_i = j\} \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x_i-x_j)^2}{2\sigma^2}\right) \exp\left(-\frac{(y_i-y_j)^2}{2\sigma^2}\right)$ (20)

Since $X_i$ and $Y_i$ are equal to +1 or -1, (18) can also be written as

$\gamma_j(R_i, m', m) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x_i^2+y_i^2+2}{2\sigma^2}\right) \exp\left(\frac{x_i y_i}{\sigma^2}\right) \Pr\{d_i = j\}$ (21)
The two first terms of (21) do not depend on \( j \) and are identical for all state transitions in the trellis. Since \( \Lambda(d_i) \) is a ratio, they can be omitted in the expressions of \( \alpha_i(m) \) and \( \beta_i(m) \).

In practice the following simplified expressions of \( \gamma_j(R_i, m', m) \) are used in (16), (17) and (18) for the LLR computation:

\[
\gamma_1^i(R_i, m', m) = \exp \left( \frac{x_i}{\sigma^2} \right) \exp \left( \frac{y_i Y_i}{\sigma^2} \right) \Pr[d_i = 1] = \exp \left( \frac{x_i + y_i Y_i}{\sigma^2} + \ln \Pr[d_i = 1] \right) \tag{22}
\]

\[
\gamma_0^i(R_i, m', m) = \exp \left( -\frac{x_i}{\sigma^2} \right) \exp \left( \frac{y_i Y_i}{\sigma^2} \right) \Pr[d_i = 0] = \exp \left( -\frac{x_i + y_i Y_i}{\sigma^2} + \ln \Pr[d_i = 0] \right) \tag{23}
\]

where the value of \( Y_i \) depends on the state transition \((m', m)\) considered.

### 4.2.2.3. Extraction of extrinsic information from LLR

Introducing (22) and (23) into (16) yields

\[
\Lambda(d_i) = \frac{2x_i}{\sigma^2} + \ln \frac{\Pr[d_i = 1]}{\Pr[d_i = 0]} + \ln \left( \sum_m \sum_{m'/d(m,m')} \alpha_{i-1}(m') \beta_i(m) \right)
\]

\[
\Lambda(d_i) = \frac{2x_i}{\sigma^2} + L_a + Z_i \tag{24}
\]

The first term of (24) is the **intrinsic information** related to \( d_i \), available at the channel output, the second term is the **a priori information** related to \( d_i \) and the third term is the **extrinsic information** \( Z_i \) produced by the decoder.

In the context of turbo decoding, at iteration \( lt \), the **a priori information** for a given decoder is provided by the extrinsic information computed by the other SISO decoder at iteration \( lt-1 \).

### 4.2.3. The MAP algorithm in the logarithmic domain: Log-MAP and Max-Log-MAP

Decoding following the symbol-by-symbol MAP criterion requires a large number of operations, including multiplications and calculating exponentials. A way of simplifying the decoding algorithm involves re-writing it in the logarithmic domain: exponentials disappear, multiplications become additions, but what about additions?
The problem of computing terms $\ln(\exp a_1 + \exp a_2 + \cdots + \exp a_n)$ can be solved by recursively using the Jacobi’s logarithm [72, 67]-[69]:

$$\max^* (a_1, a_2) \triangleq \ln(\exp a_1 + \exp a_2) = \max(a_1, a_2) + \ln(1 + \exp(-|a_1 - a_2|)) \quad (25)$$

In the logarithmic domain, the probability functions defined in section 4.2.2 are replaced by metrics:

Forward state metric: $M^F_i(m) \triangleq \sigma^2 \ln a_i(m)$ \hspace{1cm} (26)

Backward state metric: $M^B_i(m) \triangleq \sigma^2 \ln \beta_i(m)$ \hspace{1cm} (27)

State transition metric: $m^T_i(m', m) \triangleq \sigma^2 \ln y_j(R_i, m', m)$ \hspace{1cm} (28)

In the discrete Gaussian memoryless channel, the state transition metrics inferred from (22) and (23) can be written

$$m^T_i(m', m) = x_i + y_i Y_i + \sigma^2 \ln \Pr\{d_i = 1\} \quad (29)$$

$$m^T_i(0, m) = -x_i + y_i Y_i + \sigma^2 \ln \Pr\{d_i = 0\} \quad (30)$$

Using the $\max^*$ function, the LLR expression can be reformulated as

$$\Lambda(d_i) = \ln \sum_m \sum_{m'} \exp \left( \frac{m^T_i(m', m) + M^F_{i-1}(m') + M^B_i(m)}{\sigma^2} \right)$$

$$- \ln \sum_m \sum_{m'} \exp \left( \frac{m^T_i(m', m) + M^F_{i-1}(m') + M^B_i(m)}{\sigma^2} \right)$$

that is

$$\Lambda(d_i) = \max^*_{(m', m)/d(m', m)=1} \left( \frac{m^T_i(m', m) + M^F_{i-1}(m') + M^B_i(m)}{\sigma^2} \right)$$

$$- \max^*_{(m', m)/d(m', m)=0} \left( \frac{m^T_i(m', m) + M^F_{i-1}(m') + M^B_i(m)}{\sigma^2} \right)$$

(29)

The forward and backward recursions also call for

$$M^F_i(m) = \max^*_{m', j=0,1} \left( M^F_{i-1}(m') + m^T_i(m', m) \right) \quad (30)$$

$$M^B_i(m) = \max^*_{m', j=0,1} \left( M^B_{i+1}(m') + m^T_i(m, m') \right) \quad (31)$$
When the initial and final states are known, their metric are set to \(-\infty\) and the others are set to 0. When they are unknown, all the initial/final metrics can set to the same (arbitrary) value. The comment related to tail-biting codes mentioned in section 4.2.2.1 also applies here.

Two variants of the algorithm can be implemented in practice:

1. **Log-MAP algorithm**: the correction term \(\ln(1 + \exp(-|a_1 - a_2|))\) is precomputed and stored in a look-up table. Robertson et al. [67] have shown that a table size of eight is usually sufficient to keep the same performance as the original MAP algorithm.

2. **Max-Log-MAP algorithm**: The \(\max^*\) function is replaced by a simple \(\max(.)\) computation (Max-Log approximation)

\[
\max^*(a_1, a_2) \triangleq \ln(\exp a_1 + \exp a_2) \approx \max(a_1, a_2)
\]  

This simplified version of the MAP algorithm is the most currently used in hardware implementations of turbo decoders. Moreover, most of actual decoders compute a modified LLR \(\Lambda'(d_i)\) defined as

\[
\Lambda'(d_i) \triangleq \frac{\sigma^2}{2} \Lambda(d_i)
\]

Using the Max-Log approximation, \(\Lambda'(d_i)\) can be also expressed as

\[
\Lambda'(d_i) \approx \frac{1}{2} \left[ \max_{(m',m)/d(m',m)=1} \left( m_i^1(m',m) + M_i^{e-1}(m') + M_i^\beta(m) \right) - \max_{(m',m)/d(m',m)=0} \left( m_i^0(m',m) + M_i^{e-1}(m') + M_i^\beta(m) \right) \right]
\]

Extracting intrinsic and a priori information from the \(\max(.)\) functions yields

\[
\Lambda'(d_i) \approx \\
x_i + L_a + \frac{1}{2} \left[ \max_{(m',m)/d(m',m)=1} \left( y_i y_i + M_i^{e-1}(m') + M_i^\beta(m) \right) - \max_{(m',m)/d(m',m)=0} \left( y_i y_i + M_i^{e-1}(m') + M_i^\beta(m) \right) \right]
\]

With this formulation, the extrinsic information term is at the same scale as the intrinsic information present at the channel output and can be easily obtained through subtraction as described in the figures of section 4.1. An interesting ensuing property of the Max-Log MAP decoder compared to the original MAP is that the noise variance, and consequently the signal-to-noise ratio information, is not required for decoding.

Due to its sub-optimality, the use of the Max-Log MAP algorithm instead of the original MAP for turbo decoding entails a performance loss of about 0.5 dB [67]. In order to compensate for this loss, the extrinsic information can be scaled before being used by a SISO decoder [75][76]. In order to guarantee the stability of the looped structure, the scaling factor is lower than 1. It can vary over the iterations, for example from 0.7 at the first iteration to 1 for the last iteration. An additional clipping
operation can also participate to the stability of the turbo process: a typical value of the maximum
dynamics of the extrinsic information is twice the input dynamics of the decoder. This compensation
measures allow the performance loss to be lowered to 0.1-0.3 dB, depending on the block size and
the component codes.

5. Industrial impacts of turbo codes

Turbo codes are an outcome of the research of the Electronics Department of Telecom Bretagne in
the field of algorithm/silicon interaction. Since such an activity involves jointly devising new
algorithms and innovative hardware architectures, the very first codec circuits were designed in
parallel with the development of the algorithms. In the first part of this section, we present the
detailed features of the two first turbo codec circuits that were designed in close collaboration with
C. Berrou’s team. They allowed the main concepts of turbo decoding detailed in section 2 to be
validated as well as the behavior of turbo codes at low error rates to be more clearly understood.
The concurrent development of algorithms and circuits gave a boost to the adoption of turbo codes
in commercial transmission systems: although turbo codes had, at first, a reputation for presenting
non-tractable decoding complexity, the early existence of circuits clearly proved the feasibility of
turbo decoders. Nowadays, turbo codes are used in numerous applications, in proprietary as well as
in standardized systems. The second part of this section describes the pioneer applications having
adopted turbo codes. Finally, the third part gives an overall picture of the current telecommunication
standards including this family of codes.

5.1. The very first implementations of turbo codecs

The very first circuit that proved the concept of iterative decoding of parallel concatenated code was
marketed by the component manufacturer Comatlas under the reference CAS 5093 [64] from
September 1993, that is less than six months after the presentation of the first paper in conference
[8]. It stemmed from a collaboration between Telecom Bretagne, France Telecom, Comatlas and US
ASIC company VLSI Technology Inc., with the financial support of the Brittany Region and was
partially designed by a group of third year students of Telecom Bretagne. The second circuit Turbo4 is
the outcome of a common work of France Telecom and Telecom Bretagne. The aim of was to design
a modular structure that could be pipelined to perform an arbitrary number of decoding iterations,
in order to reach bit error rates in the order of $10^{-10}$. A third circuit was designed a few years later and
implemented in a FPGA in order to prove the validity of the turbo decoding principle for product
codes [77][78]. Nowadays, turbo decoder circuits call for architectures substantially different from
those presented in this section (see chapter related to turbo decoder architecture).
5.1.1. The CAS 5093 circuit

The CAS 5093 circuit consists of a turbo encoder based on the binary 8-state component code with transfer function \( \frac{1}{1+D+D^2+D^3+D^5} \) and a turbo decoder implementing 2.5 decoding iterations. This circuit aimed at encoding and decoding continuous data flows, typically for broadcasting applications. Therefore, no trellis termination technique is applied and the decoder calls for a pipelined architecture: each iteration requires the implementation of two SISO component decoders which are separated by interleaving and de-interleaving memories. Figure 30 shows a simplified architecture of the turbo decoder. Five component decoders are implemented: four SISO decoders, named SOD1 to SOD4, and a soft-input hard-output decoder named HOD performing a simple Viterbi algorithm for the last half-iteration. No de-interleaving memory is required after the fifth half iteration since the HOD decoder processes the data in the natural order.

![Simplified architecture of the CAS5093 circuit](image)

Figure 30: Simplified architecture of the CAS5093 circuit (excerpt from datasheet [64]).

The SISO decoders implement a simplified version of the SOVA algorithm [11] using the well-known register exchange architecture. The elementary cells, consisting of a D flip-flop and a multiplexer, had been specifically designed and optimized by VLSI Technology Inc. for the purpose of this circuit design. The permutation is carried out by a convolutional interleaver [79], well suited for continuous data flows. Its implementation calls for a 1024-bit dual-port memory, one port being dedicated for reading and the other for writing. The permutation equations guarantee that no conflict occurs between the writing and reading addresses. They also ensure a minimum spread of 16 for direct neighboring data but no specific optimization had been performed related to the minimum Hamming
distance of the resulting turbo code. A puncturing device is implemented to achieve coding rate 
\( R = 1/2 \), that guarantees in cooperation with the interleaver that each information data is 
transmitted with exactly one parity data.

For continuous transmissions, convolutional codes do not require any specific synchronization at the 
receiving side. However, for turbo codes, the interleavers and de-interleavers in the decoder have to 
be synchronized with the interleaver in the encoder. To this end, an original synchronization 
technique is implemented: after the encoding process, some bits are inverted following a fixed 
pattern. The first SISO decoder SOD1 detects the pattern and synchronizes the (de)interleavers [80]. 
A supervising controller is in charge of detecting false synchronizations or synchronization losses 
using the pseudo-syndrome technique [81][82] and of restarting the synchronization procedure when 
required. The pseudo-syndrome technique is also used to clear up the phase ambiguities for BPSK 
and QPSK modulations.

The CAS 5093 chip was implemented in 0.8 µm CMOS technology. It contains 485,000 transistors and 
comes in a 68-pin PLCCC package. It is able to turbo encode and decode data flows at a maximum 
throughput of 40 Mbit/s. A picture of the chip layout is presented in Figure 31.

![Figure 31: Photograph of the CAS 5093 chip layout.](image)

### 5.1.2. The Turbo4 circuit

The successor of the CAS 5093 circuit, Turbo4, got its name from the use of memory-4 component 
codes instead of 3 for CAS 5093. Like its predecessor, it is intended for continuous transmissions but 
the size of the convolutional interleaver is doubled. The chip consists of a turbo encoder based on 
the binary 16-state RSC code with transfer function \[ \frac{1+D+D^2+D^4}{1+D^3+D^4} \] and two SOVA decoders able to 
perform one decoding iteration [65]. A complete decoder performing a given number of iterations
can be prototyped by pipelining the corresponding number of circuits. A puncturing device using regular patterns allows the coding rate to be raised to 1/2, 2/3, 3/4 and 4/5. Numerous laboratory tests carried out with a hardware channel emulator allowed a better understanding of the asymptotic behavior of turbo codes [83].

The Turbo4 chip was implemented in a double level metal 0.8 µm CMOS technology. It contains 600,000 transistors in a die size of 78 mm². A picture of the chip layout is presented in Figure 32.

5.2. Early applications of turbo codes

Immediately the first results related to turbo codes have been published, several standardization committees got interested in this new family of error correcting codes. In particular, in 1994, the Digital Video Broadcasting (DVB) European-based consortium considered turbo codes for their future standard for the broadcast transmission of digital terrestrial television, DVB-T. However, most of the group members had not enough time to get familiar with this brand new technique and they were afraid of adopting a code that seemed complex to decode and that could induce possible non-controlled side effects. The group finally adopted the conventional concatenation of the outer (204, 188, 8) Reed-Solomon code with an inner 64-state convolutional code.

In June 1993, the European Space Agency (ESA) issued an invitation to tender (ITT) aimed at improving the performance of deep-space communications. At that time, ESA and the National Aeronautics and Space Administration (NASA) used the concatenation of an outer (255,223) Reed-
Solomon code and an inner 64-state convolutional code with coding rate 1/2 for such applications. This coding scheme had been recommended in 1987 by the Consultative Committee for Space Data Systems (CCSDS) for telemetry channel coding. As a response to the ITT, C. Berrou proposed a 16-state turbo code offering four coding rates from lower than or equal to 1/2. This scheme was then recommended by the CCSDS in the next *Telemetry channel coding blue book* issued in May 1999 [85] and is still present in the most recent recommendations [86].

The structure of the CCSDS turbo encoder, taken from [86], is presented in Figure 33. It calls for two 16-state component RSC codes with recursion polynomial \([1 + D^3 + D^4]\) and with redundancy polynomials \([1 + D + D^3 + D^4, 1 + D^2 + D^4, 1 + D + D^2 + D^3 + D^4]\). Four coding rates are offered: 1/2, 1/3, 1/4 and 1/6. The *information block buffer* is in charge of the permutation, which is specified using simple equations: this form avoids the storage of addresses in a memory. The CCSDS turbo code allows four different data block sizes to be encoded, from 1784 bits to 8920 bits. The conventional zero termination technique described in section 3.2 is adopted and is performed by turning the two input switches in upper position (see Figure 33). Therefore, the effective coding rates are slightly lower than the reference rates and the termination data do not benefit from the double encoding of the turbo code.
The first commercial use of turbo codes saw the light in 1997 with Inmarsat’s M4 multimedia service by satellite [87]. This new service used the component codes of Turbo4 with a 16-QAM modulation and allowed the user to communicate with Inmarsat-3 spot-beam satellite from a laptop-sized terminal at 64 kbit/s.

The appearance of turbo codes in commercial applications encouraged several teams to put significant effort in the implementation of turbo decoders: for instance, at University of South Australia, A. Barbulescu developed a complete Inmarsat M4 modem based on a DSP and a FPGA board [88][89]. Some companies also came into play: S. Pietrobon pioneered this way by creating the Australian company Small World Communications [90] in January 1997. This company is still specialized in the design of state-of-the-art error correcting encoders and decoders and proposed a MAP-based turbo decoder in its IP core list in 1998 [91]. A few months later in Europe, N. Brengarth and J. Tousch, with a fresh degree from Telecom Bretagne, created the French company TurboConcept [92] in October 1999 and proposed their first turbo decoder three months later [93].

The intellectual property rights on turbo codes are covered and protected by patents, filed by Telecom Bretagne and owned by France Telecom/Orange and Telediffusion de France (TDF). In the early 1990s, Telecom Bretagne belonged to France Telecom which was a public service. When France Telecom was privatized in 1996, Telecom Bretagne came under the aegis of the French Ministry of Industry but the patents remained the property of France Telecom. A licensing program called Turbo Codes Licensing Program (TCLP) [94] was then set up to facilitate the access to the pool of related patents. Among other things, the TCLP provides a list of IP Core providers accredited by France Telecom [95]. Nevertheless, the essential patents [1][2][3] related to turbo encoding and decoding are now in the public domain and the TCLP program does not apply to all the systems implementing turbo codes any longer.

The fast growing general interest for turbo codes soon after their invention created favorable conditions for their adoptions in telecommunication standards. A detailed snapshot of the situation in 2007 can be found in [96], whose main elements are presented in the next section and are completed with recent evolutions.

5.3. Turbo codes in standards

Nowadays, many telecommunication standards have adopted turbo codes for the physical layer of communication systems. This section lists the mains standards, especially those addressing mobile
telephony and digital video broadcasting. A table providing a synthetic view of the standardized codes is given at the end of the section.

5.3.1. Mobile communication systems

In the late 1990s, the 3rd Generation Partnership Project (3GPP) [97] adopted a turbo code for the protection of data in the third generation (3G) mobile communication systems, in particular in the Universal Mobile Telecommunications System (UMTS), whose first services were introduced in 2001. UMTS uses Wideband Code Division Multiple Access (W-CDMA) to carry the radio transmissions and the binary turbo code [25][26] with coding rate 1/3 given in Figure 34. The transfer function of the 8-state constituent codes is $[1, 1+D+D^3]$ and block coding is achieved with conventional zero termination (see section 3.2). A non-regular interleaver is specified in rectangular form for block sizes ranging from 40 to 5114 bits.

![Figure 34: Specification of the 3GPP turbo code (extract from [25]).](image)

At the same time, the 3rd Generation Partnership Project 2 (3GPP2) [98] aimed at defining the specifications of 3G mobile phone systems compatible with the International Telecommunication Union’s IMT-2000 initiative [99]. In 1999, this initiative led to the specification of the cdma2000 family of standards for spread spectrum systems [100]. The turbo code adopted in this standard [101] is an extended version of the 3GPP turbo code since it provides an additional redundancy output provided by generator polynomial $[1 + D + D^2 + D^3]$ which allows an overall coding rate as low as 1/5.
The evolution of mobile communication systems has now arrived at the fourth generation (4G) with the specifications of the Long Term Evolution (LTE) [27] and the LTE-Advanced [102] standards. In these specifications, the turbo code is based on the same constituent codes as the 3G systems but an enhanced interleaver was defined, using QPP permutation with the contention-free property and allowing a high degree of freedom for parallel processing (see section 3.3.2).

5.3.2. Digital Video Broadcasting (DVB) standards

The main second generation DVB standards [103], defining the broadcasting of TV and multimedia services through terrestrial, satellite and cable networks, adopted low-density parity-check (LDPC) codes [104] for forward error correction. Nevertheless, several subsidiary standards have adopted turbo codes. Firstly, the DVB-SH (satellite to handeld) standard dedicated to the delivery multimedia content to handheld terminals using a hybrid satellite/terrestrial downlink reused the 3GPP2 turbo code [105]. Moreover, the standards describing the return channel for DVB-T (terrestrial) and DVB-S/S2 (satellite) are protected with turbo codes. The existence of return channels is crucial to enlarge broadcasting services to interactive TV programs or internet access.

The DVB-RCS standard [33], published in 2000, specifies the return channel via satellite for DVB-S systems. A double-binary turbo code [22][93]using the tail-biting termination was adopted for the first time. The interleaver is based on an ARP permutation (see section 3.3.2) allowing some parallelism in the decoding process. This is a flexible code, able to process 12 block sizes (from 12 to 216 bytes) and 7 coding rates (from 1/3 to 6/7) providing good performance-complexity trade-off. The structure of the turbo encoder is given in Figure 35. The DVB-RCT standard [35], which specifies the return channel for DVB-T terrestrial systems, is based on the same code but with different block sizes.

![Figure 35: Structure of the DVB-RCS turbo encoder (extract from [33]).](image)
DVB-RCS was followed by the second generation standard DVB-RCS2 in 2012, which also adopted a turbo code of the same family. However, the double-binary 8-state component codes were replaced by double-binary 16-state codes, thus conferring higher minimum Hamming distances and therefore better low error rate performance to the resulting turbo code. The structure of the DVB-RCS2 turbo encoder is given in Figure 36.

![Figure 36: Structure of the DVB-RCS2 turbo encoder (extract from [34]).](image)

### 5.3.3. Other standards

Double-binary turbo codes have also been present in the IEEE 802.16 family of wireless communications standards ratified by the WiMAX (Worldwide Interoperability for Microwave Access) forum [106] since 2004 [36]. The code is similar to the one adopted in DVB-RCS (see Figure 35) except that redundancy $W$ is not used, thus limiting the lowest value of coding rate to 1/2 instead of 1/3. This code can also be found in the communications systems supported by the HomePlug Alliance [107], a trade association of electronics manufacturers, service providers, and retailers that establishes standards and certifies devices in the field of power line communications. The HomePlug
AV [108] and Homeplug AV2 specifications [109] were issued in 2005 and 2012 respectively and HomePlug AV became an IEEE standard in 2010 [111].

5.3.4. Summary

Table 4 recapitulates the current communication standards using turbo codes and provides the main features of their constituent convolutional codes.

<table>
<thead>
<tr>
<th>Standard</th>
<th>Input symbol type</th>
<th>Number of states</th>
<th>Trellis termination</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCSDS [86]</td>
<td>binary</td>
<td>16</td>
<td>Zero termination</td>
</tr>
<tr>
<td>UMTS [27]</td>
<td>binary</td>
<td>8</td>
<td>Zero termination</td>
</tr>
<tr>
<td>CDMA2000 [101]</td>
<td>binary</td>
<td>8</td>
<td>Zero termination</td>
</tr>
<tr>
<td>LTE [25][26]</td>
<td>binary</td>
<td>8</td>
<td>Zero termination</td>
</tr>
<tr>
<td>LTE-Advanced [102]</td>
<td>binary</td>
<td>8</td>
<td>Zero termination</td>
</tr>
<tr>
<td>DVB-RCS [33]</td>
<td>double-binary</td>
<td>8</td>
<td>Tail-biting</td>
</tr>
<tr>
<td>DVB-RCT [35]</td>
<td>double-binary</td>
<td>8</td>
<td>Tail-biting</td>
</tr>
<tr>
<td>DVB-RCS2 [34]</td>
<td>double-binary</td>
<td>16</td>
<td>Tail-biting</td>
</tr>
<tr>
<td>DVB-SH [105]</td>
<td>binary</td>
<td>8</td>
<td>Zero termination</td>
</tr>
<tr>
<td>WiMax (IEEE 802.16) [110]</td>
<td>double-binary</td>
<td>8</td>
<td>Tail-biting</td>
</tr>
<tr>
<td>HomePlug AV (IEEE 1901) [111]</td>
<td>double-binary</td>
<td>8</td>
<td>Tail-biting</td>
</tr>
<tr>
<td>HomePlug AV2</td>
<td>double-binary</td>
<td>8</td>
<td>Tail-biting</td>
</tr>
</tbody>
</table>

Table 4: List of communication standards having adopted a parallel concatenation of convolutional codes.

6. Conclusion

This chapter has presented the main concepts of turbo coding put in an historical perspective. The overall structures of the encoder and decoder have been analyzed and explained. Particular stress was laid on the component code and permutation properties. Then, the necessary notations related to the decoder have been introduced and the main soft-input soft-output decoding algorithms have
been briefly described. Finally, the very first proof-of-concept hardware implementations have been described and the main telecommunication applications and current transmission standards using turbo codes have been reviewed.

7. References


[97] http://www.3gpp.org/

[98] http://www.3gpp2.org/


[100] 3rd generation partnership Project 2 ; Introduction to cdma2000 standards for spread spectrum systems; 3GPP2 C.50001-0, v1.0, July 1999.


[105] Digital Video Broadcasting (DVB); *Framing structure, channel coding and modulation for satellite services to handheld devices (SH) below 3 GHz*, ETSI EN 302 583, v1.2.1, Dec. 2011.


