Turbo-like Codes Constructions

Sergio Benedetto, Dariush Divsalar and Guido Montorsi

Abstract

This chapter provides design, analysis, construction, and performance of the turbo codes, serially concatenated codes, and turbo-like codes including the design of interleavers in concatenation of codes. Also this chapter describes the iterative decoding algorithms for these codes. Performance of parallel versus serial concatenation using analysis and iterative decoding simulations is included. Extension to multiple code concatenation is discussed.

Index Terms

Turbo codes, serial codes, turbo-like codes, interleavers, iterative decoding.

CONTENTS

I Introduction and bibliography survey 3
I-A Introduction ............................................ 3

II Structure of concatenated codes 8
II-A Main characteristics of turbo encoding structures ............................................. 8
II-B Trellis encoders ........................................... 10
II-C Mapper .................................................... 11
II-C1 Repeater .................................................. 12
II-C2 Parity-check bit generator ........................................... 13
II-C3 Constellation labeling ......................................... 13
II-D Interleaver ............................................... 13
II-E Rate conversion modules ....................................... 15
II-F Puncturer .................................................. 15
II-G Summary of encoding modules ..................................... 16
II-H Some turbo-encoder structures and their main properties .................................. 16
II-H1 Serially Concatenated Convolutional Codes ............................................. 16
II-H2 Duobinary PCCC ............................................ 17
II-H3 (Irregular) Repeat accumulate codes ............................................. 17
II-H4 Self concatenation ........................................... 18
II-H5 Other turbo coding structures ........................................... 18
II-I Convolutional versus block encoding ............................................. 19
II-I1 Periodic state enforcing of trellis encoders ............................................. 19

November 27, 2013 DRAFT
### III ML analysis and design of constituent codes

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>III-A</td>
<td>Maximum likelihood analysis</td>
<td>21</td>
</tr>
<tr>
<td>III-A1</td>
<td>Word and bit error probabilities</td>
<td>21</td>
</tr>
<tr>
<td>III-A2</td>
<td>Uniform Interleaver</td>
<td>22</td>
</tr>
<tr>
<td>III-A3</td>
<td>Analysis of PCCC</td>
<td>22</td>
</tr>
<tr>
<td>III-A4</td>
<td>Analysis of SCCC</td>
<td>25</td>
</tr>
<tr>
<td>III-A5</td>
<td>Analysis of hybrid concatenated codes with interleavers</td>
<td>25</td>
</tr>
<tr>
<td>III-A6</td>
<td>More refined upper bounds</td>
<td>26</td>
</tr>
<tr>
<td>III-B</td>
<td>Design criteria for constituent encoders</td>
<td>27</td>
</tr>
<tr>
<td>III-B1</td>
<td>Design of parallel concatenated convolutional codes with interleaver</td>
<td>28</td>
</tr>
<tr>
<td>III-B2</td>
<td>A heuristic explanation of the interleaver gain</td>
<td>31</td>
</tr>
<tr>
<td>III-B3</td>
<td>Design of serially concatenated convolutional codes with interleavers</td>
<td>32</td>
</tr>
<tr>
<td>III-C</td>
<td>Comparison between parallel and serially concatenated codes</td>
<td>33</td>
</tr>
<tr>
<td>III-D</td>
<td>Finding the optimum constituent encoders</td>
<td>34</td>
</tr>
</tbody>
</table>

### IV Iterative decoding

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV-A</td>
<td>Messages in iterative decoders and independence assumption</td>
<td>35</td>
</tr>
<tr>
<td>IV-B</td>
<td>Soft-Input Soft-output modules</td>
<td>38</td>
</tr>
<tr>
<td>IV-C</td>
<td>The SISO for the data ordering encoding modules</td>
<td>38</td>
</tr>
<tr>
<td>IV-D</td>
<td>The SISO module for the trellis encoder</td>
<td>39</td>
</tr>
<tr>
<td>IV-D1</td>
<td>The SISO algorithm for computing the extrinsic LLRs</td>
<td>39</td>
</tr>
<tr>
<td>IV-D2</td>
<td>Trellis with multiple symbol labels</td>
<td>41</td>
</tr>
<tr>
<td>IV-E</td>
<td>The SISO module for the mapper</td>
<td>41</td>
</tr>
<tr>
<td>IV-E1</td>
<td>Computation of SISO updating with the minimal trellis</td>
<td>41</td>
</tr>
<tr>
<td>IV-F</td>
<td>Multiple code representations</td>
<td>43</td>
</tr>
</tbody>
</table>

### V Interleaver designs

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V-A</td>
<td>Interleaver Theory</td>
<td>45</td>
</tr>
<tr>
<td>V-B</td>
<td>Interleavers: basic definitions</td>
<td>45</td>
</tr>
<tr>
<td>V-B1</td>
<td>Causal and canonical interleavers</td>
<td>48</td>
</tr>
<tr>
<td>V-C</td>
<td>Connections among interleaver parameters</td>
<td>50</td>
</tr>
<tr>
<td>V-D</td>
<td>Convolutional and Block interleavers</td>
<td>52</td>
</tr>
<tr>
<td>V-D1</td>
<td>Convolutional interleavers</td>
<td>52</td>
</tr>
<tr>
<td>V-D2</td>
<td>Implementation of convolutional interleavers with minimal memory requi</td>
<td>53</td>
</tr>
<tr>
<td>V-D3</td>
<td>Block interleavers</td>
<td>55</td>
</tr>
<tr>
<td>V-D4</td>
<td>Block interleaver causalization</td>
<td>55</td>
</tr>
<tr>
<td>V-D5</td>
<td>The canonical causal interleaver of a block interleaver</td>
<td>55</td>
</tr>
</tbody>
</table>
### V. Performance

**VI**  
Performance

<table>
<thead>
<tr>
<th>VI-A</th>
<th>Introduction</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI-B</td>
<td>Iterative decoding versus ML upper bounds</td>
<td>65</td>
</tr>
<tr>
<td>VI-C</td>
<td>Optimality of constituent encoders</td>
<td>66</td>
</tr>
<tr>
<td>VI-D</td>
<td>Performance versus number of iterations</td>
<td>67</td>
</tr>
<tr>
<td>VI-E</td>
<td>Comparison between PCCC and SCCC</td>
<td>68</td>
</tr>
<tr>
<td>VI-F</td>
<td>Other concatenated structures</td>
<td>68</td>
</tr>
<tr>
<td>VI-F1</td>
<td>Simulation results for variations of PCCC codes</td>
<td>68</td>
</tr>
<tr>
<td>VI-F2</td>
<td>Simulation results for variations of SCCC codes</td>
<td>74</td>
</tr>
<tr>
<td>VI-F3</td>
<td>Simulation results for multiple turbo codes (DPCCC)</td>
<td>76</td>
</tr>
<tr>
<td>VI-F4</td>
<td>Simulation results for Hybrid concatenated codes (HCCC)</td>
<td>79</td>
</tr>
<tr>
<td>VI-F5</td>
<td>Simulation results for Self Concatenated Codes</td>
<td>82</td>
</tr>
<tr>
<td>VI-F6</td>
<td>Simulation results for Repeat-Accumulate (RA) Codes</td>
<td>83</td>
</tr>
<tr>
<td>VI-F7</td>
<td>Simulation results for Repeat-Accumulate-Accumulate (RAA) Codes</td>
<td>84</td>
</tr>
</tbody>
</table>

### References

I. Introduction and Bibliography Survey

**A. Introduction**

Turbo codes have been introduced for the first time in a Session of ICC 1993, in Geneva [1]. Their simulated performance showed a bit error probability versus signal-to-noise ratio (SNR) only 0.7 dB worse than predicted by Shannon in his famous theorem, an unprecedented achievement since the pioneering asymptotic limit proved in 1948. The new, totally unexpected result, spurred a worldwide interest and in the following years thousand of papers were published on variation on the theme and its applications.

To keep the length reasonable, and to help the reader to extricating himself from the plethora of published papers on the subject, we have chosen to limit the bibliography to the most important contributions (meaning the most cited papers) over the years. We apologize to the authors who have been excluded and feel the exclusion inappropriate.
Turbo codes are not an evolutionary result of the mainstream coding theory, based on algebraic and convolutional codes and the concept of asymptotically good class of codes, which can be simply stated as follows:

An asymptotically good class of codes $C_n(n, k, d_{\text{min}})$ is such that for $n \to \infty$, both the rate $R_c = k/n$ and the normalized minimum Hamming distance $\delta = d_{\text{min}}/n$ stay bounded away from zero.

That definition of good codes relies on the concept of minimum distance, which in turn is strictly related to the word error probability of the code $P_w(e)$, i.e., the probability that a code word is misinterpreted by the decoder. Using maximum-likelihood decoding over an additive white Gaussian channel, the word error probability becomes

$$P_w(e) \leq \frac{1}{2} \sum_{d=d_{\text{min}}}^{n} A_d \text{erfc} \left( \frac{dR_c E_b}{N_0} \right)$$

(1)

where $A_d$ is the number of code words with Hamming weight $d$.

So, the classical conclusion is that good codes should have as large as possible their minimum Hamming distance and as small as possible the number of code words with a Hamming weight equal to the minimum distance. Decades of research have been then devoted to the design of codes with a distance spectrum matching those requirements. None of them, though, has been able to reach, or at least to closely approach, the promised land of the Shannon bound.

On the other hand, the central role played by the concept of random coding in the proof of the Shannon capacity theorem suggests that a code with a distance spectrum close to that of a randomly constructed code should be good. In fact, Shannon showed that almost all randomly constructed, long codes are good in the sense of achieving capacity. The real problem for them is their exponential (in the code word length) decoding complexity.

Emphasizing the role of minimum distance and simple decoding techniques has led to the construction of algebraic codes, whose hard decoding algorithm is greatly simplified owing to the powerful properties of Galois fields.

A totally different approach would be that of merging a design mimicking the random coding approach with that of decoding algorithms with polynomial, possibly linear, complexity. As we will see, this is the approach of turbo codes, which led David Forney to say in his Shannon lecture of the 1995 Information Theory Symposium: Rather than attacking error exponents, they (turbo codes) attack multiplicities, turning conventional wisdom on its head.

Since nothing remarkable grows in a petrified waterless desert, the invention of turbo codes had several precursive ideas. The importance of the random coding approach in designing good codes had been emphasized by [2]. The construction of powerful codes with polynomial decoding complexity was described in Forney’s PhD thesis on concatenated codes [3], a code construction based on the cascade of two codes, an outer and an inner code, and on a decoding algorithm that first decodes the inner and then the outer code. Finally, the idea of iterative soft decoding requires soft-input, soft-output (SISO) decoding algorithms, which were proposed by several authors, like [4], [5], and [6].

The structure of the classical turbo code is shown in Figure 1. It consists in the parallel concatenation of two convolutional constituent encoders, separated by an interleaver, whose function is to modify the order of the information symbols sent to the first encoder before presenting them to the second encoder. The interleaver is a key ingredient of the turbo code design, as it randomizes the code construction. The design of interleavers is the
subject of Section V.

Performance results in [1] were obtained by simulation. The first successful approach to derive maximum-likelihood (ML) performance of turbo codes was presented in [7]. It applied the random coding approach of Shannon to obtain the ML performance of a turbo code averaged with respect to all interleavers by introducing the concept of uniform interleaver. In the same paper, the authors provided a sound analytical explanation of the performance of turbo codes, and enhanced the role played by recursive convolutional encoders in the design.

It turns out that the ML performance of a turbo code in the low-medium SNRs region is dictated by a newly defined parameter, called effective free distance, defined as the minimum weight of coded sequences generated by input information sequences with Hamming weight equal to 2. To obtain good turbo codes, that parameter should be as large possible for both constituent encoders [8]. Tables of recursive convolutional codes with maximum effective free distance were presented in [9].

Comparing the code performance with the Shannon capacity bound requires the specification of the bit error probability at which the comparison is made. Indeed, the turbo code in [1] showed performance 0.7 dB far from the capacity limits at a bit error probability of $10^{-5}$. Capacity limits, however, hold theoretically for a vanishing bit error probability, and this raises a rather important question: do the amazing performance of turbo codes hold for very low bit error probabilities? This question has a great practical importance, since many system applications require very low bit error probabilities. To cite one case, optical fiber digital transmission requires a value as low as $10^{-15}$.

Examining the performance of a turbo code in a wide range of SNRs, three regions can be identified (see Figure 2:

- the non convergence region, where the bit error probability decreases very slowly with the increasing of the SNR
- the waterfall region, where the bit error probability drops almost vertically with the increasing of the SNR
- the error floor region, where the slope of the curve decreases significantly, so that a large increase in SNR is required to further improve the performance.

A well designed system should work in the waterfall region, in order to exploit the code at its best, so that the desired bit error probability should lie above the error floor. The position of the error floor depends on several
characteristics of the turbo code design, like the constituent encoders, the interleaver, the code length, etc. In general, however, the structure of classical turbo codes, i.e., that of a parallel concatenation of convolutional interleavers (PCCC, parallel concatenated convolutional code), makes it difficult to reach very low error floors.

A solution to lower the error floor lies in the so called serial concatenation of convolutional codes (SCCC), in which the two (or \( n > 2 \)) constituent encoders are in a cascade separated by one (or \( n - 1 \)) interleaver(s). In [10] a detailed analysis of the SCCCs, together with the description of the iterative SISO decoding algorithm, is presented. The configuration of the new structure makes it suitable also for applications different from channel encoders, encompassing all serial concatenations of blocks in a digital receiver, like turbo equalization [11], turbo carrier synchronization [12], turbo multiuser detection [13], and many others.

Applied to binary encoders and modulation in the first instance, turbo codes have been successively applied to multilevel modulations giving rise to turbo trellis-encoded schemes, extending the amazing performance of binary turbo codes to the field of high bandwidth-efficient systems [14], [15]. For their flexibility in terms of code rate and block length, turbo codes are the ideal candidates for systems that require adaptive coded modulation, capable of changing their bandwidth efficiency according to the channel characteristics [16].

Since their invention in 1993, turbo codes have been widely adopted as standard in several communication systems [17], such as third and fourth generation cellular systems, the communication standard of the Consultative Committee for Space Data Systems (CCSDS), the interaction channel of satellite communication systems, such as DVB-RCS, the IEEE 802.16 (WiMAX) standard, and others.

Section II describes the structure of concatenated codes with interleavers. The general approach will aim at characterizing the constituent elementary blocks that can be used to construct a general concatenated code struc-
ture: convolutional encoders, interleavers, memoryless mappers, serial-to-parallel and parallel-to-serial converters, broadcasters, and more. Each block will be characterized by its input-output relationship. This way, any form of concatenated code with interleaver can be constructed and analyzed, including the original parallel concatenation of two convolutional codes (PCCC) with interleaver known as “tubo code”, the serial concatenation (SCCC), and the hybrid concatenation (HCCC) formed by more than two convolutional codes combined in different fashions. Section III is focused on the maximum-likelihood analysis of turbo codes and the design of their constituent encoders based on the concept of the uniform interleaver, an approach that permits to drastically simplify the performance analysis and optimal design of the convolutional encoders to be used in turbo codes construction. Central to the analysis and design will be the concept of effective free distance of a convolutional code. Section IV is devoted to the iterative decoding algorithms that make affordable in terms of complexity the implementation of turbo decoders. Starting from the soft-input soft-output (SISO) algorithm, the section will describe the input-output relationships of the inverse counterparts of the elementary blocks introduced in Section II, and will show how this makes simple and modular the construction of the iterative decoders for any kind of structure of concatenated codes. Several examples will illustrate the procedure. Section V will describe one of the most challenging design issue in turbo codes construction, i.e., the interleaver design. Following a description of the main properties and parameters of interleavers, and a set of general criteria to be followed, the most important heuristic interleaver designs will be described. Finally, Section VI will show the performance of several forms of code concatenations, including part of the examples already introduced in Sections 2, 4 and 5. The effect of number of iterations, block length, constituent encoders, code structure and interleavers will be analyzed using extensive simulation of the iterative decoding algorithms. Comparison with the ML performance will also be made in some cases.
II. STRUCTURE OF CONCATENATED CODES

A. Main characteristics of turbo encoding structures

Originally, turbo codes were introduced and studied by specifying their encoder structure, i.e., by proposing schemes that explicitly show how the sequence of information symbols is mapped to the sequence of coded symbols. Consequently, in this section we will more precisely talk about turbo encoder structures rather than of turbo codes. This peculiarity of turbo codes is in contrast with the approach used in the literature of low-density parity-check codes (LDPC), where instead the description of the code is done through the parity check matrix, or the corresponding Tanner graph, and thus the encoder structure is left unspecified. As an example, in Figure 3 we report the rate 1/2 encoding scheme originally proposed by Berrou in [1].

In it, the sequence of bits $u$ at the input of the encoder is replicated three times. The first sequence is left uncoded, generating the sequence of systematic bits $x$. The second sequence instead enters a 16-state linear binary convolutional encoder, reported in Figure 4, that generates the sequence of parity-check bits $c_1$. Finally, the third sequence enters a block interleaver of size 65,536. The interleaver reorders the input sequence and its output enters a second convolutional encoder, with a structure identical to the previous one, generating a second sequence of parity check bits $c_2$. The encoded sequence of the rate 1/2 Parallel Concatenated Convolutional Code (PCCC) is finally obtained by concatenating the systematic bits $x$ alternatively with the output of the first and the second encoder, as follows:

$$c = (\cdots, x_k, c_{1,k}, x_{k+1}, c_{2,k+1}, \cdots)$$

As clear from the previous description, another important difference of the turbo encoder structure with respect to that of LDPC codes is its sequence-oriented nature. In block diagrams, input and output, as well as inner connections
between blocks are associated to sequences of symbols rather than to individual symbols. The data rate associated to each sequence is not in general the same for all connections appearing in a block diagram, so that, if necessary, we will add on top of each connection an integer \( r \) representing its symbol rate\(^1\).

The block diagram describing a turbo-encoder is then an oriented graph where each connection represents a sequence, the input and output symbols are clearly visible, and causality is always enforced.

Following the approach described in [18] we will now introduce a set of constituent encoding modules that can be interconnected to form general concatenated encoding structures. This classification of blocks is somewhat arbitrary but allows to easily categorize the main turbo coding structures found in the literature.

A first set of three “data reordering” blocks is characterized by the fact that the output sequences actually contains only symbols taken from the input sequences, although possibly in different order:

1) **The interleaver** \( I \). It performs a permutation of the order of symbols at its input providing an output sequence containing the same symbols in a different order.

2) **The rate conversion blocks** (P/S and S/P). Parallel-to-serial and serial-to-parallel blocks change the symbol rate, so that input and output sequences are not synchronous. Sometimes they are also denoted as multiplexers and demultiplexers.

3) **The puncturer** \( P \). The puncturer is a device that deletes (punctures) some symbols from the input sequence according to a predefined periodic pattern, generating an output sequence with a reduced number of symbols and thus with a lower rate.

A second set of “encoding” blocks is instead characterized by not having the previous property:

1) **The trellis encoder** \( E \). It is a generalization of the binary convolutional encoders of Figure 3. It is characterized by a time-invariant trellis section and represents a mapping between sequences of input symbols and sequences of output symbols.

2) **The Mapper** \( M \). It is characterized by a memoryless correspondence between input and output sequences.

The correspondence introduced by the mapper needs not to be one-to-one.

Using this set of building blocks, the original PCCC scheme of Figure 3 can be represented as in Figure 5.

In the figure one can recognize all blocks introduced in the previous list, the 16-state binary convolutional encoders being examples of trellis encoders \((E)\) with binary input symbols and binary output symbols. In this scheme the repetition of the input sequence into three identical sequences can be associated to a particular type of memoryless mapping, the repetition block.

The mechanism of generating the output sequence of the encoder \( c \) from the sequences \( x, c_1, c_2 \) of Figure 3 can be conveniently represented as the cascade of a parallel to serial converter followed by a suitable puncturer.

In the following subsections we will describe in more detail the introduced constituent encoding blocks.

---

\(^1\)The symbol rate should be distinguished from the common notion of code rate, which is given by the ratio of input symbol rate and output symbol rate, when both input and output symbols belong to the same alphabet. In the following, we will use the capital letter “\( R \)” to denote code rates.
The general setting described here allows to represent turbo coding structures for sequences of symbols belonging to arbitrary alphabets. This allows to deal with scenarios where, for example, non binary channel inputs are used, like in coded modulation applications. The majority of turbo codes constructions found in the literature and in practical applications, however, make use of linear and binary encoder blocks. These two additional characteristics have only a marginal impact on the complexity of encoding and decoding, so they are not assumed from the beginning. Due to their practical importance, however, we will make often an explicit reference to this particular case.

B. Trellis encoders

Trellis encoders (see Figure 6) are the main encoding modules for all turbo coding structures. They are encoders with memory and their dynamical behavior is generally described through a trellis section.

A trellis encoder is described by the following quantities:

- \( u = (u_k) \) is the sequence of input symbols, drawn from a common alphabet \( \mathcal{U} \)
- \( c = (c_n) \) is the sequence of coded symbols, drawn from the common alphabet \( \mathcal{C} \).

The ratio between input and output symbol rates is a rational number \( k_0/n_0 \).

The behavior of a trellis encoder is specified by providing the equations describing its underlying finite-state-machine:

\[
\tilde{c}_i = \tilde{c}(s_{i-1}, \bar{u}_i) \tag{2}
\]

\[
s_i = s^E(s_{i-1}, \bar{u}_i). \tag{3}
\]
These equations specify at time $i$ the next state $s_i$ and the output label $\bar{c}_i$ for each possible configuration of the starting state $s_{i-1}$ and input label $\bar{u}_i$. In our setting, input labels consist of $k_0$-tuples of consecutive symbols taken from the input sequence, i.e., $\bar{u}_i = (u_{ik_0}, \ldots, u_{(i+1)k_0-1})$, while the output labels $\bar{c}$ consist of $n_0$-tuples of consecutive symbols of the output sequence $\bar{c}_i = (c_{in_0}, \ldots, c_{(i+1)n_0-1})$.

A trellis section is characterized by:
- a pair $(k_0, n_0)$ that specifies the length of the input and output labels
- a set of $N$ states $S$
- a set of $N \cdot |U|^{k_0}$ edges $e = (s^E, \bar{u})$ representing all possible arguments of the functions $s^E$ and $\bar{c}$ in (2) and (3)
- the two functions $\bar{c}(e)$ and $s^E(e)$ that for each edge return the output label (2) and the ending state (3), respectively. Being the trellis section assumed to be time-invariant, the set of starting states in a trellis section is equal to the set of ending states.

A trellis section is represented pictorially as in Figure 7. The starting states are represented with the circles on the left, while the ending states are represented with the circles on the right. An arrow is associated to each edge $e$. The arrow starts from the starting state $s^S(e)$ and ends into the ending state $s^E(e)$. On top of the arrow the input label $\bar{u}(e)$ and output label $\bar{c}(e)$ associated to the edge $e$ are also reported to complete the trellis description.

To clarify the meaning of these notations, which will also be used in the description of the decoder counterpart in Section IV, we report in Table I the trellis section of the linear binary convolutional encoder of Figure 8, while in Figure 9 we show its equivalent pictorial representation. In order to make the trellis section more readable, the input and output labels $\bar{u}(e)/\bar{c}(e)$ are listed to the left of the corresponding starting state, and the labels are associated to the edges from left to right and from top to bottom.

C. Mapper

Another important module used in turbo code structures is the memoryless mapper (see Figure 10). A Mapper maps the sequences $\mathbf{u} = (u_1, \ldots, u_k)$ whose symbols belong to the alphabet $\mathcal{U} = \mathcal{U}_1 \times \cdots \times \mathcal{U}_k$ into the sequences
Fig. 8. Rate 1/2 recursive linear binary convolutional encoder. The connection polynomial are $(1, 5/7)$.

<table>
<thead>
<tr>
<th>edge $e$</th>
<th>$s^S(e)$</th>
<th>$\bar{u}(e)$</th>
<th>$\bar{c}(e)$</th>
<th>$s^P(e)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>01</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>00</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>01</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>10</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

TABLE I

TRELLIS SECTION FOR THE LINEAR BINARY ENCODER OF FIGURE 8, $k_0=1$, $n_0 = 2$

$c = (c_1, \ldots, c_n)$ with symbols belonging to the alphabet $C = C_1 \times \cdots \times C_n$. The mapping is performed according to a memoryless function, i.e.,, symbol by symbol:

$$c(u) = \begin{cases} 
  c_1(u) & = c_1(u_1, \ldots, u_k) \\
  \vdots & \\
  c_n(u) & = c_n(u_1, \ldots, u_k). \end{cases}$$

Symbol rates of input and output sequences are thus the same. We do not assume the mapping to be one-to-one so that in general the inverse mapping is not defined. Indeed, while for classical encoders the one-to-one relationship between input and output is required for unique decoding, in code concatenations this constraint must be satisfied by the encoder structure as a whole but not necessarily by all its constituent blocks.

In the following we describe some examples of mappers found in concatenated structures.

1) Repeater: An important special case of mapper is the repeater, that replicates its single input $n$ times at its output:

$$c(u) = (u, \ldots, u)$$

The block is usually represented in block diagrams as a line that splits into $n$ lines, as represented in Figure 11.
Fig. 9. The pictorial representation of the trellis section for the encoder of Figure 8.

Fig. 10. A memoryless mapper $\mathcal{M}$

2) Parity-check bit generator: Another important special case of mapper is the parity-check bit generator (Figure 12), which produces an output bit that is the mod-2 sum of $n$ incoming bits

$$c(u) = \bigoplus_{k=1}^{n} u_k$$

3) Constellation labeling: In practical systems, a mapper can also correspond, with $n = 1$, to the mapper that precedes the modulator, which, in turn, maps $k = m$ multiplexed bits from a binary encoder into a signal set of $M = 2^m$ constellation points. We define a natural one-to-one mapper as follows:

$$c = \sum_{i=1}^{k} u_i N_i$$

$$N_i \triangleq \prod_{j=1}^{i-1} |U_j|$$

with $N_1 = 1$ and $|\mathcal{C}| = N_{k+1}$.

D. Interleaver

The interleaver (Figure 13) is a device with one input sequence and one output sequence with symbols belonging to the same alphabet and with the same rate. It is characterized by a one-to-one mapping between integers $\rho : \mathbb{Z} \rightarrow \mathbb{Z}$.
so that

\[ c_k = u_{\rho(k)}. \]

If causality is enforced, we must have \( \rho(k) \leq k \) for all \( k \).

Turbo-code structures make always use of interleavers between the constituent encoders. The presence of the interleaver between encoding blocks is necessary for two reasons:

1) The interleaver introduces a large memory in the turbo encoder structure, which, in itself, is characterized by constituent encoders having usually small memories. The need for large memory encoders is a necessary condition for designing capacity-achieving channel codes as dictated by the channel coding theorem. In Section III we will see how the interleaver size impacts on the performance of turbo encoding structures.

2) Iterative decoders, as described in Section IV, are based on the fundamental assumption that the sequence of input soft messages processed by each decoder module can be treated as independent. On the other hand, the messages in the sequence computed by the SISO decoder modules are typically not independent, due to the correlation induced by the underlying encoder. A properly designed interleaver ensures that the independence assumption is verified, or at least closely approached.

Interleavers can be divided into two main categories, block interleavers and convolutional interleavers. For block interleavers the permutation \( \rho \) can be represented as a sequence of block permutations, while for convolutional interleavers this is not possible. Note that the delay block falls in this definition of interleaver with \( \rho(k) = k - D \).

The type of mapping \( \rho \) is crucial for the ML performance of turbo encoders and also for the performance of the associated iterative decoder. The design criteria for \( \rho \) and several examples of interleavers used in practice are provided in Section V.
E. Rate conversion modules

Rate conversion modules (Figure 14) are used to split a sequence of symbols into several sequences with lower symbol rate or, on the opposite, to merge several sequences into a single one with higher symbol rate. The constraint on these blocks is that input and output sequences have symbols belonging to the same alphabet.

The sequences can have different symbol rates, so that we add on top of each connection an integer number \( r_i \) representing its rate. The output of a parallel-to-serial converter has a rate \( r_0 \) which is the sum of the rates of incoming sequences. The ordering convention is from top to bottom so that

\[
\begin{align*}
    c_{kr_0+i} &= \begin{cases} 
        u_{1,kr_1+i}, & 0 \leq i < r_1 \\
        u_{2,kr_2+i}, & r_1 \leq i < r_1 + r_2 \\
        \vdots 
    \end{cases} 
\end{align*}
\]

On the other hand, the input of a serial-to-parallel converter has a rate \( r_0 \) which is the sum of the rate of outgoing sequences.

F. Puncturer

The puncturer is a module with one input sequence and one output sequence, with symbols belonging to the same alphabet. The module punctures some of the input symbols generating an output sequence with a lower rate. A puncturer is usually described through a “puncturing pattern” \( P \), which is a binary vector of length \( k \) and Hamming weight \( n \). A zero in the puncturing pattern means that the input symbol is deleted, so that for each \( k \) input symbols, \( n \leq k \) output symbols are generated. Alternatively, one can describe the puncturing mechanism through a vector \( A \) of size \( n \leq k \) with integers \( A_i \in [0, \ldots, k-1] \), so that the input-output relationship can be written as

\[
c_{jn+i} = u_{jk+A_i}, \forall j \in \mathbb{Z}, i \in [0, n-1].
\]
Puncturers are not invertible.

**G. Summary of encoding modules**

In Table II we report a summary of the main characteristics of the introduced modules. For each module we provide input and output alphabets, the rate of input and output sequences, and the memory property.

**H. Some turbo-encoder structures and their main properties**

Having described the ingredients to construct turbo encoding structures, we will describe in the following some of the most common structures found in literature. We refer the reader to Sections III and VI for their performance. Due to the abundance of code concatenations that have been proposed in the literature, we consider here only those embedding at most two codes. A list of other turbo-coding structures and reference to the relevant literature is given at the end of the section.

1) **Serially Concatenated Convolutional Codes**: In a serially concatenated coding scheme (SCCC) (Figure 16) the trellis encoders are connected in series so that the output of the first “outer” encoder is connected to the input of the second “inner” encoder through an interleaver. This is the oldest and most straightforward way of concatenating two encoding modules, although its iterative decoder structure has been introduced only after that of PCCC. Serial concatenation is the basic structure of several variants found in the literature. In particular, the inner encoder can be a trellis coded modulation scheme (SCTCM), a continuous-phase modulator (SCCPM), or the channel itself, considered as a special case of trellis encoder (turbo equalization schemes).
2) Duobinary PCCC: As we will see in Sections III and IV the original parallel concatenated structures suffer from a rather high error floor, due to the slow growth of the minimum distance with the interleaver length. In the attempt to improve the interleaving gain, the inventor of PCCC [1] proposed a modification of it, named duobinary PCCC, whose block diagram is reported in Figure 17. In this scheme not all connections between blocks are associated to binary symbols, so that we add below each connection the cardinality of the associated alphabet and above it the symbol rate.

The name duobinary derives from the fact that in this scheme the input sequence is managed by the encoder as a pair of bits, and thus treated as a sequence of quaternary symbols \((00, 01, 10, 11)\). The encoder is systematic, so that the first output sequence corresponds to the input binary sequence. The binary sequence is then converted into a quaternary sequence (serial to parallel conversion and natural mapper). This sequence enters a classical PCCC encoder structure, where however the input symbols are quaternary. The two trellis encoders, which are classical linear and binary convolutional encoders with rate \(2/1\), are represented in this setting as trellis encoders with rate \(r = k_0/n_0 = 1\) but with quaternary input symbols and binary output symbols. This implies important differences in the associated iterative decoder. The total rate of the encoder is then \(R_c = 2/4 = 1/2\).

3) (Irregular) Repeat accumulate codes: The repeat accumulate (RA) encoder (Figure 18) is an interesting encoding structure because it can be interpreted both as a special case of serial concatenation and as a special case of LDPC code.

The RA is a serial concatenated encoder where the outer code is a repetition code and the inner code is an accumulator, i.e., a two-state binary recursive convolutional encoder. Also in this case the encoder is systematic, and its rate can be selected by changing the number of repetitions \(n\) of the outer code.

In the irregular variant (Figure 19), variable repetitions are considered for each incoming binary digit. This can be obtained by inserting proper puncturers after the repetition block.

The idea of inserting irregularity in the correction capability of the outer code is borrowed from LDPC designs,
where capacity-achieving LDPCs are constructed by adding irregularities to their variable degree distribution.

4) Self concatenation: In Figure 20 we report two encoding structures where a single trellis encoder is used. The output of the trellis encoder (or part of it) is fed back to its input through a proper interleaver. The use of strictly causal devices in the encoding structures must be enforced in order to avoid zero delay loop in the block diagram.

5) Other turbo coding structures:
   • Double SCCC [19]
Hybrid concatenated codes [20], [21]
Multiple PCCC [22]
Multiple repeat accumulate [23], [24].

I. Convolutional versus block encoding

In previous subsections we have introduced a framework for describing turbo encoding structure based on modules that process sequences of symbols. The turbo code structure behaves as a state machine with very large and time-varying trellis [25], and thus like a convolutional encoder. Convolutional encoders are characterized by the fact that their state space never collapses into a single state, so that encoding, as well as decoding, cannot be performed block by block, and actually the notion itself of a “block” is meaningless. This convolutional nature yields better performance but has some drawbacks:

- At the encoder side, it is not suited to packet transmission, as encoding must be done sequentially and one coded frame depends on the final state of the encoder at the end of the previous frame.
- More importantly, decoding cannot be easily parallelized. High-speed iterative decoders can be based only on pipelined architectures. Multiple instances of the same decoder working on different blocks cannot be realized due to the data dependency.

For these reasons “block” turbo encoder are usually adopted in standards and applications.

Turbo encoders can be turned into block encoders by adopting the following measures:

1) All constituent trellis encoders are periodically terminated, so that the state of all encoders is forced to some known state and the encoding of a block does not depend anymore on the previous block
2) Block interleavers must be used instead of the more general convolutional interleavers, so that the permutation acting on sequences can be decomposed into a sequence of permutations on blocks.

The two measures lead to slightly worse performance and decrease the code rate. The difference between convolutional and block interleavers are described in Section V. In the following subsection we will briefly summarize the termination techniques for trellis encoders.

While LDPC codes were originally introduced as block codes, there have been attempts to study variations of them trying to mimic the performance of convolutional codes, like for example the latest spatially coupled LDPC codes [26]. On the other hand, while the original version of turbo code structures were based on convolutional schemes, practical system requirements forced to adopt block-code variants of them.

1) Periodic state enforcing of trellis encoders: A periodic state enforcing technique for a convolutional encoder is a technique that periodically forces (terminates) the encoder state to some predetermined state (known also at the receiver side), independently from the information sequence, so that encoding of a given block of data is independent from the previous one.

There are three basic termination techniques:

1) Reset of the encoder: This technique simply resets periodically the state of the encoder to a predetermined
state (the “zero” state). It should be avoided as it destroys the convolutional code features, decreasing its free distance and its effective free distance.

2) **Trellis termination:** After a fixed number $N$ of trellis sections, in which encoding proceeds normally, additional $\nu$ terminating trellis sections are inserted in the trellis diagram so that the final state is again forced to be the zero state. The inserted trellis sections are subsets of the original trellis section with only one edge leaving each starting state. The edge is not associated to any information symbol, but keeps the same output symbol of the original trellis. The number of inserted trellis sections $\nu$ is such that the zero state can be reached from any starting state. The rate loss due to termination is

$$R_T = \frac{N}{N + \nu} < 1$$

and becomes negligible when $N \rightarrow \infty$. The insertion of the terminating trellis sections allows to preserve the free distance of the code. Trellis termination of a non-recursive convolutional encoder simply requires insertion of $\nu$ zeros at the encoder input. Recursive convolutional encoders require something different, as explained in [27].

3) **Tail-biting.** The tail-biting technique is one that avoids the rate loss introduced by trellis termination. The valid code words of a tail-biting convolutional code are described by all trellis paths that start and end in the same state, not necessarily the zero state. The encoding procedure to achieve this is more complicated and exploits the linearity of the encoder. The advantage of tail-biting is that the rate of the original encoder is not decreased, thus making this technique particularly suited to short block codes, where the rate loss $R_T$ due to termination is non negligible [28].
III. ML ANALYSIS AND DESIGN OF CONSTITUENT CODES

In Section II we have introduced the general concept of concatenated codes with interleaver and described various forms of concatenation, like the parallel, serial and hybrid. In this section we will show how to compute the maximum-likelihood error probabilities for those concatenated codes and how to design the constituent encoders embedded in turbo codes construction.

Throughout the section we will use following assumptions:

1) The binary symbols emitted by the equivalent binary source (source plus source encoder) are independent and identically distributed random variables
2) The encoders are linear and time-invariant block or terminated convolutional encoders
3) The modulation format is 2-PAM with energy $E_b$ associated to each waveform
4) The channel is additive white Gaussian (AWGN) with double-sided power spectral density $N_0/2$
5) The decoder performs maximum-likelihood (ML) complete soft decoding.

A. Maximum likelihood analysis

1) Word and bit error probabilities: With reference to the general definition of code and encoder of Section II (see also Figure 6), we define an $(n, k)$ binary channel code $C$ with rate $R_c = k/n$ as the set of $M = 2^k$ $n$-tuples of bits, the code words $c_i$. Similarly, we define an encoder $E$ as the set of the ordered $2^k$ pairs $(u_i, c_i)$, where $u_i$ is a data word, i.e., a $k$-tuple of information bits, and $c_i$ the corresponding code word. These definitions should clarify the fundamental difference between the notion of a code and the notion of an encoder. The code is a collection of code words and is independent of the way they are obtained. The encoder, instead, refers to the one-to-one correspondence between data words and code words.

Crucial information about an $(n, k)$ linear code is brought by the weight enumerating function (WEF) $A(D)$ of the code, a polynomial in the indeterminate $D$ defined as

$$A(D) \triangleq \sum_{d=0}^{n} A_d D^d$$

where the output coefficient $A_d$ is the number of code words with Hamming weight $d$, obeying the obvious relation $\sum_{d=0}^{n} A_d = 2^k$.

The WEF describes the weight distribution of the code words. The equivalent information on the decoder requires the knowledge of the input-output weight enumerating function (IOWEF), a polynomial in the indeterminates $W$ and $D$ defined as

$$A(W, D) \triangleq \sum_{w=0}^{k} \sum_{d=0}^{n} A_{w,d} W^w D^d$$

where the input-output (I-O) coefficient $A_{w,d}$ is the number of code words with weight $d$ generated by data words with weight $w$. In the following of this section we will also make use of the conditional weight enumerating function (CWEF) $A_w(D)$, defined as

$$A_w(D) \triangleq \sum_{d=0}^{n} A_{w,d} D^d$$
which represents the weight distribution of code words generated by data words with weight \( w \).

Using the union bound [29], the word error probability of a code is upper bounded as

\[
P_w(e) \leq \frac{1}{2} \sum_{d = d_{\text{min}}}^{n} A_d \text{erfc} \left( \sqrt{\frac{d_r E_b}{N_0}} \right)
\]

(7)

showing that the knowledge of the WEF is all we need to upper bound the word error probability, which is then a characteristic of the code.

Finding the bit error probability is somewhat more complicated. First of all we have to define precisely which are the decoder operations, i.e., state if it aims at minimizing the bit, or word, error probability. The second case is by far more manageable, and thus we will evaluate an upper bound to the bit error probability assuming maximum-likelihood decoding at the code word level. With this approach, we must enumerate the actual code word errors, and weight the error probability of each error by the number of data bit errors that occurred.

Applying again the union bound, we obtain the following upper bound to the bit error probability:

\[
P_b \leq \frac{1}{2} \sum_{w=1}^{K} \frac{w}{K} \sum_{d = d_{\text{min}}}^{n} A_w,d \text{erfc} \left( \sqrt{\frac{d_r E_b}{N_0}} \right)
\]

(8)

showing that the knowledge of the I-O coefficients permits to upper bound the bit error probability, a characteristic of the encoder.

2) Uniform Interleaver: Previous subsection has clarified the central role played by the I-O weight enumerating function or, equivalently, I-O coefficients in computing upper bounds to the bit error probability. As a consequence, the first (and main) problem to be solved is that of computing those coefficients for concatenated codes with interleavers. In doing this, we will need to introduce the concept of the uniform interleaver, which will permit to simplify the coefficients computation, and to evaluate the average performance of the coding scheme. The uniform interleaver will be applied to the case of block and terminated convolutional code concatenations.

A uniform interleaver with size \( K \) is a probabilistic device that maps a given input word of \( K \) bits and weight \( w \) into all distinct \( \binom{K}{w} \) permutations of it with equal probability \( 1/\binom{K}{w} \) (see an example in Figure 21).

3) Analysis of PCCC: The block diagram of a parallel concatenated convolutional code (PCCC) using terminated convolutional codes is shown in Figure 22. A block \( \mathbf{u} \) of \( K \) data bits enter the systematic upper convolutional encoder to generate a code word \( \mathbf{c}_u = (\mathbf{u}, \mathbf{c}_1) \) of length \( K + N_1 \), where \( \mathbf{c}_1 \) are the redundant bits including those generated by trellis termination. Then, the \( K \) input bits are interleaved and passed to the systematic lower convolutional encoder that generates a word \( \mathbf{c}_2 \) of \( N_2 \) redundant bits including the coded bits generated by trellis termination. The PCCC is formed by concatenating in some way the two words \( \mathbf{c}_u \) and \( \mathbf{c}_2 \) obtaining the PCCC code word with length \( N = K + N_1 + N_2 \).

Our aim is to derive the I-O coefficients \( A^{(p)}_{w,d} \), whose meaning is the number of code words of the PCCC with weight \( d \) generated by data words with weight \( w \). Obviously, we assume knowledge of the I-O coefficients \( A^{(u)}_{w,d} \) and \( A^{(l)}_{w,d} \) of the two constituent encoders.

To compute the I-O coefficients of the PCCC, we should take each \( K \)-bit long data word of weight \( w \), encode it by the upper encoder \( \mathcal{E}_u \) and store the obtained weight \( d_u \). Then, the data word should be passed through the
interleaver and encoded by the lower encoder $E_l$ to yield the second weight $d_2$. Finally, the code word weight is obtained as $d = d_u + d_2$. Unfortunately, the code word of the lower encoder (and its weight) will depend not only on the weight $w$ of the data word, but also on the permutation induced by the interleaver, so that the previous operations should be repeated for all $\binom{K}{w}$ data words with weight $w$, and for all weights $w = 1, \ldots, K$. In performing this operation, the knowledge of the I-O coefficients of the constituent encoders cannot be exploited, so that the computational complexity for large $K$ becomes overwhelming.

To overcome this difficulty, we replace the actual interleaver with the uniform interleaver, and compute the I-O coefficients $A_{w,d}^{(p)}$ averaged with respect to all $K!$ interleavers of size $K$ acting on the weight $w$ data word:

$$
A_{w,d}^{(p)} \triangleq \mathbb{E}_I A_{w,d}^{(p)}(I) = \frac{1}{K!} \sum_I A_{w,d}^{(p)}(I) P[I] = \frac{1}{K!} \sum_I A_{w,d}^{(p)}(I)
$$

Since in this section the I-O coefficients will always have this “average” meaning, we use the notation $A_{w,d}^{(p)}$ for the average I-O coefficient, and $A_{w,d}^{(p)}(I)$ for the I-O coefficient of a PCBC employing the particular interleaver $I$. 

---

Fig. 21. An example of uniform interleaver

Fig. 22. Parallel concatenated convolutional code
where \( P[I] \) is the probability of choosing the particular actual interleaver \( I \), i.e., \( 1/(K!) \) assuming that all interleavers are equally likely. To evaluate the average I-O coefficient, let us first compute it for a given data word \( u_i \) with weight \( w \) at the input of the upper encoder, and then sum over the distinct \((K\,w)\) data words with weight \( w \)

\[
A_{w,d}^{(p)} = \sum_{I} \frac{1}{K!} \sum_{i} A_{w,d}^{(p)}(I, u_i)
\]  

(10)

Denoting by \( d_{u,i} \) the Hamming weight of the code word generated by the upper encoder in correspondence of the data word \( u_i \), we obtain

\[
A_{w,d}^{(p)} = \frac{1}{K!} \sum_{i} \sum_{I} A_{w,d}^{(p)}(I, u_i) = \frac{(K-w)!w!}{K!} \sum_{i} A_{w,d-d_{u,i}}^{(l)}
\]  

(11)

where the last equality comes from two facts: first, a number \((K-w)!w!\) of the \( K! \) interleavers, i.e., those acting separately on the zeros and ones of the data word, leave unchanged the data word, and second, the sum over all distinct interleavers produces all permutations of \( u_i \) at the input of the lower encoder, and thus generates by definition \( A_{w,d-d_{u,i}}^{(l)} \) code words with the weight \( d - d_{u,i} \) needed to yield a weight \( d \) for the PCCC code word. Finally, grouping together the \( A_{w,d}^{(u)} \) code words of the upper code with weight \( d_{u} \) and summing over the weights, yields

\[
A_{w,d}^{(p)} = \frac{1}{(K\,w)} \sum_{d_{u}=0}^{\frac{K}{R_{u}}} A_{w,d_{u}}^{(u)} A_{w,d-d_{u}}^{(l)}
\]  

(12)

where \( R_{u} \) is the rate of the upper code. Equation (12) is the desired result. Considering now that the conditional weight enumerating function (see (6)) is the \( D \) transform of the I-O coefficients, and that a convolution like the one in (12) for the coefficient domain becomes a product in the transform domain, we obtain

\[
A_{w}^{(p)}(D) = A_{w}^{(u)}(D) A_{w}^{(l)}(D)
\]  

(13)

Equation (13) shows that the average over all interleavers makes the CWEFs of the two encoders independent, so that the CWEF of the PCBC becomes simply the normalized product of the two.

The introduction of the uniform interleaver permits an easy derivation of the weight enumerating functions of the PCCC. However, in practice, one is confronted with deterministic interleavers, giving rise to one particular permutation of the input bits. So, what is the practical significance of the results obtained with the uniform interleaver? The answer to this question comes from the averaging effect of the uniform interleaver: the performance obtained through it represents those of the class of PCCC employing the same constituent encoders averaged with respect to all possible interleavers. As a consequence, there will be at least one deterministic interleaver yielding performance as good as those obtained by the uniform interleaver. This statement recalls of the conclusion coming from the Shannon coding theorem. The difference, though, is that in our case we will see that finding a deterministic interleaver approaching closely the performance of the uniform interleaver, unlike the problem of finding a code approaching the Shannon limit, is very easy, like generating a purely random permutation of \( K \) integers.
4) Analysis of SCCC: Applying the previous analysis to the case of a serially concatenated convolutional code (SCCC) employing terminated convolutional codes (see Figure 23) we obtain the following result for the CWEFs of the SCCC:

\[ A_w(d) = \sum_{j=0}^{N} \frac{A_w(j) \times A_i(j)}{(N)} \]  

where \( A_w(j) \) and \( A_i(j) \) are the I-O coefficients of the outer and inner encoders, respectively, and \( N \) is the interleaver size.

5) Analysis of hybrid concatenated codes with interleavers: Hybrid concatenated codes with interleavers are a generalization of the parallel and serial structures, involving more than just simply two CCs and one interleaver. We consider here the hybrid concatenation of block (or terminated convolutional) codes (HCBC) with interleavers shown in Figure 24 (a case that encompasses also the use of terminated convolutional codes), in which an upper encoder is concatenated in parallel through an interleaver with a serially concatenated block encoder. The HCBC then employs three CCs and two interleavers. The size \( K \) of the fist interleaver must be an integer multiple of both \( k_u \) and \( k_o \), and the size \( L \) of the second interleaver must be an integer multiple of the minimum common multiple of \( n_o \) and \( k_i \). Denoting by \( R_u, R_o, R_i \) the rates of the upper, outer, and inner encoders, respectively, the rate \( R_h \) of the HCBC becomes

\[ R_h \triangleq \frac{K}{N} = \frac{K}{L/R_i + K/R_u} = \frac{R_u R_o R_i}{R_u + r_o R_i} \]
The union upper bound to the bit error probability for the HCBC of Figure 24 can be easily obtained in the form

\[ P_b \leq \sum_{w=1}^{K} \frac{w}{2K} \sum_{d=d_{\text{min}}^{(h)}}^{d_{\text{max}}^{(h)}} A_{w,d}^{(h)} \text{erfc} \left( \sqrt{\frac{dR_b E_b}{N_0}} \right) \]  

(16)

where \(d_{\text{min}}^{(h)}\) is the minimum distance of the HCBC, and \(A_{w,d}^{(h)}\) are the I-O coefficients of the HCBC, i.e., the number of code words of the HCBC with weight \(d\) generated by weight \(w\) information words.

Equation (16) shows that, in order to apply the upper bounds to the bit and word error probabilities for the HCBC, we only need to compute the I-O coefficients \(A_{w,d}^{(h)}\).

For large values of \(K\), the computation of \(A_{w,d}^{(h)}\) for a given interleaver is an almost impossible task, so that we resort once again to the concept of uniform interleaver applied to both first and second interleavers. This permits an easy computation of the input-output coefficients averaged over all possible pairs of the two interleavers.

According to the properties of the uniform interleaver, the first interleaver transforms input data of weight \(w\) at the input of the outer encoder into all distinct \(\binom{K}{w}\) permutations at the input of the upper encoder. Similarly, the second interleaver transforms a code word of weight \(m\) at the output of the outer encoder into all distinct \(\binom{L}{m}\) permutations at the input of the inner encoder. As a consequence, all input data words with weight \(w\), through the action of the first uniform interleaver, enter the upper encoder generating the same \(\binom{K}{w}\) code words of the upper code, and all code words of the outer code with weight \(m\), through the action of the second uniform interleaver, enter the inner encoder generating the same \(\binom{L}{m}\) code words of the inner code. Thus, the expression for the input-output coefficients of the HCBC becomes

\[ A_{w,d}^{(h)} = \sum_{d_u=d_{\text{um}}}^{K/r_u} \sum_{d_m=d_{\text{om}}}^{L} A_{w,d_u}^{(a)} \times A_{w,m}^{(a)} \times A_{m,d_m-d_u}^{(i)} \frac{\binom{K}{d_u}}{\binom{L}{m}} \]  

(17)

where \(d_{\text{um}}\) and \(d_{\text{om}}\) are the minimum distances of the upper and outer code, respectively.

6) More refined upper bounds: The upper bounds presented in previous subsection were based on the union bound, which is known to become loose and almost useless for signal-to-noise ratios below the channel cutoff rate. Various bounding techniques have been proposed in the literature to overcome the limitations of the union bound. All of them make use of the concept of uniform interleaver.

An upper bound on the block and bit error probabilities of turbo codes with ML decoding is derived in [30], using a modified version of Gallagers bound [31] rather than the standard union bound. This result is a generalization of the transfer function bounds providing a tighter upper bound as compared to the union bound. It requires the partition of the code to constant weight subcodes, such that each one of them includes code words that have also the same information weight. Then, the improved upper bound is applied on each subcode, and finally the union bound is applied to get an upper bound on the bit error probability of the overall code. The bound in [30] is useful for some range below the channel cutoff rate and it does not diverge at the cutoff rate like the union bound. Typically, the upper bound on the block error probability is a tight bound for \(E_b/N_0\) values 0.50.7 dB below the \(E_b/N_0\) value that corresponds to cutoff rate.

In [32] an upper bound on the block error probability for an arbitrary binary-input symmetric channel is presented.
This upper bound, based again on Gallagers technique, is examined for the binary-input AWGN channel and some parallel concatenated turbo codes.

An extended bounding technique based on Gallagers bound is reported in [33], where the basic idea is based on applying a modified version of the tangential sphere bound [34], without any need to partition the code to subcodes, and hence without the need to employ a union bound over subcodes as in [30]. The bounding technique in [33] is applied to different concatenated structures, and, when compared to that in [30], it is shown to extend further the region of $E_b/N_0$ for which the bounds are useful.

In Figure 25, borrowed from [33], the upper bounding technique presented in [33] are compared to the union bound for an SCCC with rate 1/4 employing two convolutional encoders with code block lengths of 50, 100, 200 and 400. While the curves representing the union bound diverge at the cutoff rate of the channel, those obtained with the bounding technique in [33] never exceeds 1.

B. Design criteria for constituent encoders

In the previous subsection the performance of concatenated codes with interleaver with maximum-likelihood decoding have been evaluated by introducing the concept of uniform interleaver, and some examples have shown
in a preliminary way the dependence of the bit error probabilities on the uniform interleaver size. This chapter exploits the analytical tools previously developed to deal with the design of concatenated codes with interleavers.

The ingredients of concatenated codes with interleavers are two constituent codes (CCs) and one interleaver. For medium-large interleavers, the state complexity [35] of the overall code is so large to prevent from exhaustive searches for good codes. The only way to get some insight into the design of PCCCs and SCCCs is to split it into the separate designs of the CCs and of the interleaver. The tool yielding a separate treatment of the two problems is still the uniform interleaver, which permits to identify the most important performance parameters of the CCs and their consequent design based on a simplified computer search. Using this approach, one first designs the CCs as the optimum ones for the uniform (average) interleaver, and then, having fixed the CCs, chooses a suitable interleaver (the latter being the subject of Section V.

This subsection will be devoted to the identification of the main CC parameters that affect the performance of PCCCs and SCCCs. In doing this, we will also show analytically the dependence of the error probability on the interleaver size. The whole analysis will be based on the union bound and maximum-likelihood decoding, and this poses important limitations to the “optimality” of the design, which is valid only for signal-to-noise ratios above the cutoff rate of the channel. A long practical experience on turbo codes design has shown that these designs are particularly suited for codes aiming at good performance in terms of error floor.

The design criteria for CCs of both PCCCs and SCCCs will be based on the following assumptions, which add to those previously introduced at the beginning of this section:

1) The concatenated codes make use of terminated convolutional codes and of uniform block interleavers
2) The interleaver size is much larger than the memory of the CCs.

Under the previous assumptions, it can be proved, after a lengthy analysis whose details can be found in [8] and [10], that the union upper bound to the word and bit error probabilities for PCCCs and SCCCs employing two convolutional CCs and an interleaver with size $K$ can be written in the form

$$P(e) \leq \frac{1}{2} \sum_{d=d_f}^{N} C_d K^{\alpha(d)} \text{erfc} \left( \sqrt{\frac{dR E_b}{N_0}} \right)$$

where $d_f$ is the free distance of the concatenated code, $R$ is its rate, $N$ is the code words block length, $C_d$ is a coefficient that depends on the weight $d$ of the code words but not on the interleaver size $K$, and $\alpha(d)$ is an integer that depends on the CCs and on the kind of concatenation, parallel or serial. Equation (18) shows that, for large interleavers, the error probability is dominated by the term $K^{\text{max}(\alpha)}$, and an interleaver gain is possible only if $\text{max}(\alpha)$ is a negative number. In the following, we will discuss the behavior of $\alpha(d)$ for PCCCs and SCCCs.

1) Design of parallel concatenated convolutional codes with interleaver: With reference to (18), it has been proved in [8] that for PCCCs $\text{max}[\alpha(d)] \leq -1$ if and only if both CCs are recursive convolutional encoders. Moreover, $\text{max}[\alpha(d_{pf, eff})] = -1$, where $d_{pf, eff}$ is the effective free distance of the PCCC.

For very large values of $K$ the summation in (18) is dominated by the term in $d$ that corresponds to $\text{max}[\alpha(d)]$, November 27, 2013 DRAFT
so that, for both recursive CCs, the bit error probability of a PCCC can be written as

\[ P_b(e) \simeq \frac{1}{2} C_{pf, eff} K^{-1} \text{erfc} \left( \sqrt{\frac{d_{pf, eff} R_p E_b}{N_0}} \right) \]  

(19)

where \( C_{pf, eff} \) is a coefficient independent from \( K \) that decreases with the number of nearest neighbors of CCs with weight equal to the effective free distance.

Since the effective free distance of the PCCC is the sum of the effective free distances of the CCs, we can finally list the design criteria for the CCs of a PCCC:

- The CCs must be recursive convolutional encoders
- The parameters of the CCs to be maximized are their effective free distance
- The parameters to be minimized are the number of nearest neighbors with weight equal to the effective free distance.

An example will help clarifying the design considerations.

**Example 1**

Consider two PCCCs using uniform interleavers with the same size \( K \) and two different 4-state systematic recursive convolutional CCs, identified as CC1 and CC2. The generator matrices are

\[ G_1 = \begin{bmatrix} 1 & 1 + Z + Z^2 \\ 1 + Z^2 & 1 \end{bmatrix} \]

for CC1 and

\[ G_2 = \begin{bmatrix} 1 & 1 + Z^2 \\ 1 + Z + Z^2 \end{bmatrix} \]

for CC2.

The two encoders generate equivalent codes, and have the same free distance of 5.

We embed CC1 and CC2 into two PCCCs (denoted by PCCC1 and PCCC2). For CC1 the weight-2 information sequence \( 1 + Z^2 \) generates a code sequence \( [1 + Z^2, 1 + Z + Z^2] \) with weight 5, which in turn leads to the PCCC1 sequence \( [1 + Z^2, 1 + Z + Z^2, 1 + Z + Z^2] \) with \( d_{pf, eff} = 8 \). On the other hand, for CC2 the information sequence \( 1 + Z^3 \) generates a code sequence \( [1 + Z^2, 1 + Z + Z^2 + Z^3] \) with weight 6, and this leads to the PCCC2 sequence \( [1 + Z^2, 1 + Z + Z^2 + Z^3, 1 + Z + Z^2 + Z^3] \) with \( d_{pf, eff} = 10 \).

Therefore, the effective free distance for PCCC2 is 10, as opposed to 8 for the PCCC1. Note also that \( d_{pf} = 8 \) for PCCC1, whereas for PCCC2 the free distance is obtained from the information sequence \( 1 + Z + Z^2 \), which generates the code sequence \( [1 + Z^2, 1 + Z^2, 1 + Z^2] \) with weight \( d_{pf} = 7 \). We have then two PCCCs: one (PCCC1) would be preferred according to the design criteria for standard convolutional codes, since its free distance is larger, whereas the second (PCCC2) should be chosen according to the design criteria previously developed.

These conjectures will be supported now by analytical results based on the evaluation of (18). They are reported in Figure 26, where we plot the bit error probabilities for PCCC1 and PCCC2 for interleaver sizes \( K = 30, 100 \) and 1,000. We see that the improvement yielded by PCCC2 over PCCC1 increases progressively with increasing \( K \), and that the performance curves would cross only for very large values of the signal-to-noise ratios, due to the lower free distance of PCCC2.

The following example will show the crucial role played by the recursiveness of CCs.
Example 2
Consider two rate 1/3 PCCCs. The first, PCCC1, is based on two 2-state, systematic non recursive convolutional encoders with generator matrices

\[ G_{11} = [1, 1 + Z] \]

for CC11, with rate 1/2, and

\[ G_{12} = [1 + Z] \]

for CC12, with rate 1. The second, PCCC2, is based on two 2-state, systematic recursive convolutional encoders with generator matrices

\[ G_{21} = \begin{bmatrix} 1, \frac{1}{1 + Z} \end{bmatrix} \]

for CC21, with rate 1/2, and

\[ G_{22} = \begin{bmatrix} \frac{1}{1 + Z} \end{bmatrix} \]

for CC22, with rate 1. The two encoders CC11 and CC21 generate the same codes, but have different I-O coefficients, and thus different bit error probabilities, as shown in Figure 27.

As the figure shows, the non recursive encoder CC11 yields better performance for \( E_b/N_0 \) greater than 1 dB.

Consider now the performance of the two PCCCs based on feed-forward and recursive CCs. Using the bounding technique previously described we have computed the bit error probabilities for PCCC1 and PCCC2. The results are reported in Figure 28 for different interleaver sizes.

The curve “A” corresponds to the uncoded binary PSK which is reported for reference. Curves “B” and “C” refer to PCCC1: curve “B” represents the performance of PCCC1 obtained by simply duplicating the redundant bit of the CC, while curve “C” derives from the use of an interleaver with size 1,000. The results show that the difference with \( K \) are marginal (less than half a dB), and limited to a short range of error probabilities. In fact, the curves practically merge below \( 10^{-7} \).

A completely different behavior is offered by the curves “D” and “E”, referring to the same situations as previous curves “B” and “C” for PCCC2. Here, in fact, we notice a significant improvement for \( K = 1,000 \), yielding a gain of 3 dB at \( 10^{-5} \). Interestingly enough, for \( K = 1 \) (compare curves “B” and “D”) PCCC1 is better than PCCC2. This is due to the fact that...
the same free distance of both rate $1/2$ CCs (recursive and not) is obtained from different contributions of the information and parity bits, so that duplicating the parity bits leads to a larger free distance for PCCC1. The hierarchy is completely reversed for $K = 1,000$.

2) A heuristic explanation of the interleaver gain: We have seen previously that recursive constituent encoders do play a fundamental role in PCCCs, owing to the *interleaving gain* they yield. We provide here a simple analytical explanation of this fact, which illuminates the most important parameters of the CCs in determining the PCCC performance.
Consider a PCCC with an interleaver of size $K$ and two CCs $C_1$ and $C_2$ having the same trellis. For feed-forward CCs, an information sequence with weight 1 will generate an error path in the first encoder, and, after interleaving, an error path also in the second encoder, since the interleaver can only modify the position of the error path but not destroy it. Thus, independently from the size $K$, the ratio between “bad” interleavers, keeping the error path in the second encoder, and the whole class of interleavers, is 1, and no interleaving gain is possible.

On the other hand, information sequences with weight 1 are not harmful for recursive encoders, since the corresponding code sequences have very large weights. The minimum weight in input sequences that generate an error path is 2, with the two “1”s in a particular configuration. After interleaving, that configuration must be kept in order to generate an error path with the same distance also in the second encoder, and this yields a number of “bad” interleavers roughly equal to $K$ (the simplification here consists in neglecting the length of the error path as compared with the interleaver size), i.e., those interleavers that rigidly shift the two 1’s. Then, the fraction of bad interleavers is

$$\frac{K}{(K/2)} \approx 2K^{-1}$$

This fraction can also be interpreted as the probability that a randomly chosen interleaver is able to break the error path in the second encoder, or, similarly, as a factor reducing the bit error probability, namely the interleaver gain.

By extension, considering a data sequence with weight $w \geq 2$, the interleaver gain becomes

$$\frac{K}{(K/w)} \approx w!K^{1-w}$$

so that we can affirm that for large $K$ the most likely weights in PCCC code sequences are, in the order of their probability, $d_2 = d_{of,of}, d_3, \ldots, d_w, \ldots$, where $d_w$ is the minimum weight of code sequences generated by data sequences with weight $w$.

3) Design of serially concatenated convolutional codes with interleavers: With reference to (18), it has been proved in [10] that for SCCC$s$ using a recursive inner encoder the following result holds true:

$$\max[\alpha(d)] = -\left\lfloor \frac{d_{of}+1}{2} \right\rfloor$$

where $d_{of}$ is the free distance of the outer code.

For very large values of $K$ the summation in (18) is dominated by the term in $d$ that corresponds to $\max[\alpha(d)]$, so that, for recursive inner encoders, the bit error probability of an SCCC can be written as

$$P_b \approx \frac{1}{2} C_{even}K^{-d_{of}/2} \text{erfc}\left( \frac{d_{of}d_{i,f,eff}R_sE_b}{2N_0} \right)$$

for even values of $d_{of}$, and

$$P_b \approx \frac{1}{2} C_{odd}L^{-(d_{of}+1)/2} \text{erfc}\left\{ \sqrt{\frac{(d_{of} - 3)d_{i,f,eff} + 2d_{i,3}}{2N_0}} \right\}$$

for odd values of $d_{of}$, where $d_{i,f,eff}$ is the effective free distance of the inner code, $d_{i,3}$ is the minimum weight of sequences of the inner code generated by weight-3 input sequences, $C_{even}$ and $C_{odd}$ are coefficients independent from $K$ that decrease with the number of nearest neighbors of CCs with weight equal to the effective free distance.
In both cases of $d_{of}$ even and odd, we can draw from (21) and (22) a few important design considerations:

- The inner encoder must be a convolutional recursive encoder
- The effective free distance of the inner encoder must be maximized
- The interleaver gain is equal to $L - d_{of}/2$ for even values of $d_{of}$ and to $L - (d_{of} + 1)/2$ for odd values of $d_{of}$. As a consequence, we should choose, compatibly with the desired rate of the SCCC, an outer code with a large and, possibly, odd value of the free distance.
- As to other outer code parameters, the number of input sequences generating free distance error events of the outer code should be minimized.

C. Comparison between parallel and serially concatenated codes

In this section, we will use the bit error probability bounds previously derived to compare the performance of parallel and serially concatenated convolutional codes. To obtain a fair comparison we have chosen the following PCCC and SCCC: the PCCC is the one described in Example 1 as PCCC2, i.e., a rate 1/3 code obtained concatenating two 4-state systematic recursive convolutional encoders. The SCCC is a rate 1/3 code using as outer code the same rate 1/2, 4-state code as in the PCCC, and, as inner encoder, a rate 2/3, 4-state systematic recursive convolutional encoder. Also in this case, the interleaver sizes have been chosen so as to yield the same data block size. The results are reported in Figure 29, where we plot the bit error probability versus $E_b/N_0$ for various data block sizes.

The results show the difference in the interleaver gain. In particular, the PCCC shows an interleaver gain going as $K^{-1}$, whereas the interleaver gain of the SCCC goes as $K^{-\frac{d_{of}+1}{2}} = K^{-3}$, being the free distance of the outer code equal to 5. This means, for $P_b = 10^{-9}$, a gain of 3 dB in favor of the SCCC.
D. Finding the optimum constituent encoders

Before the invention of turbo codes, the convolutional encoders used in all applications were feed-forward, non systematic encoders, optimized by maximizing their free distance. The parameters describing those encoders are reported in coding books like for example [29].

On the other hand, we have proved previously in this chapter that interleaving gain is allowed provided that we use recursive encoders for both CCs in PCCCs, and for the inner CC in SCCCs. Moreover, instead of the free distance, the encoder parameter to be maximized is the effective free distance, i.e., the minimum weight of code words generated by weight 2 data words. This makes the existing tables of best convolutional encoders useless for our purposes, and we need to find new encoders based on different optimization rules. The following theorem (for a proof see [8]) provides an upper bound to the achievable effective free distance of recursive encoders.

**Theorem 1.** The effective free distance of a rate $1/n$ recursive convolutional encoder with memory $\nu$ satisfies the following inequality

$$d_{f,\text{eff}} \leq n(2 + 2^{\nu-1})$$

Equality holds in (23) if and only if the generating matrix $G(Z)$ is of the form

$$G(Z) = \begin{bmatrix} n_1(Z) & \frac{n_2(Z)}{p(Z)} & \ldots & \frac{n_n(Z)}{p(Z)} \end{bmatrix}$$

where $p(Z)$ is a primitive polynomial of degree $\nu$ and $n_i(Z)$ is any monic polynomial of degree $\nu$ different from $p(Z)$.

An extensive search for good systematic, recursive convolutional encoders to be embedded in concatenated codes with interleavers have been performed in [9].
IV. ITERATIVE DECODING

The key idea introduced by the inventors of turbo-codes in [1] was the use of an iterative algorithm to compute the maximum-a-posteriori probability of information symbols, necessary to the maximum-a-posteriori decision on them. The two decoders, associated to each of the CCs, accepts the information sequence related to the soft reliability of the information symbols, upgrades the soft information and, after interleaving, passes the updated sequence to the second decoder. The second decoder, in turns, processes the soft inputs, and, after de-interleaving, returns the sequence to the first decoder. The processes continues for a certain number of iterations until a hard decision can be reliably made on information symbols.

In this section we will start by describing the general conventions and assumptions needed to derive such an iterative decoding algorithm. These assumptions will lead to an iterative decoder structure that can be derived directly from the block diagram of the corresponding concatenated encoder.

A. Messages in iterative decoders and independence assumption

Iterative decoders are constructed using a set of constituent soft-input soft-output (SISO) modules that accept and deliver “messages” or “soft information” about the symbols of the constituent encoders in the turbo structure (see Figure 30).

Iterative decoders for a given turbo encoding structure are obtained by substituting the encoding modules in the encoder block diagram presented in Section II with their corresponding SISO modules. As an example, Figure 31 shows the block diagram of the iterative (turbo) decoder for the parallel concatenated convolutional code (PCCC) of Figure 3.

The fundamental difference between encoder and iterative decoder is that in the encoder the inputs and outputs of each module are clearly identified, and unidirectional arrows connect the constituent modules, while for the corresponding iterative decoder this is not true anymore. SISO modules have bidirectional sequences of messages.
on each side of the block. The presence of these bidirectional connections is a direct consequence of the iterative nature of the decoding algorithm.

Before describing in more details how each SISO module computes the output messages, we will focus in the following on the possible message alternatives that can be used. The choice of the message representation adopted in the iterative decoder is crucial for the memory requirements as well as for the complexity of the implementation.

Consider a random variable \( S \) belonging to a finite alphabet \( S \), representing a generic symbol in the encoding scheme. The likelihood \( L(s) \) associated to it is a function defined on \( S \) that returns positive real quantities. It is physically stored as a vector of \( |S| \) elements. The likelihood is usually associated to the probability of some observation \( Y \), conditioned on the knowledge of the random variable \( S = s \), considered as a function of \( s \):

\[
L(s) = P(Y|S = s).
\]

The fundamental assumption in the construction of the iterative decoder is that likelihoods are always treated as independent variables by SISO modules, in the sense that the likelihood of any sequence of symbols \( (S_1, \ldots, S_n) \)

\[
L(s_1, \ldots, s_n) = P(Y|S_1 = s_1, \ldots, S_n = s_n),
\]

is assumed to be the product of the individual likelihoods

\[
L(s_1, \ldots, s_n) = \prod_{i=1}^{n} L(s_i).
\] (24)

The SISO modules of the iterative decoder compute the updated likelihoods imposing the constraint of the corresponding encoding module, that set to zero the probability of all the not valid input-output configurations\(^3\).

In the most general setting, an encoding module is a mapping between the set of unconstrained inputs \( u \) to the set of coded symbols \( c \), and can be described as a subset \( E \) of all possible input-output configurations. In formulæ

\(^3\)The SISO updating equations are actually a special case of the more general Belief Propagation setting, where the code constraints, which here consist in indicator functions, are substituted with more general joint probabilities specifying the statistical dependence between the variables involved in a given node.
\[ E = \{(u, c) : c = c(u)\} \subseteq \{(u, c)\} = \bigotimes_k U_k \bigotimes_n C_n. \]

Consider now a particular input symbol \( u_i \) to the encoder; its output "total" likelihood \( L_T(u_i; O) \) can be computed as

\[
L_T(u_i; O) = \sum_{(\tilde{u}, \tilde{c}) \in E : \tilde{u}_i = u_i} L(\tilde{u}; I) L(\tilde{c}; I)
\]

\[
= \sum_{(\tilde{u}, \tilde{c}) \in E : \tilde{u}_i = u_i} \prod_k L(\tilde{u}_k; I) \prod_n L(\tilde{c}_n; I),
\]

where the second identity stems from the independence assumption (24).

By definition, the sum in (25) has the common factor \( L(\tilde{u}_i; I) = L(u_i; I) \), which can be extracted from the sum obtaining

\[
L_T(u_i; O) = L(u_i; I) L(u_i; O),
\]

where we have defined the extrinsic likelihood

\[
L(u_i; O) \triangleq \sum_{(\tilde{u}, \tilde{c}) \in E : \tilde{u}_i = u_i, k \neq i} \prod_k L(\tilde{u}_k; I) \prod_n L(\tilde{c}_n; I).
\]

(26)

where in (26) the input likelihood \( L(u_i; I) \) of the symbol at hand is not considered.

In a similar way, considering a generic coded symbol \( c_i \) we get its extrinsic likelihood as

\[
L(c_i; O) \triangleq \sum_{(\tilde{u}, \tilde{c}) \in E : \tilde{c}_i = c_i, k \neq i} \prod_k L(\tilde{u}_k; I) \prod_n L(\tilde{c}_n; I).
\]

(27)

This form of the updating equation for the SISO modules that use likelihoods originates the sum-prod version of the iterative decoder. The use of these updating equations shows some drawbacks as it requires to perform products and can generate numerical problems.

Among the possible alternative message representations, the most used in practice is the Log-Likelihood Ratio (LLR) defined as

\[
\lambda(s) \triangleq \log \left( \frac{L(s)}{L(s_0)} \right)
\]

(28)

where \( s_0 \) is an arbitrary reference symbol, e.g., \( s_0 = 0 \).

The use of LLRs \( \lambda \) instead of the likelihoods \( L \) as messages in the iterative decoder has the following advantages:

- being based on logarithms it is more suited to represent messages with exponential-like distribution
- it converts all products in the updating equations (26) and (27) into simpler sums
- the normalization allows to spare one value in its representation, as \( \lambda(s_0) = 0 \) by definition. Of particular importance is the binary case where the LLR becomes a scalar:

\[
\lambda \triangleq \lambda(1) = \log \left( \frac{L(1)}{L(0)} \right)
\]

- the normalization also solves possible numerical problems deriving from unwanted and unnecessary constants in (27) and (26).
The sum operator in (26)-(27), when using LLRs, is mapped into the \( \max^* \) operator

\[
L_1 + L_2 \rightarrow \max^*(\lambda_1 + \lambda_2) \triangleq \log (e^\lambda_1 + e^\lambda_2)
\]

The \( \max^* \) operator shares the same properties of the addition like commutativity and associativity, so that it is not ambiguous to use the summation notation

\[
\sum_{i=1}^{n} L_i \rightarrow \max^*_{i=1}^{n} \lambda_i = \max^*(\lambda_1, \ldots, \lambda_n).
\]

The \( \max^* \) is a rather simple operator, very similar to the \( \max \) operator and is often used in practical implementations of turbo-decoders.

Due to its importance in practical applications, we restate here the generic updating equations of the SISO module (26) and (27) using the LLR messages

\[
\lambda(u_i; O) = \max^*_{(\tilde{u}, \tilde{c}) \in \mathcal{E}: \tilde{u}_i = u_i} \sum_{k \neq i} \lambda(\tilde{u}_k; I) + \sum_{n} \lambda(\tilde{c}_n; I) - \max^*_{(\tilde{u}, \tilde{c}) \in \mathcal{E}: \tilde{u}_i = 0} \sum_{k \neq i} \lambda(\tilde{u}_k; I) + \sum_{n} \lambda(\tilde{c}_n; I) \quad (29)
\]

\[
\lambda(c_i; O) = \max^*_{(\tilde{u}, \tilde{c}) \in \mathcal{E}: \tilde{c}_i = c_i} \sum_{k} \lambda(\tilde{u}_k; I) + \sum_{n \neq i} \lambda(\tilde{c}_n; I) - \max^*_{(\tilde{u}, \tilde{c}) \in \mathcal{E}: \tilde{c}_i = 0} \sum_{k} \lambda(\tilde{u}_k; I) + \sum_{n \neq i} \lambda(\tilde{c}_n; I) \quad (30)
\]

The brute force computation of (26)-(27) or (29)-(30) requires a number of products and sums (or, equivalently, sums and \( \max^* \)) that is proportional to the number of elements in \( \mathcal{E} \), and thus is practically affordable only for small mappings.

The iterative decoders based on the LLR message representation will be called \( \max^* \)-sum decoders, while the suboptimal ones based on the approximation of the \( \max^* \) with the simpler \( \max \) operator will be called the max-sum decoder. These two versions of the decoder correspond to the sum-prod and min-sum versions of the LDPC decoders.

B. Soft-Input Soft-output modules

Having described the general characteristics of a SISO module to be embedded into an iterative decoder, and the type of messages that are used, we consider in more detail the characteristics of the SISO modules associated to individual encoding modules introduced in Section II.

We will assume to use messages in the form of LLRs, so that we will describe the \( \max^* \)-sum version (29) and (30) of the SISO updating equations for each module. The use of other type of messages requires straightforward replacements of the operators that will be omitted.

C. The SISO for the data ordering encoding modules

As we have explained in Section II, the output sequences for data ordering modules (rate converters, interleavers and puncturers) contain only input symbols. For these modules the SISO bidirectional counterparts reduce to two independent modules. Messages in the SISO modules are not updated but just reordered according to the same rule used in the encoder.
Fig. 32. The SISO module corresponding to an interleaver is a pair of interleaver and deinterleaver blocks

As an example we show in Figure 32 the block diagram of the SISO module corresponding to the interleaver. It consists of a pair of blocks, the first *interleaves* the sequence of messages relative to the input $\lambda(u;I)$ to generate the output sequence of messages $\lambda(c;O)$, while the second *deinterleaves* the sequence of messages relative to the output to generate $\lambda(c;I)$ the input sequence of messages $\lambda(u;O)$.

Similarly, in Figure 33 we show the SISO counterparts of the rate converters, which reduces to a pair of P/S and S/P blocks.

For the puncturer block (Figure 34) the input sequence contains also symbols that are not present in the output sequence. The inverse operation of the puncturer at the decoder side, called "depuncturing", consists in inserting dummy all-zero messages in the deleted positions. All-zero messages in the LLR representation correspond to the uniform distribution and thus maximum uncertainty about the deleted symbols.

**D. The SISO module for the trellis encoder**

The SISO module for the trellis encoder (Figure 35) is a four-port device that accepts at the input the *sequences* of LLR $\lambda_k(u;I)$ and $\lambda_n(c;I)$ and computes the sequences of extrinsic LLR $\lambda_k(u;O)$ and $\lambda_n(c;I)$.

Since for trellis encoders we have assumed that the input and output alphabets are the same for the symbols in the sequence, we assign the subscript denoting the time index to the LLRs $\lambda$, and not to its argument as we did for the general case.

The SISO algorithm described here is a slight generalization of the BCJR algorithm introduced in [4]. This algorithm can be interpreted as an efficient way to compute (29) and (30) with a linear, rather than exponential, complexity in the sequence length by exploiting the trellis diagram representation of the mapping between sequences.

1) **The SISO algorithm for computing the extrinsic LLRs:** We will start assuming that $k_0 = n_0 = 1$, so that we can set $\bar{u} = u$ and $\bar{c} = c$, leaving to the next subsection the more general case. The SISO algorithm is based on
the preliminary computation of the so-called \textit{forward} and \textit{backward} recursions defined as follows:

\begin{align}
\alpha_{i+1}(s) &= \max_{e : s^S(e) = s}^* \alpha_i(s^S(e)) + \lambda_i[u(e); I] + \lambda_i[c(e); I] \\
\beta_{i-1}(s) &= \max_{e : s^S(e) = s}^* \beta_i(s^E(e)) + \lambda_i[u(e); I] + \lambda_i[c(e); I],
\end{align}

where we have used the notations introduced in Section II to describe a trellis section.

The two recursions compute the forward and backward path metrics $\alpha_i(s)$ and $\beta_i(s)$, which represent the log-likelihood of each state in the trellis diagram given the past and future sequences of input LLRs.

The forward recursion is formally identical to that of the Viterbi algorithm, with the difference that the $\max^*$ operator is used instead of the $\max$ operator, and that the metric is formed here by the two contributions relative to both the input and output labels associated to each edge $e$. The backward recursion, instead, proceeds from future to past. Notice that the forward and backward recursions (31) and (32) need to be properly initialized at some points, a problem that can be solved in several ways. We refer the interested reader to [18] for more details.

Based on the forward and backward recursion (31) and (32), the computation of the output extrinsic LLRs is performed as follows:

\begin{align}
\lambda_i(u; O) &= \max_{e : u(e) = u}^* (\alpha_i(s^S(e)) + \lambda_i[c(e); I] + \beta_i(s^E(e))) - \\
& \quad \max_{e : u(e) = 0}^* (\alpha_i(s^S(e)) + \lambda_i[c(e); I] + \beta_i(s^E(e))) \\
\lambda_i(c; O) &= \max_{e : c(e) = c}^* (\alpha_i(s^S(e)) + \lambda_i[u(e); I] + \beta_i(s^E(e))) - \\
& \quad \max_{e : c(e) = 0}^* (\alpha_i(s^S(e)) + \lambda_i[u(e); I] + \beta_i(s^E(e)))
\end{align}

Note that the computation of the output extrinsic information takes into account all the “past” LLRs through the forward path metric $\alpha$, all the future LLRs through the backward path metrics $\beta$, and the present LLRs $\lambda_i$, excluding that of the symbol for which the extrinsic LLR is computed.

From (31), (32) and (33), one can see that the complexity of the SISO algorithm is proportional to the number of edges of the trellis section.
2) Trellis with multiple symbol labels: When the labels of the trellis are composed by multiple input symbols \( \bar{u} = (u_1, \ldots, u_k) \) and \( \bar{c} = (c_1, \ldots, c_n) \), a preliminary step is required to compute the LLRs of the labels \( \bar{u} \) and \( \bar{c} \) starting from the LLRs of their constituent symbols. This can be realized using the independence assumption as

\[
\lambda_i(\bar{u}; I) = \sum_{k=0}^{k_0-1} \lambda_{ik_0+k}(u; I),
\]

\[
\lambda_i(\bar{c}; I) = \sum_{n=0}^{n_0-1} \lambda_{in_0+n}(c; I).
\]

Similarly, at the output one needs to compute the extrinsic LLRs of the constituent symbols rather than that of the trellis labels. This can be achieved by first substituting (33) with the computation of the total information of labels \( \gamma \)

\[
\gamma_i(\bar{u}) = \max_{u: e = \bar{u}} \left( \alpha_i[s^S(e)] + \lambda_i[u(e); I] + \lambda_i[c(e); I] + \beta_i[s^E(e)] \right),
\]

\[
\gamma_i(\bar{c}) = \max_{c: e = \bar{c}} \left( \alpha_i[s^S(e)] + \lambda_i[u(e); I] + \lambda_i[c(e); I] + \beta_i[s^E(e)] \right),
\]

and then marginalize, normalize and subtract the input LLR to get the extrinsic LLRs of the constituent symbols

\[
\lambda_{ik_0+k}(u; O) = \max_{\bar{u}: u_k = u} \gamma_i(\bar{u}) - \max_{\bar{u}: u_k = 0} \gamma_i(\bar{u}) - \lambda_{ik_0+k}(u; I)
\]

\[
\lambda_{in_0+n}(c; O) = \max_{\bar{c}: c_n = c} \gamma_i(\bar{c}) - \max_{\bar{c}: c_n = 0} \gamma_i(\bar{c}) - \lambda_{in_0+n}(c; I).
\]

E. The SISO module for the mapper

The final module is the SISO module for the memoryless mapper defined in Subsection II-C (see Figure 36). This block is rather general and characterized only by the fact of being memoryless. For this reason the updating equations remain those of the general case (29) and (30).

1) Computation of SISO updating with the minimal trellis: Any memoryless mapping, including, for example, all block encoders, can be conveniently represented over a suitably defined trellis diagram, exactly like a convolutional encoder. The main difference w.r.t. the trellis diagram of the convolutional encoder is that in this case the trellis diagram is time varying, i.e., the trellis sections change within the trellis diagram. Moreover, being the mapping memoryless, the set of starting states of the first trellis section and the set of ending states of the last trellis section have cardinality 1.

The trellis diagram corresponds to the mapping in the sense that all valid mapper configurations correspond to a path in the trellis diagram. Furthermore, the SISO algorithm described in section IV-D for the SISO module of
the trellis encoder can be extended to work also on time-varying trellis diagrams by allowing the trellis sections \((s^S(e), u(e), c(e), s^E(e))\) to vary within the trellis diagram \((s^i_S(e), u_i(e), c_i(e), s^i_E(e))\). The complexity of the SISO algorithm is proportional to the total number of edges in the trellis diagram. Thus, the complexity will be minimized by finding, among all possible trellis representations of the considered mapping, the one associated to the minimal number of edges (minimal trellis).

The procedure to find a minimal trellis diagram for a given mapping is not straightforward and is outside the scope of this book. The interested reader can refer to [36] for more details.

Here we consider as examples two very important minimal trellis representations used in practice, \(i.e.,\), the one associated to a generic \((1, n)\) repetition code (Figure 37) and the one associated to its dual, the \((n - 1, n)\) parity-check code (Figure 38).

The trellis diagram of the repetition code is made up with \(n\) trellis sections of three different types. The first
trellis section (A) is associated to one input bit and one output bit, and the next \( n - 1 \) trellis sections (B) are associated to no input bits (1 edge per state) and one output bit. The final trellis section (C) has a single final state.

The total number of paths is 2, equal to the number of codewords. The complexity, normalized to the number of sections is:

\[
C = \frac{\text{number of edges}}{\text{number of trellis sections}} = 2,
\]

and is independent from the code size \( n \).

The trellis diagram of the parity-check code (Figure 38) is also made up with \( n \) trellis sections of three different types. The first trellis section is associated as before to one input bit and one output bit, and the next \( n - 1 \) trellis sections now have 4 edges, and are associated to one input bit and one output bits. The final trellis section has a single final state and no input bit\(^4\). In this case the total number of trellis paths is \( 2^{n-1} \), still equal to the number of code words. The complexity, normalized to the number of input bits is then:

\[
C \propto \frac{4n}{n} = 4.
\]

Also in this case it is independent from the code size, and much smaller than the complexity of the brute force computation of (29) and (30), which is proportional to the number of code words.

**F. Multiple code representations**

We have seen up to now that the definition of a turbo-encoding structure leads naturally to the definition of the corresponding iterative decoding structure. The proposed encoding modules, however, can be used to describe a given encoding structure in several ways.

Consider for example the 4-state linear binary convolutional encoder of Figure 39. We have seen that the whole convolutional encoder can be represented as a trellis encoder using our conventions. On the other hand, the same encoder can be represented at a lower level as the concatenation of mappings and interleavers (delays), as in the right-hand side of the figure.

While from the encoding perspective the two representations are identical the corresponding iterative decoders are different. In particular, the decoder associated to the right-hand side representation will have performances far from the optimal one.

---

\(^4\)Note that the trellis diagram can also be interpreted as that of a terminated two-state convolutional encoder.
Fig. 39. Two representations of the rate 1/2 4-state binary convolutional encoder.
V. INTERLEAVER DESIGNS

A. Interleaver Theory

Interleavers are devices that permute sequences of symbols: they are widely used for improving error correction capabilities of coding schemes over bursty channels [37]. Their basic theory has received a relatively limited attention in the past, apart from some classical papers ([38], [39]). Since the introduction of turbo codes [1], where interleavers play a fundamental role, researchers have dedicated many efforts to the interleaver design. However, the misunderstanding of the basic interleaver theory often causes confusion in turbo code literature.

In this section, interleaver theory is revisited. The intent is two-fold: first, to establish a clear mathematical framework which encompasses old definitions and results on causal interleavers. Second, to extend this theory to non-causal interleavers, which can be useful for turbo codes.

We begin by a proper definition of the key quantities that characterize an interleaver, like its minimum/maximum delay, its characteristic latency and its period. Then, interleaver equivalence and deinterleavers are carefully studied to derive physically realizable interleavers. Connections between interleaver quantities are then explored, especially those concerning latency, a key parameter for applications. Next, the class of convolutional interleavers is considered and block interleavers, which are the basis of most concatenated code schemes, are introduced as a special case. Finally, we describe the classes of interleavers that have been used in practical and standard turbo codes construction.

B. Interleavers: basic definitions

We begin to revisit interleaver theory by introducing some basic definitions. They can be considered an extension of the definitions introduced in [38] for causal interleavers.

An interleaver \( I \) is a device characterized by a fixed permutation of the infinite time axis \( \rho_I : \mathbb{Z} \leftrightarrow \mathbb{Z} \). \( I \) maps bi-infinite input sequences \( \mathbf{x} = (x(i))_{i=-\infty}^{+\infty} \in A^\mathbb{Z} \) into permuted output sequences \( \mathbf{y} = (y(i))_{i=-\infty}^{+\infty} = \rho_I(\mathbf{x}) \in A^\mathbb{Z} \), with \( y(i) = x(\rho_I(i)) \).

Although irrelevant for the interleaver properties, we will assume for simplicity in the following that \( A \) is the binary alphabet \( \mathbb{Z}_2 \).

For an interleaver \((I, \rho_I)\) it is possible to define the following parameters:

**Delay function.**

The delay function \( d_I(i) \) of an interleaver \( I \) is defined as:

\[
d_I(i) = i - \rho_I(i)
\]

The interleaver action over a sequence can also be described through its delay function as follows:

\[
y(i) = x(i - d_I(i))
\]

**Period**

We say that an interleaver \( I \) is periodic with period \( N_I \), if its delay function is periodic, i.e., if there exists a
positive integer $N_I$ such that:

$$d_I(i + N_I) = d_I(i) \forall i \Rightarrow \text{i.e., } \rho_I(i + N_I) = \rho_I(i) + N_I$$

We will only consider periodic interleavers, since non-periodic permutations are not used in practice. For this reason in the following “interleaver” will stand always for “periodic interleaver”. The period $N_I$ is usually referred to, in the turbo code literature, as the interleaver length or size, and is a crucial parameter in determining the code performance, as we have seen in Section III. Often, the period is also directly related to the latency introduced in the transmission chain; this is not strictly correct, as we will see soon that the minimum delay introduced by the interleaver-deinterleaver pair is instead the characteristic latency $D_I$ to be defined shortly. The ambiguity stems from the common suboptimal implementation of a block interleaver (see Subsection V-D4).

In the following, we will describe an interleaver of period $N_I$ only by its values in the fundamental interval $[0, N_I - 1] : \rho_I = (\rho_I(0), ..., \rho_I(N_I - 1))$.

**Maximum delay**

The maximum delay of an interleaver is defined as:

$$d_{I_{\text{max}}} = \max_{0 \leq i < N_I} d_I(i)$$

**Minimum delay**

The minimum delay of an interleaver is defined as:

$$d_{I_{\text{min}}} = \min_{0 \leq i < N_I} d_I(i)$$

**Characteristic latency**

The characteristic latency of an interleaver is defined as:

$$D_I = d_{I_{\text{max}}} - d_{I_{\text{min}}}$$

**Constraint length function**

The constraint length function $\nu_I(i)$ of an interleaver at time $i$ is the number of positions $j$ smaller than $i$ such that $\rho(j) \geq i$ plus the number of positions $j$ greater than $i$ such that $\rho(j) < i$:

$$\nu_I(i) \triangleq |\{j < i : \rho(j) \geq i \text{ or } j > i : \rho(j) < i\}|.$$

The constraint length can be defined also through the delay function as:

$$\nu_I(i) \triangleq |\{k \in \mathbb{Z} : \text{sgn}(k)d_I(k + i) > |k|\}|$$

with the convention $\text{sgn}(0) = 1$. The constraint length function is perhaps the least known and obvious parameter for an interleaver; we will see however that for causal interleavers this quantity has strong relationships with the memory requirements.

**Inverse interleaver**

Given an interleaver $(I, \rho)$, its inverse interleaver is obtained through the inverse permutation $\rho_I^{-1}$

$$(\bar{I}, \rho_{\bar{I}} = \rho_I^{-1})$$.
The delay functions of $\mathcal{I}$ and $\overline{\mathcal{I}}$ are tied by: $d_{\mathcal{I}}(i) = -d_{\overline{\mathcal{I}}}(\rho_{\mathcal{I}}(i))$. $\overline{\mathcal{I}}$ has the same period and characteristic latency of $\mathcal{I}$, its minimum/maximum delays are: $d_{\mathcal{I}}_{\text{min}} = -d_{\mathcal{I}}_{\text{max}}$ and $d_{\mathcal{I}}_{\text{max}} = -d_{\mathcal{I}}_{\text{min}}$ and obviously $D_T = D_{\mathcal{I}}$.

**Example 3**

Consider the following interleaver $\mathcal{I}$ and its action on the binary sequences $\mathbf{x}$:

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\rho_{\mathcal{I}}(i)$</th>
<th>$d_{\mathcal{I}}(i)$</th>
<th>$\nu_{\mathcal{I}}(i)$</th>
<th>$x(i)$</th>
<th>$y(i)$</th>
<th>$\rho_T(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$-4$</td>
<td>$-1$</td>
<td>$-3$</td>
<td>$2$</td>
<td>$1$</td>
<td>$0$</td>
<td>$2$</td>
</tr>
<tr>
<td>$-3$</td>
<td>$-6$</td>
<td>$3$</td>
<td>$3$</td>
<td>$0$</td>
<td>$x(-6)$</td>
<td>$-1$</td>
</tr>
<tr>
<td>$-2$</td>
<td>$-8$</td>
<td>$6$</td>
<td>$3$</td>
<td>$1$</td>
<td>$x(-8)$</td>
<td>$1$</td>
</tr>
<tr>
<td>$-1$</td>
<td>$-3$</td>
<td>$2$</td>
<td>$2$</td>
<td>$0$</td>
<td>$0$</td>
<td>$-4$</td>
</tr>
<tr>
<td>$0$</td>
<td>$3$</td>
<td>$-3$</td>
<td>$2$</td>
<td>$0$</td>
<td>$1$</td>
<td>$6$</td>
</tr>
<tr>
<td>$1$</td>
<td>$-2$</td>
<td>$3$</td>
<td>$3$</td>
<td>$1$</td>
<td>$1$</td>
<td>$3$</td>
</tr>
<tr>
<td>$2$</td>
<td>$-4$</td>
<td>$6$</td>
<td>$3$</td>
<td>$0$</td>
<td>$1$</td>
<td>$5$</td>
</tr>
<tr>
<td>$3$</td>
<td>$1$</td>
<td>$2$</td>
<td>$2$</td>
<td>$1$</td>
<td>$1$</td>
<td>$0$</td>
</tr>
<tr>
<td>$4$</td>
<td>$7$</td>
<td>$-3$</td>
<td>$2$</td>
<td>$1$</td>
<td>$0$</td>
<td>$10$</td>
</tr>
<tr>
<td>$5$</td>
<td>$2$</td>
<td>$3$</td>
<td>$3$</td>
<td>$0$</td>
<td>$0$</td>
<td>$7$</td>
</tr>
<tr>
<td>$6$</td>
<td>$0$</td>
<td>$6$</td>
<td>$3$</td>
<td>$1$</td>
<td>$0$</td>
<td>$9$</td>
</tr>
<tr>
<td>$7$</td>
<td>$5$</td>
<td>$2$</td>
<td>$2$</td>
<td>$0$</td>
<td>$0$</td>
<td>$4$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$\mathcal{I}$ has period $N_T = 4$, minimum delay $d_{\mathcal{I}}_{\text{min}} = -3$, maximum delay $d_{\mathcal{I}}_{\text{max}} = 6$, and characteristic latency $D_T = 9$. Its delay function is depicted in Figure 40. In Figure 41 we have pictorially represented the meaning of the constraint length function. The number of points of the delay function that fall into the shaded zone (borders included) corresponding to time $i$, represent the constraint length $\nu_{\mathcal{I}}(i)$. The constraint length function can then be derived as $\nu_{\mathcal{I}}(i) = 2, 3, 3, 2$. 
1) Causal and canonical interleavers: In previous subsection we have introduced a set of definitions for an interleaver $\mathcal{I}$ and its underlying permutation $\rho_{\mathcal{I}}$ without constraining the interleaver to be causal. However, for practical purposes, an interleaver $(\mathcal{I}, \rho_{\mathcal{I}})$ must be physically realizable, or causal, i.e., it must satisfy the property

$$i \geq \rho_{\mathcal{I}}(i) \quad \forall i \in \mathbb{Z}, \quad i.e., \quad d_{\mathcal{I}}_{\min} \geq 0.$$ 

In order to transform a non causal interleaver into a causal interleaver we need to introduce the concept of “interleaver equivalence”

a) Interleaver equivalence: For a concatenated system with an interleaver placed between two time-invariant devices a delay on the input sequence or on the output sequence does not influence the system behaviour. It is natural in this case to introduce the following equivalent classes.

Two interleavers are equivalent if one differs from the other only by a pure delay of the input or output sequence: $(\mathcal{I}, \rho_{\mathcal{I}})$ and $(\mathcal{I}', \rho_{\mathcal{I}'})$ are equivalent $(\mathcal{I} \equiv \mathcal{I}')$ if there exists a pair of integers $(a, b)$ such that for all $i$

$$\rho_{\mathcal{I}'}(i) = \rho_{\mathcal{I}}(i + a) + b$$

or, equivalently:

$$d_{\mathcal{I}'}(i) = d_{\mathcal{I}}(i + a) - (a + b)$$

We will denote an equivalence class of interleavers by the symbol $E_{\mathcal{I}}$. The characteristic latency $D_{\mathcal{I}}$ and the period $N_{\mathcal{I}}$ are invariant for all interleavers belonging to $E_{\mathcal{I}}$.

It is also useful to introduce the two subclasses $E_{\mathcal{I}}(x)$ and $E_{\mathcal{I}}(y)$ of $E_{\mathcal{I}}$ composed by all interleavers that are equivalent to $\mathcal{I}$ by a pure delay of the output $(y)$ or input $(x)$ sequences. This equivalent classes should be
used when the interleaver is placed between a time variant and a time-invariant device. In Figure 42 we present
the block diagrams realizing the introduced equivalent classes.

b) Canonical interleaver $I^*$: Having introduced the concept of interleaver equivalence, we can associate to any
interleaver $(I, \rho_I)$, the (equivalent causal) canonical interleaver $(I^*, \rho_{I^*}) \in Eq_x(I)$ with $d_{I^*, min} = 0$
characterized by:

$$I^* : \rho_{I^*}(i) = \rho_I(i) + d_{I^*, min}$$

For $I^*$, we have $d_{I^*, max} = D_{I^*} = D_I$. It is clear that an infinite number of other interleavers belonging to $Eq(I)$
with minimal delay could have been chosen as canonical for $I$ but, for reason that will be clear in the following
subsection, we have chosen the only one belonging to $Eq_x(I)$.

c) Canonical deinterleaver $J^*$: The action of an interleaver $(I, \rho_I)$ must be inverted at the receiver side by a
casual deinterleaver $(J, \rho_J)$ such that any sequence $\mathbf{y} = \rho_J(\mathbf{x})$ that enters $J$ is permuted into an output sequence
$\mathbf{z} = \rho_J(\mathbf{y})$ that is a possibly delayed version of $\mathbf{x}$: $z(i) = x(i - T(I, J))$ with $T(I, J) \in \mathbb{Z}$. $T(I, J)$ is the latency,
$i.e.$, the delay introduced in the transmission chain by the interleaver/deinterleaver pair.

Given an interleaver $(I, \rho_I)$, its inverse interleaver $(I, \rho_I)$ is surely a deinterleaver of $I$, and it yields $T(I, J) = 0$.
Moreover, all deinterleavers for $I$ are the elements of $Eq_y(I)^5$. Since the delay functions of the inverse interleaver
are the opposite of those of the corresponding interleaver, the inverse of a causal interleaver is anti-causal and not
physically realizable.

We can then define the (equivalent causal) canonical deinterleaver in the class of $Eq_y(I)$ as follows:

$$J^* : \rho_{J^*}(i) = \rho_I^{-1}(i - d_{I, max})$$

By definition, $J^*$ has $d_{J^*, min} = 0$ and $d_{J^*, max} = D_I$.

\[5\] In fact the interleaver that precedes the deinterleaver is not a time-invariant device, so that a delay on the input of the inverse interleaver is
not permitted.

---

Fig. 42. An interleaver and its equivalent classes.
The most interesting feature of the pair of canonical interleaver and deinterleaver \((\mathcal{I}^*, \mathcal{J}^*)\) concerns its latency, and is clarified by the following property stated as a theorem without proof.

**Theorem 2.** Given an interleaver \(\mathcal{I}\) with characteristic latency \(D_{\mathcal{I}}\), the latency \(T_{(\mathcal{I}^*, \mathcal{J}^*)}\) introduced by the pair of canonical interleaver and deinterleavers \((\mathcal{I}^*, \mathcal{J}^*)\) is minimum and equal to \(D_{\mathcal{I}}\).

In Figure 43 an interleaver with its canonical interleaver and deinterleaver is presented.

**Example 4**

Consider the interleaver \(\mathcal{I}\) of Example 3 and the two interleavers of period four: \(\rho_{\mathcal{I}_1}(i) = (4, -1, -3, 2)\) and \(\rho_{\mathcal{I}_2}(i) = (-8, -3, 3, -2)\). Their delay functions are, respectively: \(d_{\mathcal{I}_1}(i) = (-4, 2, 5, 1)\) and \(d_{\mathcal{I}_2}(i) = (8, 4, -1, 5)\). We have: \(\mathcal{I}_1, \mathcal{I}_2 \in Eq(\mathcal{I})\) (with \((a, b) = (0, 1)\) for \(\mathcal{I}_1\) and \((a, b) = (-2, 0)\) for \(\mathcal{I}_2\)), so that \(\mathcal{I}_2 \in Eq_y(\mathcal{I})\) and \(\mathcal{I}_1 \in Eq_x(\mathcal{I})\). The characteristic latency is still equal to nine for both \(\mathcal{I}_1\) and \(\mathcal{I}_2\).

The latency is important for system applications having stringent delay requirements. The previous theorem shows that the minimum latency introduced by an interleaver/deinterleaver causal pair is equal to the characteristic latency \(D_{\mathcal{I}}\). Practical motivations can lead to latencies greater than this minimum value (see Section V-D4 for the two-register implementation of block interleavers). For this reason, in the turbo code literature, the minimum latency \(D_{\mathcal{I}}\) and the period \(N_{\mathcal{I}}\) are often confused.

For causal interleavers the constraint length function \(\nu_{\mathcal{I}}(i)\) is constant and assumes an important meaning: it is the minimum amount of memory required to implement the interleaver.

### C. Connections among interleaver parameters

In this subsection, we explore the relations between the various interleaver parameters. The connection between the state spaces of an interleaver and its inverse is clarified by the following lemma.
Lemma 1. The constraint length functions of an interleaver \((\mathcal{I}, \rho_{\mathcal{I}})\) and its inverse \((\mathcal{I}, \rho_{\mathcal{I}}')\) are equal:

\[
\nu_{\mathcal{I}}(i) = \nu_{\mathcal{I}}'(i).
\]

A strong relation exists between the delay function of an interleaver and its constraint length function.

Theorem 3. Given an interleaver \((\mathcal{I}, \rho_{\mathcal{I}})\) of period \(N_{\mathcal{I}}\), the average of the absolute values of the delays is equal to the average of the constraint length function:

\[
\frac{\sum_{i=0}^{N_{\mathcal{I}}-1} |d_{\mathcal{I}}(i)|}{N_{\mathcal{I}}} = \frac{\sum_{i=0}^{N_{\mathcal{I}}-1} \nu_{\mathcal{I}}(i)}{N_{\mathcal{I}}}
\]

For a causal interleaver, the constraint length function is constant and equal to \(\nu_{\mathcal{I}}\). From the Theorem 3 then we have:

Corollary 1. For causal interleavers, the average delay is equal to the constraint length:

\[
\frac{\sum_{i=0}^{N_{\mathcal{I}}-1} d_{\mathcal{I}}(i)}{N_{\mathcal{I}}} = \nu_{\mathcal{I}}
\]

Given a causal interleaver \(\mathcal{I}\) and one of its causal deinterleavers \(\mathcal{J}\), the sum of the average delays through \(\mathcal{I}\) and \(\mathcal{J}\) is clearly equal to the introduced latency \(T(\mathcal{I}, \mathcal{J})\). This lemma follows from Corollary 1:

Lemma 2. The sum of the constraint lengths of a causal interleaver/deinterleaver pair \((\mathcal{I}, \mathcal{J})\) is equal to the latency \(T(\mathcal{I}, \mathcal{J})\):

\[
\nu_{\mathcal{I}} + \nu_{\mathcal{J}} = T(\mathcal{I}, \mathcal{J})
\]

When \(\mathcal{I}\) and \(\mathcal{J}\) are the canonical causal interleaver/deinterleaver pair, the introduced latency is equal to the characteristic latency, and then we have the following key property:

Lemma 3. The sum of the constraint lengths of a causal canonical interleaver/deinterleaver pair \((\mathcal{I}^*, \mathcal{J}^*)\) is equal to the characteristic latency \(D_{\mathcal{I}}\):

\[
\nu_{\mathcal{I}^*} + \nu_{\mathcal{J}^*} = D_{\mathcal{I}}
\]

This Lemma coincides with Theorem 4 of [38], where it was shown that the sum of the constraint lengths of the interleaver and the deinterleaver (called minimum storage capacities) is equal to the latency. Lemma 3 completely clarifies the fundamental role of the characteristic latency of an interleaver. In fact, given \((\mathcal{I}, \rho_{\mathcal{I}})\), its characteristic latency \(D_{\mathcal{I}}\) is equal to:

- The difference between the maximum and the minimum delay of \(\mathcal{I}\)
- The difference between the maximum and the minimum delay of any interleaver equivalent to \(\mathcal{I}\)
- The difference between the maximum and the minimum delay of any deinterleaver for \(\mathcal{I}\)
- The maximum delay of the canonical interleaver \(\mathcal{I}^*\) associate with \(\mathcal{I}\)
- The maximum delay of the canonical deinterleaver \(\mathcal{J}^*\) associate with \(\mathcal{I}\)
- The sum of the constraint lengths of \(\mathcal{I}^*\) and \(\mathcal{J}^*\)
The minimum amount of memory cells required to implement the pair of canonical interleaver and deinterleaver.

- The minimum latency introduced in the transmission chain by an interleaver/deinterleaver pair involving $I$ or any interleaver equivalent to $I$.

### D. Convolutional and Block interleavers

The interleaver parameters and properties previously introduced hold for all periodic interleavers. In the following, we will introduce a way to describe general periodic interleavers that will be called convolutional interleavers, as opposed to the special and important subclass of block interleavers, which are the most used in practice.

**1) Convolutional interleavers:**

Given an interleaver $(I, \rho_I)$ of period $N_I$, a convenient and completely general way to represent the permutation $\rho_I$ is the following:

$$\rho_I(i) = \pi(i \mod N_I) + \left(\left\lfloor \frac{i}{N_I} \right\rfloor - \alpha(i \mod N_I)\right)N_I$$

In the fundamental period thus we have:

$$\rho_I(i) = \pi(i) - \alpha(i)N_I$$

where $\pi$ is a finite basic permutation of length $N$, $\pi : \mathbb{Z}_N \rightarrow \mathbb{Z}_N$, and $\alpha(j)$ is the $j$-th element of a shift vector $\alpha$ of $N$ elements taking values in $\mathbb{Z}$.

A visual implementation of the convolutional interleaver described in (40) is shown in Figure 44, where the reader can recognize the similarities with the familiar structure of convolutional interleavers described by Ramsey and Forney in [38], [39].

The input stream is permuted according to the basic permutation $\pi$. The $i$-th output of the permutation register is sent to a delay line with a delay $\alpha(i)$, whose output is read and sent out. Notice that in the realization of Figure 44 the permutation $\pi$ is always non causal, except for the case of the identity permutation. As a consequence, it is in general not physically realizable. In the next subsection, we will describe a minimal realization of any periodic causal interleaver.

![Fig. 44. Structure of a convolutional interleaver](image)
Example 5

A simple class of interleavers stems from the following choice of the identity basic permutation:

\[ \rho(i) = i - \alpha(i)N \]

As an example, choosing \( N = 4 \), \( \pi = (0, 1, 2, 3) \), and \( \alpha = (2, 1, 0, -10) \) yields:

\[
\begin{bmatrix}
  i & ... & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & ... \\
  \rho(i) & ... & -8 & -3 & 2 & 43 & -4 & 1 & 6 & 47 & 0 & 5 & 10 & 51 & 4 & 9 & 14 & 55 & 8 & ... \\
\end{bmatrix}
\]

2) Implementation of convolutional interleavers with minimal memory requirement: We have already seen in Section V-C that the minimal memory requirement for the implementation of a causal interleaver is the constraint length \( \nu_I \), which in turn is equal to the average of the delay profile. An important property that relates the shift vector \( \alpha \) to the constraint length of causal interleavers is given in the following theorem stated without proof.

**Theorem 4.** The constraint length of a causal interleaver of period \( N \) is equal to the sum of the shift vector elements \( \alpha \) through:

\[
\nu_I = \sum_{i=0}^{N-1} \alpha(i)
\]

We can now describe an implementation with minimal memory requirements of a generic periodic causal interleaver. It is realized through a single memory with size equal to the constraint length \( \nu_I \). Since the device require the minimum amount of memory, the input and output symbols at a given time \( i \) are read and stored with a single Read-Modify-Write operation acting on a single memory cell. The address \( a(i) \) of this cell is derived as follows (see Figure 45).

At time \( i \) we read \( x(\rho(i)) \) from a given memory cell, and write \( x(i) \) into it. As a consequence, the successive contents of that memory cell are

\[ \ldots, x(\rho^2(i)), x(\rho(i)), x(i), x(\rho^{-1}(i)), x(\rho^{-2}(i)), \ldots. \]

Thus, the permutation \( \rho \) induces a partition of the set of integers \( \mathbf{Z} \) with the following equivalence relationship:

\[ i \equiv j \iff \exists k \in \mathbf{Z} : i = \rho^k(j) \]
and the constraint length $\nu_2$ thus equals the cardinality of this partition.

With the previous considerations in mind, we can devise the following algorithm to compute the addresses $a(i)$ to be used in the minimal memory interleaver of Figure 45:

1) Set the running variable $\nu = 0$
2) for all $i \geq 0$
3) if $\rho(i) = i$, then the input symbol $x(i)$ is directly transferred to the output. In this case the memory contents are not modified (the address $a(i)$ will be conventionally set to 0).
4) if $\rho(i) < 0$ then $\nu = \nu + 1$ and $a(i) = \nu$
5) if $\rho(i) \geq 0$ then $a(i) = a(\rho(i))$.

Since the interleaver is periodic, the address sequence $a(i)$ is also periodic. Its period can be obtained by deriving the following expression of the $l$-th power of the permutation $\rho$ in the fundamental interval $[0, N - 1]$:

$$\rho^l(i) = \pi^l(i) - \left[ \sum_{j=0}^{l-1} \alpha(\pi^j(i)) \right] N$$

To derive the period of the sequence $a$ we must use the cycle decomposition of the basic permutation $\pi$

$$\pi = (\pi_1) \cdots (\pi_n)$$

For each cycle $(\pi_k)$ of the basic permutation we can define the following basic period:

$$P_k = \sum_{j=1}^{l_k} \alpha(\pi_{kj}) \ \forall k = 1, \ldots, n$$

where $l_k$ is the length of the cycle $(\pi_k)$ and $\pi_{kj}$ is its $j$-th element. The periods $P_k$ of a cycle then corresponds to the sum of all the shift vector elements associated to the cycle elements.

The period of the address sequence is $N$ times the least common multiple of all basic periods $P_k$

$$P = N\text{lcm}(P_1, \ldots, P_n)$$

From this last expression it can be observed that, even with very small periods $N$, the period of the address sequence $a(i)$ of the minimal memory implementation can assume large values.

Example 6
As an example, consider the causal periodic permutation with period $N = 2$, $\rho(0) = -4$, $\rho(1) = -5$; we have:

$$\pi = (0, 1)$$

$$\alpha = (2, 3)$$

The basic permutation can be decomposed as follows:

$$\pi = (0)(1)$$

So that the period of the address sequence is

$$P = 2\text{lcm}(2, 3) = 12$$
Applying the algorithm previously described we have in fact that the address sequence is periodic with period 12:

\[ a = (1, 2, 3, 4, 1, 5, 3, 2, 1, 4, 3, 5) \]

3) **Block interleavers:** Block interleavers, a particular case of the general class of periodic interleavers previously described, are very important for practical applications, and form the basis of most turbo code schemes. A block interleaver is generated by a permutation \( \pi \) of length \( N \), \( \pi : \mathbb{Z}_N \rightarrow \mathbb{Z}_N \), made periodic on all \( \mathbb{Z} \), and so yielding the infinite permutation

\[ \rho : \mathbb{Z} \rightarrow \mathbb{Z}, \quad \rho(i) = \pi(i \mod N) + \left\lfloor \frac{i}{N} \right\rfloor N \quad (42) \]

The period of \( \rho \) is clearly equal to the length \( N \) of \( \pi \). A block interleaver (apart when \( \pi \) is the identity) is not causal, and has a non-positive minimum delay \(-(N - 1) \leq \delta_{I_{\text{min}}} \leq 0\). The maximum delay is \( 0 \leq \delta_{I_{\text{max}}} \leq (N - 1) \), then the characteristic latency is \( 0 \leq D \leq 2(N - 1) \).

Block interleavers are a particular case of convolutional interleavers, obtained by posing \( \alpha = (0, \ldots, 0) \) in the general representation (40). A convolutional interleaver \( I' \) with period \( N \) is equivalent to a block interleaver if there exists an \( i \in \mathbb{Z} \) such that the set \( \{\rho_{I'}(i), \rho_{I'}(i + 1), \ldots, \rho_{I'}(i + N - 1)\} \) is a set of \( N \) adjacent numbers. As an example, this happens if the shift vector in the representation (40) has the form

\[ \alpha = (k - 1, \ldots, k - 1, k, \ldots, k) \quad \forall l \leq N \]

4) **Block interleaver causalization:** Among all possible choices to construct a causal interleaver equivalent to a given block interleaver, two are particularly interesting, and will be described in the following.

5) **The canonical causal interleaver of a block interleaver:** For a block interleaver \( I \), generated by a permutation \( \pi \) of size \( N \) with a certain minimum delay \( \delta_{I_{\text{min}}} \), we can consider its equivalent causal canonical interleaver \( (I^*, \rho_{I^*}) \in Eq_y(I) \), with \( \rho_{I^*}(i) = \rho_{I}(i + \delta_{I_{\text{min}}}) \), with \( \delta_{I_{\text{min}}} = 0 \), and \( \delta_{I_{\text{max}}} = D \). By definition of canonical interleaver, \( I^* \) is the best choice in terms of latency: by using the canonical deinterleaver, \( T \) is equal to \( T = D \leq 2(N - 1) \). The constraint length of \( I^* \) is characterized by the following lemma:

**Lemma 4.** The canonical interleaver \( I^*, \rho_{I^*} \) of a block interleaver \( I, \rho_I \) with minimum delay \( \delta_{I_{\text{min}}} \) has constraint length:

\[ \nu_{I^*} = |\delta_{I_{\text{min}}}| \]

Lemma 4 shows that a minimal encoder for \( I^* \) needs only \( \nu_{I^*} = \delta_{I_{\text{min}}} \) cells. The algorithm previously presented in Section V-D2 can be applied to realize it.

6) **The two-register causal interleaver of a block interleaver:** In practice, to make a block interleaver causal, the causal interleaver \( (I', \rho') \) with \( \rho'(i) = \rho(i - N) \) is often used instead of the canonical interleaver. \( I' \) corresponds to an encoder implementation largely used in practice, which consists of two registers of length \( N \) used alternatively, one for writing the input bits of the current block, and the other for reading the output permuted bits of the preceding
block. Clearly, in this case \( \nu_{I'} = N \). \( I' \) has a maximum delay \( d_{I'_{\text{max}}} = d_{I_{\text{max}}} + N \geq D \) usually larger than \( D \), and then it leads to a non-minimum latency. If also the deinterleaver \( J' \) for \( I' \) is realized by the same two-register strategy, the introduced latency is equal to \( T_{(I', J')} = 2N \).

**Example 7**

The block interleaver \( (I, \pi) \) with \( N = 8 \) and \( \pi = (3, 5, 0, 2, 1, 7, 6, 4) \) has \( d_{I_{\text{min}}} = -4 \) and \( D_I = 7 \). Its canonical interleaver is \( (I^*, \rho_{I^*}) \) with \( \rho_{I^*} = (-7, -1, -2, -4, 3, 5, 0, 2) \), with a minimum \( \nu_{I^*} = 4 \).

The two-register causal interleaver is \( (I', \rho') \) with \( \rho' = (-5, -3, -8, -6, -7, -1, -2, -4) \), with \( \nu_{I'} = 8 \).

By using the canonical deinterleaver \( J^* \) of \( I^* \) with \( \rho_{J^*} = (0, -1, -4, 3, -3, 2, 1, 6) \), the pair \( (I^*, J^*) \) introduces a latency equal to 7.

By using the two-register deinterleaver \( J' \) for \( I' \) with \( \rho_{J'} = (-6, -4, -5, -8, -1, -7, -2, -3) \), the pair \( (I', J') \) introduces a latency equal to 16.

\[ \Box \]

**E. Some practical interleavers**

In this section we will review some of the interleavers that have been proposed in the literature as good candidates for concatenated schemes.

We have seen in Section III that using the uniform interleaver technique it is possible to design constituent encoders for concatenated schemes that are good on average with respect to the whole class of possible permutations associated with them. The second step in the design procedure of a good concatenated scheme is the choice of a particular interleaver that gives the best possible performance associated to the considered scheme.

The interleaver law heavily influences both the turbo code distance spectrum (ML decoding) and the iterative decoding algorithm. As such, the choice of the interleaver is a crucial design issue. Optimization of the interleaver for a given code structure, however, is a very hard task to accomplish for the following reasons:

- The set of candidates is huge, even for very small interleaving lengths (essentially, \( N! \) for block interleavers)
- It is very hard to associate to different interleavers a set of parameters, easy to compute, that characterize their performance and allow to compare them, so that an effective optimization must be based mainly on simulation results
- Since the iterative decoding process is suboptimal, and little is known about its convergence, the relationship between the interleaver structure and the decoding process is still an open problem
- All results obtained so far suggest that the interleaver optimization also depends on the required bit (or frame) error probability and/or the operating signal-to-noise ratio. Quite often, a good interleaver for low signal-to-noise ratios may present a pronounced error floor, and vice-versa.

The task of finding the optimal interleaver is then impossible to realize and most of the optimization techniques adhere to the following steps:

- Start with a class of interleavers that satisfy basic system constraints.
- Evaluate analytically the performance and reduce the class to a few samples. In general this task is quite hard, since it implies to possess some analytical techniques applicable to a given deterministic interleaver and fixed CCs.
Simulate the concatenated scheme with the survived interleavers. This task is always required since analytical techniques are not sufficiently reliable and accurate to predict the full behavior of a given concatenated scheme, especially in the error floor region.

The following are a set of heuristic rules that have been used to design good interleavers:

**Randomness:**

The analytical results obtained using a uniform interleaver show that good ML performance can be obtained by selecting randomly an interleaver in the set of permutations. Exactly as random codes gives optimal performance when the size of the code grows to infinity, random interleavers in concatenated schemes gives optimal performance when their length grows to infinity.

**Spread-Correlation:**

In Section IV we have seen that the iterative decoding procedure is based on the assumption that the LLRs of consecutive bits entering the SISO modules are independent. On the other hand, each SISO decoder introduces correlations between the output LLRs of consecutive symbols so that, to satisfy the independence assumption, the interleaver should spread as much as possible consecutive bits, or, even better, should decorrelate as much as possible the output sequence.

**Free distance:**

ML analysis shows that the free distance of the turbo code, together with the first terms of its distance spectrum, dominate the performance at high signal-to-noise ratio and are responsible of the error floor phenomenon. However, when the iterative decoding algorithm is applied, the influence on it of the distance spectrum is still unclear.

**Simple law of generation:**

Although interleavers are very simple to implement by storing the permutation addresses in some read-only-memory, it is sometimes preferable to compute the permutation law on-the-fly. This method allows to define a class of interleavers with different lengths working in a given concatenated scheme without having to store all permutations. In these cases, the algorithm to compute the permuted addresses should be analytically defined and to be as simple as possible.

**Code matching:**

In a more sophisticated attempt to optimize the interleaver for a particular concatenated scheme, one has to consider the CCs as a part of the design. In this respect the interleaver design becomes part of the overall code optimization. This approach, which is optimum in principle, has the same computational problems that are found in the design of good block codes with large block lengths. All main parameters of the code (free distance, weight enumerating function) present an NP complexity, and become unaffordable for large interleaver lengths.

In the following subsections, we will present the algorithmic description that permits to derive some of the most known permutations, trying to emphasize which of the previous aspects were considered in the derivation of them:
1) Random Interleaver
2) Spread-random (S-random) Interleaver
3) Pseudo-Random Interleaver
4) Congruential-type permutations
5) Multidimensional Interleaver

Then more interleaver classes such as
1) (Dithered) Golden Interleaver
2) Berrou's Interleaver
3) PIL interleaver Interleaver

will be discussed. Finally we address the concept of pruning, extending interleavers, and the important problem of memory collision.

1) Random Interleaver: A random permutation of length $N$ can be generated according to the following algorithm, which performs $N - 1$ random transpositions on any starting valid permutation (for example the identity permutation):

1) Define a register $R$ containing the integers from 0 to $N - 1$, for example $R(i) = i, i = 0, \ldots, N - 1$.
2) For $n = 0, \ldots, N - 2$:
   3) Generate an integer $g_n$ according to a uniform distribution between 0 and $N - n - 1$.
   4) Transpose the two elements $R(n)$ and $R(n + g_n)$.

Random interleavers yields performance close to those predicted by the uniform interleaver analysis, and thus are very good reference interleavers. If one is interested to emulate the uniform interleaver behavior in a simulation scheme, the closest approximation is obtained by picking a new random interleaver for each transmitted frame.

The performance of random interleaver at medium-low SNRs i.e., where the performance of the coding scheme depends on its whole distance spectrum and not only of the first few terms, is very good and in general it is hard to find interleavers that outperform the random interleaver in this SNR region. On the contrary, for high signal-to-noise ratios, random interleavers show rather high error floors, close to those predicted by the uniform interleaver, and thus they can be easily outperformed by other interleaving laws, specifically designed to increase the free distance.

2) Spread Interleaver: We can define the spread of an interleaver as follows:

$$S = \min S' : |\pi(i) - \pi(j)| \geq S' \quad \forall |i - j| \leq S' \quad i, j = 0, \ldots, N - 1$$

The spread of an interleaver then measures the ability of a permutation to separate adjacent symbols in a sequence.

The spread interleaver, proposed in [40], is in fact a spread-random interleaver. It is based on the random generation of $N$ integers from 0 to $N - 1$ (as for the random interleaver), with the following constraint:

The randomly selected integer at step $n$ is compared to the $S_1$ most recently selected integers. If the current selection is within a distance of $S_2$ from at least one of the previous $S_1$ numbers, it is rejected and a new extraction takes place until the previous condition is satisfied.
Fig. 46. An example of pseudo-random generator, generating a permutation of size $N = 3^3 - 1 = 7$

The process is repeated until all $N$ integers have been extracted. The two parameters $S_1$ and $S_2$ should be chosen larger than the memory span of the two CCs, and, when the two CCs are equal, it is appropriate to impose:

$$S_1 = S_2 = S$$

The searching time to complete the interleaver increases with $S_1$ and $S_2$, and there is no guarantee that the process will finish successfully. As a rule of thumb, the choice

$$S = \sqrt{\frac{N}{2}}$$

produces a solution in a reasonable time. Note, finally, that for $S = 1$ we have a purely random interleaver.

The spread-interleaver is a very good interleaver for all range of signal-to-noise ratios. Being a random interleaver, in fact, it preserves the good properties of random interleavers in the waterfall region. In addition, the spread constraint adds two beneficial features:

- Since consecutive LLRs at the output of one SISO module cannot be mapped into consecutive LLRs at the input of the next SISO, the independence assumption of the iterative decoding process is reinforced.
- Also, the error events associated to the lowest interleaver gain are those with small lengths and weights on both CCs. The spread constraint, together with the recursiveness of the constituent encoders, guarantees that pairs of short error events on both constituent codes are avoided up to a given point and thus increases the minimum distance of the code.

It must be noticed, however, that a large spread in itself, without randomness, can lead to very bad performance. As an extreme example, the classical square row-by-column interleaver, with size $N = M \times M$, yields the best possible spread $S = M - 1 = \sqrt{N} - 1$, but is known to offer very poor performance, due to the clustering of the distance spectrum that increases the multiplicities of some particular error patterns.

3) Pseudo-Random Interleavers: A straightforward way to generate a random sequence of numbers between 0 and $2^m - 1$, and thus a possible permutation, is to read the contents of a feedback shift register whose feedback coefficients are determined by a primitive polynomial of degree $m$, as in Figure 46.

The shift register is initially loaded with a word different from 0 and left to cycle through all $2^m - 1$ different binary words, which can be used as addresses for the permutation.

In spite of the pseudo-randomness of the output sequence of a feedback shift register, however, its state sequence is far from being a random sequence of integers. This fact makes pseudo-random interleavers rather poor ones.
4) Congruential-type Interleavers:

a) Linear: A general class of permutations of length \( N \) can be obtained using the linear congruential equation

\[
\pi(n) = np + s \mod N
\]

where \( p \) is relatively prime with respect to \( N \). Choosing properly the factor \( p \) and the offset \( s \) allows to adapt the permutation to a given pair of CCs.

This very simple law allows to define a large class of permutations starting with two parameters, and provides interleavers easy to generate on-the-fly. Its performance, however, is not very good in general. Next case describes a possible improvement.

b) Exponential: The Welch-Costas permutation [41], is defined as:

\[
\pi(n) = [a^n \mod p] - 1 \quad 0 \leq n < N
\]

where the length of the interleaver is \( N = p - 1 \) with \( p \) a prime number, and \( a \) is a primitive element, which has the property that \( a^1, a^2, \ldots \) are distinct modulo \( p \). This permutation leads to a maximum dispersion. Variations of this permutation can be obtained substituting the exponent in (43) with another permutation \( \pi' \):

\[
\pi(n) = [a^{\pi'(n)} \mod p] - 1 \quad 0 \leq n < N
\]

When it is required that the permutation be generated “on-the-fly” the algorithm that generates the reading address must be as simple as possible. Congruential permutations satisfy this constraint and thus are sometimes used as a starting class for the exponent in (43) to keep the law simple. The nonlinear relationship between the congruential law and the permutation addresses makes this class of interleavers better in performance than the plain congruential ones.

Since we mentioned dispersion, we provide its definition as follows: The dispersion of an interleaver \( \pi \) is the number of elements (pair of integers) of the set

\[
\mathcal{D}(\pi) = \{ [(j - i), (\pi(j) - \pi(i))] | 0 \leq i < j < N \}
\]

The normalized dispersion is defined as

\[
D_s = \frac{2|\mathcal{D}(\pi)|}{N(N-1)}
\]

\( D_s \) is also called dispersion factor and it is between \( \frac{1}{N-1} \) and 1. The closer is \( D_s \) to 1, the better is the interleaver from point of randomness. Note that the condition imposed on the components of the pair of integers was used to describe the spread interleaver.

5) Multidimensional Interleaver: In general, permutation laws that are described through a simple analytical relationship, like a congruential one, do not lead to very good performance.

On the other hand good random interleavers can be described only through their permutation vector, and thus they are not well suited for applications in which the generation of the permutation on-the-fly and the frequent changing of the law itself is a must.
In this subsection we describe a general technique to build complex permutation laws starting from a set of simpler constituent permutations. This hybrid technique leads to most of the good permutation proposed in literature and adopted in standards.

Let us assume that the interleaver size $N$ can be factorized as

$$N = N_1 \times N_2 \times \ldots \times N_p$$

and that we have defined a set of permutation laws $\pi_1, \ldots, \pi_p$, one for each of its factors, with size equal to the factor itself.

A multidimensional permutation with size $N$ can be defined as follows:

1) Decompose the writing index $n \in [0, N - 1]$ as follows:

$$n = n_1 + n_2 N_1 + n_3 N_1 N_2 + \ldots + n_p N_1 \cdot \cdots \cdot N_p - 1$$

where

$$n_1 = n \mod N_1$$
$$n_2 = \left\lfloor \frac{n}{N_1} \right\rfloor \mod N_2$$
$$\vdots$$
$$n_p = \left\lfloor \frac{n}{N_1 \cdot \cdots \cdot N_{p-1}} \right\rfloor \mod N_p$$

In this way establish a one-to-one correspondence between the integer $n \in \{0, \ldots, N - 1\} N$ and the $p$-tuple of coordinate integers $(n_1, \ldots, n_p)$ with $n_i \in \{0, \ldots, N_i - 1\}$

2) Construct the reading address as follows:

$$\pi(n) = \sum_{i=1}^{p} \pi_{\Pi(i)}(n_{\Pi(i)}) \prod_{j=1}^{i-1} N_{\Pi(j)}$$

where $\Pi$ is the coordinate permutation with size $p$.

**Example 8**

Suppose that $N$ can be factored into two factors $N = N_1 N_2$ then a two-dimensional congruential interleaver can be obtained as follows:

$$n = n_1 + n_2 N_1 \quad \text{where} \quad n_1 = n \mod N_1, \quad n_2 = \left\lfloor n/N_1 \right\rfloor \mod N_2$$

$$\pi_1(n_1) = p_1 N_1 + s_1 \mod N_1$$

$$\pi_2(n_2) = p_2 N_2 + s_2 \mod N_2$$

$$\pi(n) = \pi_2(n_2) + \pi_1(n_1) N_2$$
6) More interleaver classes:

- The (dithered) Golden interleavers [42]: This class of interleavers draws its name from the golden section value and had been shown to provide good performance in terms of error floor Golden interleavers are based on the golden section value 
  \[ g = \frac{\sqrt{5} - 1}{2} = 0.618 \]

They are constructed as follows: First compute real increment

\[ c = N \frac{g^m + j}{r} \quad m, j, r \in \mathbb{Z} \]

select \( p \) close to \( c \) such that \( p \) and \( N \) are relatively prime.

Build a golden vector \( \nu \) from the following congruential equation on numbers \( 0 \leq n < N \):

\[ \nu(n) = s + np \mod N \]

where \( s \) is a starting value. The starting value \( s \) is usually set to 0.

Typical values for the parameters \( m, j, \) and \( r \) are as follows. The preferred value for \( m \) are 1-2. For maximum spreading of adjacent elements \( j = 0 \) and \( r = 1 \). For turbo codes greater values of \( j \) and \( r \) may be used to obtain the spreading for elements spaced \( r \) apart. For the simplest golden interleaver without dither we just set \( \pi(n) = \nu(n) \). In order to increase the randomness of golden interleavers, a desirable property for convergence of the iterative decoding algorithm at very low signal-to-noise ratios, a real perturbation (dither) vector \( d(n) \) is added to the congruential equation as

\[ \nu(n) = s + nc + d(n) \mod N \]

Now \( \nu \) is a real vector and there is no need to find \( p \) rather we use directly \( c \) in the congruential equation. Next find a sort vector \( z(n) \) such that for \( 0 \leq n < N \), the vector \( a(n) = \nu(z(n)) \), namely the vector \( a \) contains the elements of \( \nu \) sorted in a rising or descending order. Finally \( \pi(z(n)) = n \), or it can be also defined as \( \pi(n) = z(n) \). Each element \( d(n) \) is uniformly distributed between 0 and \( ND \) where typical value of \( D \) is 0.01.

Golden interleavers have very good spreading properties, as \( s \)-random interleavers. For more details and variations of the above algorithm see reference [42].

- The original Berrou’s interleaver [1]: This interleaver proposed in the original PCCC construction, and is based on a square array with \( N = M \times M \) entries, where \( M \) is a power of 2. The array entries are read according to a algorithm described below in which the column index is a function of the row index.

Let \( 0 \leq i < M \) and \( 0 \leq j < M \) be the addresses of the row and column for writing, and \( 0 \leq i_r < M \) and \( 0 \leq j_r < M \) the addresses of the row and column for reading. The permutation is given by

\[
\begin{align*}
  i_r &= \left( \frac{M}{2} + 1 \right) (i + j) \mod M \\
  k &= (i + j) \mod 8 \\
  j_r &= [p(k)(j + 1)] - 1 \mod M
\end{align*}
\]
The $L$ numbers $p(k)$ are distinct numbers, relatively prime with respect to $M$, functions of the row address, and $L$ is a small integer whose value is chosen according to $M$. Typically, $L=8$ is the choice for $M$ up to 128, which corresponds to $N = 16,384$. The original sequence for $p(k), k = 1, ..., 8$ was the following:

$$7, 17, 11, 23, 29, 13, 21, 19$$

The rationale for the reading rule are the following: the multiplying factor $(M/2+1)$ prevents two neighbouring input data written on two consecutive rows from remaining neighbors in reading.

Reading is also performed diagonally (the term $(i + j)$) with respect to writing to avoid regular pattern effects concerning input sequences with low weight that are so detrimental in connection with standard row-column interleavers.

- The PIL interleaver [43]: is the permutation that is currently adopted for the UMTS-3GPP channel coding standard based on turbo codes for data application. PIL interleavers with block sizes in the range (40-5114) are obtained by pruning a set of rectangular mother interleavers with 20 rows and a number of columns that must be equal to a prime number $p$ plus or minus 1. Input bits are written row by row and permutation are performed on both rows and columns. The row permutation is done according to one of the following two vectors

$$A : (19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 10, 8, 13, 17, 3, 1, 16, 6, 15, 11)$$
$$B : (19, 9, 14, 4, 0, 2, 5, 7, 12, 18, 16, 13, 17, 15, 3, 1, 6, 11, 8, 10)$$

The column permutation is a function of the row $i$ defined as follows:

$$c_j(i) = g_0^{ip_j \mod (p-1)} \mod p$$

where $p_j$ is a set of 20 numbers greater than 6 relatively prime with respect to $p - 1$

In Figure 47, borrowed from [42], we show the BER performance of some interleavers drawn from the classes previously described. The curves show how important the interleaver choice can be for the error floor behavior.

7) Pruning and extending interleavers: A very important, practical problem arising in the implementation of concatenated codes with interleavers is the so-called memory collision problem. For its description, together with a general solution to implement any given interleaver avoiding collision, see [44].

For system applications requiring different code block lengths, such as for example adaptive coded modulation, it is essential to be able to obtain all required interleaver length minimizing the memory requirements.

To this aim, it is possible to construct a class of interleavers having different lengths starting from a mother interleaver with size equal to largest one and simply pruning (deleting) the positions that are not required for the shorter interleavers, or alternatively, starting with shortest one and growing it through all required lengths.

Useful references describing techniques to construct classes of interleavers with different block lengths and good performance can be found in [45] and [46].
Fig. 47. Simulated performance of different interleavers. The figure is borrowed from [42]
VI. Performances

A. Introduction

In this section, we will show several examples of performance of turbo code structures obtained by simulation. The examples often refer to those already presented in previous sections of the Chapter.

B. Iterative decoding versus ML upper bounds

We consider the two PCCCs for which we computed the ML upper bound to the bit error probability in 1. The ML performance showed that the code PCCC2 yielded better performance as predicted by the design criteria. We show now that the improvement yielded by PCCC2 over PCCC1 under uniform interleaving is preserved with actual, fixed interleavers. We have simulated the two PCCCs with interleavers of size $K = 100$ chosen at random. The decoder uses the iterative algorithm described in Section IV with a number of iterations equal to 20. The results are reported in Figure 48, where we have also redrawn the union upper bounds obtained in Section III.

They show a good agreement between simulation and ML analysis down to signal-to-noise ratios around 2 dB. Below that value (cutoff rate) the union bound diverges and becomes useless.

Next, consider again the two PCCCs of Example 2, one constructed using a non recursive CC, and the other with a recursive CC. We have simulated the two turbo codes with same parameters already considered in Section III, and the results in terms of BER \(^6\) are reported in Figure 49. They show that the performance (dashed curves) of the turbo code employing a recursive CC improve with the interleaver length, whereas those of the PCCC based

\(^6\)We distinguish between bit error probability obtained by simulation, called BER in the figures, and that obtained through the ML analysis, called $P_b(e)$. 

Fig. 48. Simulation results for the codes PCCC1 and PCCC2 of Example 1 compared to union upper bounds. The interleaver used for simulations is a random interleaver with size $K = 100$, and the simulated iterative decoding algorithm employs 20 iterations.
Fig. 49. Comparison of BER performance of a turbo code using 2-state recursive (SR) or not recursive (NSR) systematic encoders. Block sizes are 100, 1,000, and 10,000 on non recursive CC stay on top of each other. The curve corresponding to an interleaver length of 10,000 also shows the error floor phenomenon just above BER=10^{-6}.

C. Optimality of constituent encoders

In Example 1 we had already showed by ML analysis, confirmed by simulation in Figure 48, that the PCCC based on a CC matching the design criteria in terms of maximization of its effective free distance offered better performance wrt a non optimum CC. Consider now an an SCCC using as inner encoder two 4-state recursive encoders with different effective free distances, one optimized and the other not optimized. The simulation results in terms of BER and FER (the simulated word error probability) are shown in Figs. Figure 50 and Figure 51. They fully confirm that also for SCCCs the inner CC must be chosen according to the design criterion of maximizing its effective free distance. The comparison of performance of the SCCCs with interleaver length 10,000 are particularly impressive. The SCCC with optimized inner CC does not show any trace of error floor, while the other has an error floor around 10^{-6}.
Fig. 50. Comparison of simulated BER and FER performance of an SCCC using two 4-state recursive encoders with different effective free distances. Information block size is 1000.

**D. Performance versus number of iterations**

In this subsection the behavior of the iterative decoding algorithm will be examined versus the number of iterations. In Figure 52, Figure 53, and Figure 54 we report the BER performances of the iterative decoder for a PCCC with 2, 4 and 8 states recursive CCs versus signal-to-noise ratio (SNR). Different curves refer to different number of iterations, from 1 to 10.

It is evident how the BER curves become steeper with increasing number of iterations, up to a point where the number of iterations does not help anymore. It can also be verified that increasing the number of states of the CCs yields a an increasingly steeper slope of the curves in the waterfall region. Finally, the error floor effect around $10^{-6}$ is also apparent.

In Figs. 56 and 57 we show the behavior of BER versus number of iterations for a PCCC and an SCCC using 4-state CCs. Different curves refer to different values of the SNR. When the SNR value is lower than the convergence value, the BER is almost constant with SNR. Then, the curves become steeper and steeper versus number of iterations as long as the SNR increases. Comparing PCCC with SCCC we also notice the different behavior with respect to the error floor, which is evident at BER=$10^{-6}$ for the PCCC, while it does not show up for the SCCC.


**E. Comparison between PCCC and SCCC**

We have already seen in Section III that the interleaver gain of SCCC can be significantly larger than that of PCCC, depending on the free distance of the outer code. In general, PCCC show a better convergence abscissa than SCCC, but pay this advantage with a more pronounced error floor. This is clearly visible in Figure 58, where we compare the BER performance of iterative decoders for rate 1/3 PCCC (dashed red lines) and SCCC (solid green lines) for different number of iterations. Both concatenated encoders use 4-state CCs and an information block size of 10,000. The results show the slight delay in convergence of the SCCC, compensated by the better behavior in terms of error floor.

**F. Other concatenated structures**

1) **Simulation results for variations of PCCC codes:** Examples of performance of variations of turbo codes (in Figures 59, 60, and 61) are compared in Figure 62. Performance of rate 1/4 and rate 1/15 turbo codes are also shown in the same Figure, taken from [47].
Fig. 52. BER performance of an iterative decoder for a rate 1/3 PCCC with 2-state recursive constituent encoders versus $E_b/N_0$. Iterations from 1 to 10 are reported.

Fig. 53. BER performance of an iterative decoder for a rate 1/3 PCCC with 4-state recursive constituent encoders versus $E_b/N_0$. Iterations from 1 to 10 are reported.
Fig. 54. BER performance of an iterative decoder for a rate 1/3 PCCC with 8-state recursive constituent encoders versus $E_b/N_0$. Iterations from 1 to 10 are reported.

Fig. 55. Performance of an iterative decoder for a rate 1/3 SCCC with 4-state recursive outer and inner constituent encoders. Iterations from 1 to 10 are reported.
Fig. 56. BER performance of an iterative decoder for a rate 1/3 PCCC with 4-state recursive upper and lower constituent encoders. The horizontal axis here refers to the number of iterations, while the parameter for different curves is the $E_b/N_0$.

Fig. 57. BER performance of an iterative decoder for a rate 1/3 SCCC with 4-state recursive outer and inner constituent encoders. The horizontal axis here refers to the number of iterations, while the parameter for different curves is the $E_b/N_0$. 
Fig. 58. Comparison of BER performance of iterative decoders for rate 1/3 PCCC (dashed red lines) and SCCC (solid green lines). Both concatenated encoders use 4-state constituent encoder and an information block size of 10,000.

Fig. 59. Rate 2/4 turbo encoder A

Fig. 60. Rate 1/2 turbo encoder B
Fig. 61.  Rate 1/3 turbo encoder C

Fig. 62.  Performance of turbo codes for codes A, B, C shown above and rate 1/4, 1/15 codes.
All concatenated codes are constructed from two 4-state codes with:
- Input information block=16384 bits
- No. of iterations=12

Input information block=16384 bits

2) **simulation results for variations of SCCC codes:** Examples of performance of serially concatenated codes are shown in the following Figures 63, 64, 65 for various block sizes. Figure 63 has been borrowed from [48].
Fig. 64. Performance of SCCC with a two 4-state codes for various iterations, input block=256

Fig. 65. Performance comparison of two SCCC’s with a two 4-state codes, input block=256
3) *simulation results for multiple turbo codes (DPCCC)*: Examples of performance of DPCCC (multiple turbo codes) are shown in Figure 66, and Figure 67.
Fig. 66. Performance of DPCCC with three 4-state codes for input block=4096
Fig. 67. Performance of DPCCC with three codes for short input blocks
4) Simulation results for Hybrid concatenated codes (HCCC): A rate 1/4 HCCC code with three 4-state constituent convolutional codes was simulated over an AWGN channel for input block of \(N=16384\) bits.

The bit error probability versus the number of iterations for a given \(E_b/N_o\) in dB as a parameter is shown in Figure 68. The same results are also plotted as bit error probability versus \(E_b/N_o\) in dB, for a given number of iterations as a parameter, in Figure 69.

As it is seen from Figure 68 at \(E_b/N_o=0.1\) dB, a bit error probability of \(10^{-5}\) can be achieved with 19 iterations.

Thus at this bit error probability the performance is within 0.9 dB from the Shannon limit for rate 1/4 codes, over binary input AWGN channels. An optimum rate 1/4 PCCC (turbo code) with two 4-state code at \(10^{-5}\) requires \(E_b/N_o=0.4\) dB with 18 iterations and input block of \(N=16384\) bits.

The main advantage of HCCC over PCCC as can be seen from Figure 69 occurs at low bit error probability (less than \(10^{-6}\)). Even if we increase the number of states of the PCCC constituent codes to 8 states the HCCC with three 4-state codes outperforms the PCCC at low bit error rates. This also can be predicted by analysis. At high bit error probabilities HCCC and SCCC have similar performance

However, at very low bit error probabilities HCCC outperforms SCCC since, the interleaving gain is \(N^{-4}\) for HCCC and \(N^{-3}\), for SCCC (a factor of 16384), while \(h(\alpha_M) = 11\) in both cases. To obtain the results in Figure 68 and Figure 69, \(5 \times 10^8\) random bits were simulated for each point.
Fig. 68. Simulation results for hybrid concatenated convolutional codes (HCCC) with $N = 16384$: BER vs. number of iterations at different bit SNRs.
Fig. 69. Simulation results for hybrid concatenated convolutional codes (HCCC) with $N = 16384$: BER vs. $E_b/N_0$ at different number of iterations.
5) **Simulation results for Self Concatenated Codes**: Consider a rate 1/4 self concatenated code with 4-state recursive convolutional code $G(D) = [1, \frac{1+D^2}{1+D+D^2}]$, $q = 3$, and input block $N$ as shown in the Figure 70.

The simulation result for this code is shown in Figure 71. One use of SISO per input block was considered as one iteration.
6) Simulation results for Repeat-Accumulate (RA) Codes: Consider a rate 1/3 ($q = 3$) and rate 1/4 ($q = 4$) repeat accumulate (RA) codes with various block sizes. The simulation result for these codes using the iterative decoder is shown in Figure 72.
7) Simulation results for Repeat-Accumulate-Accumulate (RAA) Codes: Consider a rate 1/3 \((q = 3)\) repeat accumulate accumulate (RAA) code in Figure 73. The RAA code is special case of double sccc code. The simulation results for RAA code is shown in Figure 73 and it is compared with RA code and a serial concatenation of repeat-3 and a rate-1 4-state (RDD) code.
REFERENCES


