Chapter 7

Convolutional turbo codes

The error correction capability of a convolutional code increases when the length of the encoding register increases. This is shown in Figure 7.1, which provides the performance of four RSC codes with respective memories $\nu = 2, 4, 6$ and 8, for rates 1/2, 2/3, 3/4 and 4/5, decoded according to the MAP algorithm. For each of the rates, the error correction capability improves with the increase in ν, above a certain signal to noise ratio that we can assimilate almost perfectly with the theoretical limit calculated in Chapter 3 and identified here by an arrow. To satisfy the most common applications of channel coding, a memory of the order of 30 or 40 would be necessary (from a certain length of register and for a coding rate 1/2, the minimum Hamming distance of a convolutional code with memory ν is of the order of ν). If we knew how to easily decode a convolutional code with over a billion states, we would no longer speak much about channel coding and this book would not exist.

A turbo code is a coding trick, aiming to imitate a convolutional code with a large memory ν. It is built on the principle of the saying *divide and rule*, that is, by associating several small RSC codes whose particular decodings are of reasonable complexity. A judicious exchange of information between the elementary decoders enables the composite decoder to approximate the performance of maximum likelihood decoding.

7.1 The history of turbo codes

The invention of turbo codes is not the outcome of a mathematical development. It is the result of an intuitive experimental approach whose origin can be found in the work of several European researchers: Gerard Battail, Joachim Hagenauer and Peter Hoeher who, at the end of the 80s [7.8, 7.7, 7.31, 7.30] highlighted the interest of probabilistic processing in receivers. Others before them, mainly in the United States: Peter Elias [7.25], Michael Tanner [7.45], Robert Gallager

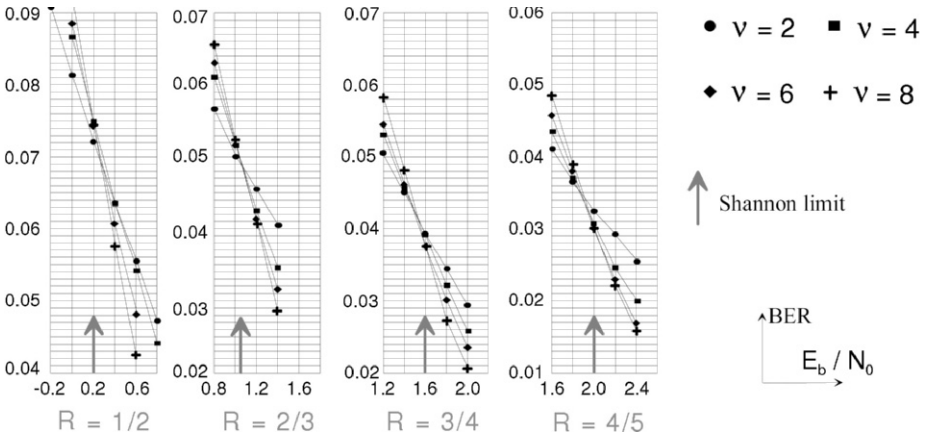

Figure 7.1 – Performance of recursive systematic convolutional codes (RSC) for different rates and four values of the memory of code ν. Comparison with Shannon limits.

[7.26], etc. had earlier imagined procedures for coding and decoding that were the forerunners of turbo codes.

In a laboratory at École Nationale Supérieure des Télécommunications de Bretagne (Telecom Bretagne), Claude Berrou and Patrick Adde were attempting to transcribe the Viterbi algorithm with weighted input (SOVA: *Soft-Output Viterbi Algorithm*) proposed in [7.7], into MOS transistors, in the simplest possible way. A suitable solution [7.10] was found after two years which enabled these researchers to form an opinion about probabilistic decoding. Claude Berrou, then Alain Glavieux, pursued the study and observed, after Gerard Battail, that a decoder with weighted input and output could be considered as a signal to noise ratio amplifier. This encouraged them to implement the concepts commonly used in amplifiers, mainly feedback. Perfecting turbo codes involved many very pragmatic stages and also the introduction of neologisms, like "parallel concatenation" or "extrinsic information", nowadays common in information theory jargon. The publication in 1993 of the first results [7.14], with a performance 0,5 dB from the Shannon limit, shook the coding community. A gain of almost 3 dB, compared to solutions existing at that time, had been found by a small team that was not only unknown, but also French (France, a country known for its mathematical rigour, *versus* turbo codes, an empirical invention to say the least). There followed a very distinct evolution in habits, as underlined by A. R. Calderbank in [7.20] (p. 2573): "*It is interesting to observe that the search for theoretical understanding of turbo codes has transformed coding theorists into experimental scientists*"

[7.13] presents a chronology describing the successive ideas that appeared in the search to perfect turbo codes. This new coding and decoding technique was

first baptized turbo-code, with a hyphen to show that it was a code decoded in a turbo way (by analogy with the turbo engine that uses exhaust gas to increase its power). As the hyphen is not used much in English, it became *turbo code*, that is, a "turbo" code, which does not mean very much. In French today, turbo code is written as a single word: *turbocode*.

7.2 Multiple concatenation of RSC codes

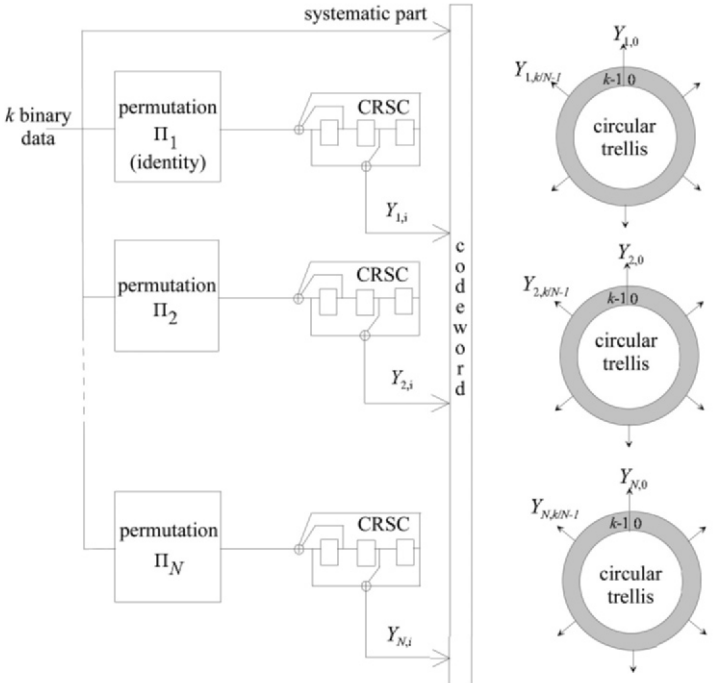

Figure 7.2 – Multiple parallel concatenation of circular recursive systematic convolutional (CRSC) codes. Each encoder produces k/N redundancy symbols uniformly distributed on the circular trellis. Global coding rate: 1/2.

Since the seminal work of Shannon, random codes have always been a reference for error correction coding (see Section 3.1.5). The systematic random coding of a block of k information bits, leading to a codeword of length n, can, as the first step and once for all, involve drawing at random and memorizing k binary markers containing $n - k$ bits, whose memorization address is denoted i ($0 \leq i \leq k - 1$). The redundancy associated with any block of information is then formed by the modulo 2 sum of all the markers whose address i is such

that the i-th information bit equals 1. In other words, the k markers are the bases of a vector space of dimension k. The codeword is finally made up of the concatenation of the k information bits and of the $n-k$ redundancy bits. The rate R of the code is k/n. This very simple construction of the codeword relies on the linearity property of the addition and leads to high minimum distances for sufficiently large values of $n-k$. Because two codewords are different by at least one information bit and the redundancy is drawn at random, the average minimum distance is $1+\frac{n-k}{2}$. However, the minimum distance of this code being a random variable, its different realizations can be lower than this value. A simple realistic approximation of the effective minimum distance is $\frac{n-k}{4}$.

A way to build an almost random encoder is presented in Figure 7.2. It is a multiple parallel concatenation of circular recursive systematic convolutional codes (CRSC, see Chapter 5) [7.12]. The sequence of k binary data is coded N times by N CRSC encoders, in a different order each time. The permutations Π_j are drawn at random, except the first one that can be the identity permutation. Each elementary encoder produces $\frac{k}{N}$ redundancy symbols (N being a divisor of k), the global rate of the concatenated code being $1/2$.

The proportion of input sequences of a recursive encoder built from a pseudo-random generator with memory ν, initially positioned in state 0, which return the register back to the same state at the end of the coding, is:

$$p_1 = 2^{-\nu} \tag{7.1}$$

since there are 2^ν possible return states, with the same probability. These sequences, called *Return To Zero* (RTZ) sequences,(see Chapter 5), are linear combinations of the minimum RTZ sequence, which is given by the recursivity polynomial of the generator ($1+D+D^3$ in the case of Figure 7.2).

The proportion of RTZ sequences for the multi-dimensional encoder is lowered to:

$$p_N = 2^{-N\nu} \tag{7.2}$$

since the sequence must, after each permutation, remain RTZ for the N encoders.

The other sequences, with proportion $1-p_N$, produce codewords that have a distance d satisfying:

$$d > \frac{k}{2N} \tag{7.3}$$

This worst case value assumes that a single permuted sequence is not RTZ and that redundancy Y takes the value 1, every other time on average, on the corresponding circle. If we take $N=8$ and $\nu=3$ for example, we obtain $p_8 \approx 10^{-7}$ and, for sequences to encode of length $k=1024$, we have $d_{min}=64$, which is a sufficient minimum distance if we refer to the curves of Figure 3.6.

Random coding can thus be approximated by using small codes and random permutations. The decoding can be performed following the turbo principle, described in Section 7.4 for $N=2$. The scheme of Figure 7.2 is, however, not used in practice, for reasons linked to the performance and complexity of the

decoding. First, the convergence threshold of the turbo decoder, that is, the signal to noise ratio from which the turbo decoder can begin to correct most of the errors, degrades when the dimension of the concatenation increases. Indeed, the very principle of turbo decoding means considering the elementary codes one after the other, iteratively. As their redundancy rate decreases when the dimension of the composite code increases, the first steps in the decoding are penalized compared to a concatenated code with a simple dimension 2. Then, the complexity and the latency of the decoder are proportional to the number of elementary encoders.

7.3 Turbo codes

Fortunately, concerning the above, it is not necessary to carry dimension N to a high value. By replacing the random permutation Π_2 by a judiciously elaborated permutation, good performance can be obtained by limiting ourselves to a dimension $N = 2$. That is the principle of turbo codes.

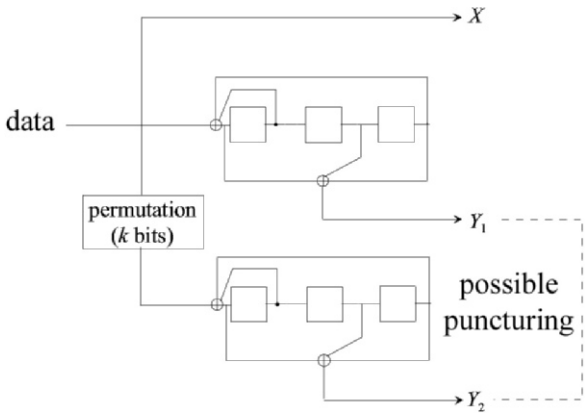

Figure 7.3 – A binary turbo code with memory $\nu = 3$ using identical elementary RSC encoders (polynomials 15, 13). The natural coding rate of the turbo code, without puncturing, is 1/3.

Figure 7.3 presents a turbo code in its most classical version [7.14]. The binary input message, of length k, is encoded in its natural order and in a permuted order by two RSC encoders called C_1 and C_2, which can be terminated or not. In this example, the two elementary encoders are identical (generator polynomials 15 for the recursivity and 13 for the construction of the redundancy) but this is not a necessity. The natural coding rate, without puncturing, is 1/3. To obtain higher rates, redundancy symbols Y_1 and Y_2 are punctured. Another way to have higher rates is to adopt m-binary codes (see 7.5.2).

As the permutation function (Π) concerns a message of finite size k, the turbo code is by construction a block code. However, to distinguish it from concatenated algebraic codes decoded in a "turbo" way, like product codes which were later called *block turbo codes*, this turbo coding scheme is called a *convolutional* code or, more technically, a *Parallel Concatenated Convolutional Code* (PCCC).

Arguments in favour of this coding scheme (some of which have already been introduced in Chapter 6) are the following:

1. A decoder for convolutional codes is vulnerable to errors arriving in packets. Coding the message twice, following two different orders (before and after permutation), makes fairly improbable the simultaneous appearance of error packets at the input of the decoders of C_1 and C_2. If there are errors grouped at the input of the decoder of C_1, the permutation disperses them over time and they become isolated errors that are easy for the decoder of C_2 to correct. This reasoning also holds for error packets at the input of this second decoder, which correspond, before permutation, to isolated errors. Thus two-dimensional coding, on at least one of the two dimensions, greatly reduces the vulnerability of convolutional coding concerning grouped perturbations. But which of the two decoders should be relied on to take the final decision? No criterion allows us to be more confident about one or the other. The answer is given by the "turbo" algorithm that avoids having to make this choice. This algorithm implements exchanges of probabilities between the two decoders and constrains them to converge, during these exchanges, towards the same decisions.

2. As we saw in Section 6.1, parallel concatenation leads to a higher coding rate than that of serial concatenation. Parallel concatenation is therefore more favourable when signal to noise ratios close to the theoretical limits are considered, with average error rates targeted. It can be different when very low error rate are sought for since the MHD of a serial concatenated code can be larger.

3. Parallel concatenation uses systematic codes and at least one of these codes must be recursive, for reasons also described in Section 6.1

4. Elementary codes are small codes: codes with 16, 8, or even 4 states. Even if the decoding implements repeated probabilistic processing, it remains of reasonable complexity.

Figure 7.4 presents turbo codes used in practice and Table 7.2 lists some industrial applications. For a detailed overview of the applications of turbo and LDPC codes see [7.29]. The parameters defining a particular turbo code are the following:

a– m is the number of bits in the symbols applied to the turbo encoder. The applications known to this day consider binary ($m = 1$) or double-binary ($m = 2$) symbols.

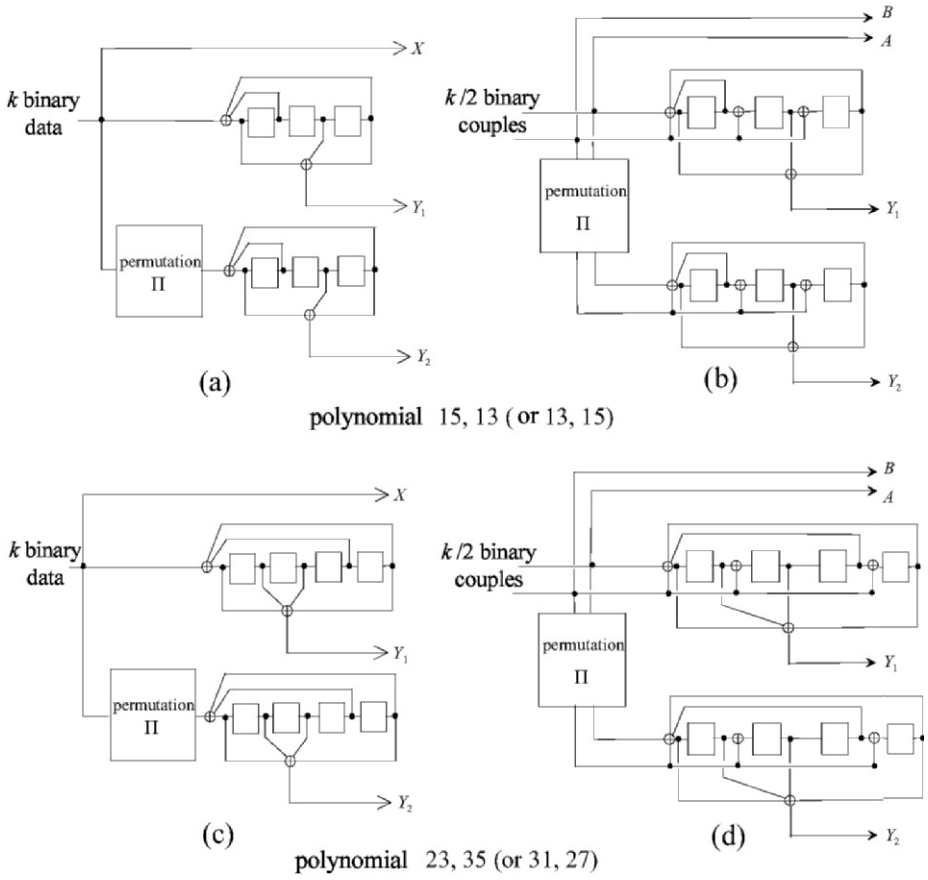

Figure 7.4 – Turbo codes used in practice.

b– Each of the two elementary encoders C_1 and C_2 is characterized by

- its code memory ν
- its generator polynomials for recursivity and redundancy
- its rate

The values of ν are in practice lower than or equal to 4. The generator polynomials are generally those that we use for classical convolutional codes and that were the subject of much literature in the 1980s and 1990s.

c– The way in which we perform the permutation is important when the targeted binary error rate is lower than around 10^{-5}. Above this value, per-

Application	Turbo code	Termination	Polynomials	Rates
CCSDS (deep space)	binary, 16 states	tail bits	23, 33, 25, 37	1/6, 1/4, 1/3, 1/2
UMTS, CDMA2000 (mobile 3G)	binary, 8 states	tail bits	13, 15, 17	1/4, 1/3, 1/2
DVB-RCS (Return Channel on Satellite)	double-binary, 8 states	circular	15, 13	from 1/3 to 6/7
DVB-RCT (Return Channel on Terrestrial)	double-binary, 8 states	circular	15, 13	1/2, 3/4
Inmarsat (M4)	binary, 16 states	none	23, 35	1/2
Eutelsat (skyplex)	double-binary, 8 states	circular	15, 13	4/5, 6/7
IEEE 802.16 (WiMAX)	double-binary, 8 states	circular	15, 13	from 1/2 to 7/8

Table 7.2 – Standardized applications of convolutional turbo codes.

formance is not very sensitive to permutation, on condition of course that it respects at least the principle of dispersion (which, for example, can be a regular permutation). For a low or very low targeted error rate, performance is dictated by the minimum distance of the code and the latter is highly dependent on the permutation Π.

d– The puncturing pattern must be the as regular as possible, in the same way as for classical convolutional codes. However, it can be advantageous to have a slightly irregular puncturing pattern when we are looking for very low error rates and when the puncturing period is a divisor of the period of the generator polynomial of recursivity or parity.

Puncturing is performed classically on the redundancy symbols. It can be envisaged instead to puncture the information symbols, in order to increase the minimum distance of the code. This is done to the detriment of the convergence threshold of the turbo decoder. From this point of view, in fact, puncturing

data shared by the two decoders is more penalizing than puncturing data that are only useful to one of the decoders.

What must be closely considered when building a turbo code and decoding it, are the RTZ sequences, whose output weights limit the minimum distance of the code and fix its asymptotic performance. In what follows it will be assumed that the error patterns that are not RTZ do not contribute to the MHD of the turbo code and will therefore not have to be considered.

7.3.1 Termination of constituent codes

For a turbo code, there are two trellises to be terminated and the solutions presented in Section 5.5.1 can be envisaged:

Doing nothing in particular concerning the terminal states: the data situated at the end of the block, in either the natural order or in the permuted order, are thus less well protected. This leads to a decrease in the asymptotic gain, but this degradation, which is a function of the size of the block, may be compatible with some applications. It should be noted that non-termination of the trellis penalizes the PER (Packet Error Rates) more greatly than the BER.

Terminating the trellis of one or both elementary codes using tail bits: CCSDS [7.4] and UMTS [7.3] standards use this technique. The bits ensuring the termination of one of the two trellises are not used in the other encoder. These bits are therefore not turbo encoded, which leads, but to a lesser degree, to the same drawbacks as those presented in the previous case. Moreover, the transmission of the tail bits causes a decrease in the coding rate and therefore in the spectral efficiency.

Using interleaving enables the automatic termination of the trellis: it is possible to close the trellis of a turbo code automatically, without adding any tail bits, by slightly transforming the coding scheme (self-concatenation) and by using interleaving that respects certain periodicity rules. This solution described in [7.15] does not decrease the spectral efficiency but imposes constraints on the interleaving which makes it difficult to control performance at low error rates.

Adopting circular encoding: a circular encoder for convolutional codes guarantees that the initial state and the final state of the register are identical. The trellis then takes the form of a circle, which, from the point of view of the decoder, can be considered as a trellis with infinite length [7.32, 7.48]. This termination process, already known as *tail-biting* for non-recursive codes, offers two main advantages:

- Unlike the other techniques, circular termination does not present any edge effects: all the bits of the message are protected in the same way and are all doubly encoded by the turbo code. Therefore, during the design of the permutation, there is no need to give special importance to such and such a bit, which leads to simpler permutation models.

- The sequences that are not RTZ have an influence on the whole circle: on average, one parity symbol out of two is modified along the block. For typical values of k (a few hundred or more), the corresponding output weight is therefore very high and these error patterns do not contribute to the MHD of the code, as already mentioned at the end of the previous section. Without termination or with termination using tail bits, only the part of the block after the beginning of the non-RTZ sequence has any effect on the parity symbols.

To these two advantages we can, of course, add the interest of having to transmit no additional information about termination and therefore losing nothing in spectral efficiency.

The circular termination technique was chosen for the DVB-RCS and DVB-RCT [7.2, 7.1] standards, for example.

7.3.2 The permutation function

Whether we call it interleaving or permutation, the technique that involves dispersing the data over time has always been very useful in digital communications. For example, we use it profitably to reduce the effects of more or less long attenuations in transmissions affected by fading and, more generally, in situations where perturbations can alter consecutive symbols. In the case of turbo codes too, permutation allows us to efficiently combat the appearance of error packets, on at least one of the dimensions of the composite code. But its role does not stop there: in close relation with the properties of the constituent codes, it also determines the minimum distance of the concatenated code.

Let us consider the turbo code presented in Figure 7.3. The worst permutation that we can use is, of course, identity permutation, which minimizes the diversity of the coding (we then have $Y_1 = Y_2$). On the other hand, the best permutation that we could imagine, but that probably does not exist [7.42], would enable the concatenated code to be equivalent to a sequential machine of which the number of irreducible states would be 2^{k+6}. There are indeed $k + 6$ binary memorization elements in the structure: k for the permutation memory and 6 for the two convolutional codes. If we could assimilate this sequential machine to a convolutional encoder and for common values of k, the number of corresponding states would be very large, in any case large enough to guarantee a large minimum distance. For example, a convolutional encoder with a code memory of 60 (10^{18} states !) shows a free distance of the order of a hundred (for $R = 1/2$), which is quite sufficient.

Thus, from the worst to the best permutation, there is a wide choice and we have not yet discovered any perfect permutation. Having said that, good permutations have been defined even so in order to elaborate standardized turbo coding schemes.

There are two ways to specify a permutation, the first by equations linking addresses before and after permutation, the second by a *look-up table* providing the correspondence between addresses. The first is preferable from the point of view of simplicity in the specification of the turbo code (standardization committees are sensitive to this aspect) but the second can lead to better results since the degree of freedom is generally larger when designing the permutation.

Regular permutation

The point of departure when designing interleaving is regular permutation, which is described in Figure 7.5 in two different forms. The first assumes that the block containing k bits can be organized as a table of M rows and N columns. The interleaving then involves writing the data in an *ad hoc* memory, row by row, and reading them column by column (Figure 7.5(a)). The second applies without any hypothesis about the value of k. After writing the data in a linear memory (address i, $0 \leq i \leq k-1$), the block is assimilated to a circle, the two extremities ($i = 0$ and $i = k-1$) then being adjacent (Figure 7.5(b)). The binary data are then extracted in such a way that the j-th datum read has been previously written in position i, with value:

$$i = \Pi(j) = Pj + i_0 \bmod k \tag{7.4}$$

where P is a prime integer with k and i_0 is the index of departure[1].

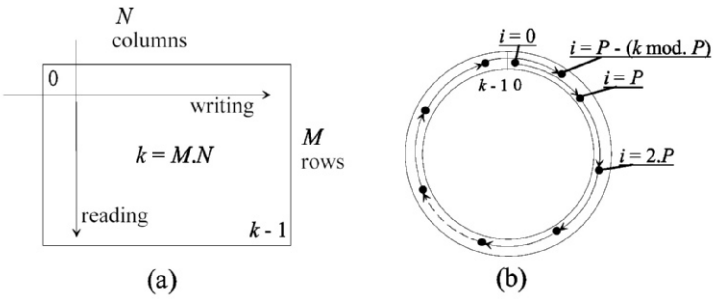

Figure 7.5 – Regular permutation in rectangular (a) and circular (b) form.

For circular permutation, let us define the accumulated spatial distance $S(j_1, j_2)$ as the sum of the two spatial distances separating two bits, before and after permutation, whose reading indices are j_1 and j_2:

$$S(j_1, j_2) = f(j_1, j_2) + f(\Pi(j_1), \Pi(j_2)) \tag{7.5}$$

[1] The permutation can, of course, be defined in a reciprocal form, that is, j function of i. It is a convention that is to be adopted once and for all, and the one that we have chosen is compatible with most standardized turbo codes.

where:

$$f(u,v) = \min\left\{|u-v|, k-|u-v|\right\} \tag{7.6}$$

Function f is introduced to take into account the circular nature of the addresses. Finally, we call S_{min} the smallest of the values of $S(j_1, j_2)$, for all the possible pairs j_1 and j_2:

$$S_{\min} = \min_{j_1, j_2}\left\{S(j_1, j_2)\right\} \tag{7.7}$$

It is proved in [7.19] that an upper bound of S_{min} is:

$$\sup S_{\min} = \sqrt{2k} \tag{7.8}$$

This upper bound is only reached in the case of a regular permutation and with conditions:

$$P = P_0 = \sqrt{2k} \tag{7.9}$$

and:

$$k = \frac{P_0}{2} \bmod P_0 \tag{7.10}$$

Let us now consider a sequence of any weight that, after permutation, can be written:

$$\tilde{d}(D) = \sum_{j=0}^{k-1} a_j D^j \tag{7.11}$$

where a_j can take the binary value 0 (no error) or 1 (one error) and, before permutation:

$$d(D) = \sum_{i=0}^{k-1} a_i D^i = \sum_{j=0}^{k-1} a_{\Pi(j)} D^{\Pi(j)} \tag{7.12}$$

We denote j_{min} and j_{max} the j indices corresponding to the first and last non-null values a_j in $\tilde{d}(D)$. Similarly, we define i_{min} and i_{max} for sequence $d(D)$. Then, the regular permutation satisfying (7.9) and (7.10) guarantees the property:

$$(j_{\max} - j_{\min}) + (i_{\max} - i_{\min}) > \sqrt{2k} \tag{7.13}$$

This is because $d(D)$ and $\tilde{d}(D)$, both considered between min and max indices, contain at least 2 bits whose accumulated spatial distance, as defined by (7.5), is maximum and equal to $\sqrt{2k}$. We must now consider two cases:

- sequences $d(D)$ and $\tilde{d}(D)$ are both of the *simple* RTZ type, that is, they begin in state 0 of the encoder and return to it once, at the end. The parity bits produced by these sequences are statistically 1s, every other time. Taking into account (7.13), for common values of k ($k > 100$), the redundancy weights are high and these RTZ sequences do not contribute to the MHD of the turbo code.
- at least one of sequences $d(D)$ and $\tilde{d}(D)$ is of the *multiple* RTZ type, that is, it corresponds to the encoder passing several times through state 0. If these passes through state 0 are long, the parity associated with the sequence may have reduced weight and the associated distance may be low. Generally, in this type of situation, the sequences before and after permutation are both multiple RTZ.

The performance of a turbo code, at low error rates, is closely linked with the presence of multiple RTZ patterns and regular permutation is not a good solution for eliminating these patterns.

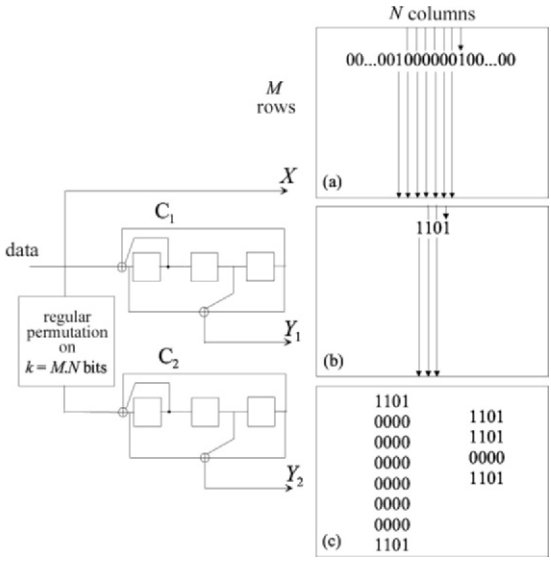

Figure 7.6 – Possible error patterns of weight 2, 3, 6 or 9 with a turbo code whose elementary encoders have a period 7 and with regular permutation.

Necessity for disorder

Again assuming that the error patterns that are not RTZ have weights high enough not to have any effect on performance, an ideal permutation for a turbo code could be defined by the following rule:

If a sequence is of the RTZ type before permutation, then it is no longer so after permutation and vice-versa.

The rule above is impossible to satisfy in practice and a more realistic objective is:

If a sequence is of the RTZ type before permutation, then it is no longer so after permutation or it has become a simple long RTZ sequence and vice-versa.

The dilemma in designing good permutations for turbo codes lies in the need to satisfy this objective for two distinct classes of input sequences that require opposing types of processing: simple RTZ sequences and multiple RTZ sequences, as defined above. To illustrate this problem, consider a rate 1/3 turbo code, with regular rectangular permutation (writing in M rows, lecture in N columns) over blocks of $k = MN$ bits (Figure 7.6). Elementary encoders are encoders with 8 states whose period is 7 (recursivity generator 15).

The first pattern (a) of Figure 7.6 concerns a sequence of possible errors with input weight $w = 2$: 10000001 for code C_1, that we can also call the horizontal code. This is the RTZ minimum sequence with weight 2 for the encoder considered. The redundancy produced by this encoder is of weight 6 (exactly: 11001111). The redundancy produced by the vertical encoder C_2, for which the sequence considered is also RTZ (its length is a multiple of 7), is much more informative because it is simple RTZ and produced over seven columns. Assuming that Y_2 is equal to 1 every other time on average, the weight of this redundancy is around $w(Y_2) \approx \frac{7M}{2}$. When we make k tend towards infinity via the values of M and N ($M \approx N \approx \sqrt{k}$), the redundancy produced by one of the two codes, for this type of pattern, also tends towards infinity. We then say that the code is *good*.

The second pattern (b) is that of the minimum RTZ sequence of input weight 3. Here again, the redundancy is poor on the first dimension and much more informative on the second. The conclusions are the same as above.

The other two diagrams in (c) present examples of multiple RTZ sequences, made up of short RTZ sequences on each of the two dimensions. The input weights are 6 and 9. The distances associated with these patterns (respectively 30 and 27 for this rate 1/3 code) are not generally sufficient to ensure good performance at low error rates. Moreover, these distances are independent of block size and therefore, in relation to the patterns considered, the code is not *good*.

Regular permutation is therefore a good permutation for the class of simple RTZ error patterns. For multiple RTZ patterns, however, regular permutation is not appropriate. A good permutation must "break" the regularity of rectangular composite patterns like those of Figure 7.6(c), by introducing some disorder. But this must not be done to the detriment of the patterns for which regular permutation is good. The disorder must therefore be managed well! Therein lies the whole problem when looking for a permutation that must lead to a high

enough minimum distance. A good permutation cannot be found independently of the properties of elementary codes, of their RTZ patterns, their periodicities, etc.

Intra-symbol disorder

When the elementary codes are m-binary codes, we can introduce a certain disorder into the permutation of a turbo code without however removing its regular nature! To do this, in addition to intersymbol classical permutation, we implement intra-symbol permutation, that is, a non-regular modification of the content of the symbols of m bits, before coding by the second code [7.11]. We briefly develop this idea with the example of double-binary turbo codes ($m = 2$).

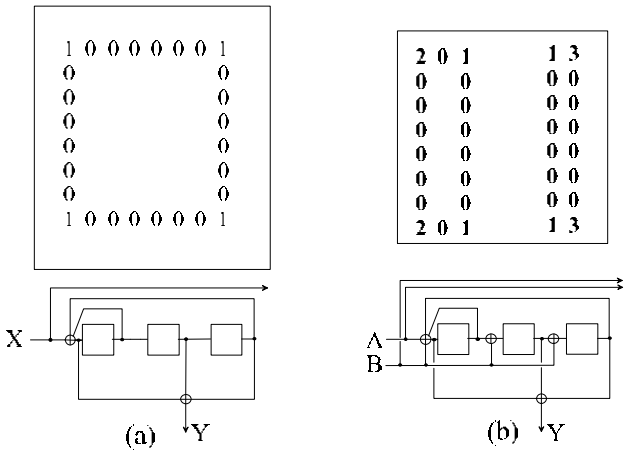

Figure 7.7 – Possible error patterns with binary (a) and double-binary (b) turbo codes and regular permutation.

Figure 7.7(a) presents the minimum pattern of errors with weight $w = 4$, again using the code of Figure 7.6. It is a square pattern whose side is equal to the period of the pseudo-random generator with polynomial 15, that is, 7. It has already been mentioned that some disorder had to be introduced into the permutation to "break" this kind of error pattern but without altering the properties of the regular permutation in relation to patterns with weight 2 and 3, which is not easy. If, as an elementary encoder, we replace the binary encoder by a double-binary encoder, the error patterns to consider are no longer made up of bits but of couples of bits. Figure 7.7(b) gives an example of a double-binary encoder and of possible error patterns, when the permutation is regular. The couples are numbered from **0** to **3**, according to the following correspondence:

$$(0,0) : \mathbf{0}; \quad (0,1) : \mathbf{1}; \quad (1,0) : \mathbf{2}; \quad (1,1) : \mathbf{3}$$

The periodicities of the double-binary encoder are resumed in the diagram of Figure 7.8. There we can find all the combinations of pairs of couples of the RTZ type. For example, if the encoder, initialized in state 0, is fed by the successive couples **1** and **3**, it immediately returns to state 0. It is the same for the sequences **201** or **2003** or **3000001**, for example.

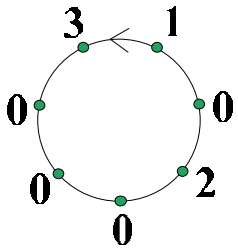

Figure 7.8 – Periodicities of the double-binary encoder of Figure 7.7(b). The four input couples $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ are denoted 0, 1, 2 and 3, respectively. This diagram gives all the combinations of pairs of couples of the RTZ type.

Figure 7.7(b) gives two examples of rectangular, minimum size error patterns. First note that the perimeter of these patterns is larger than half the perimeter of the square of Figure 7.7(a). Now, for a same coding rate, the redundancy of a double-binary code is twice as dense as that of a binary code. We thus deduce that the distances of the double-binary error patterns will naturally be larger, everything else being equal, than those of binary error patterns. Moreover, there is a simple way to eliminate these elementary patterns.

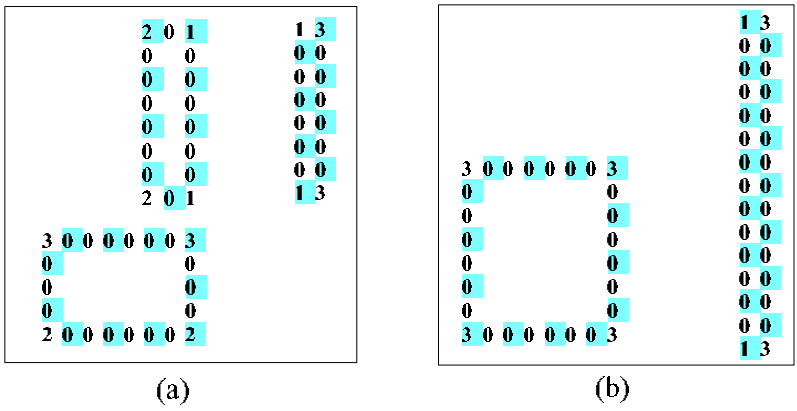

Figure 7.9 – The couples of the grey boxes are inverted before the second (vertical) encoding. 1 becomes 2, 2 becomes 1; 0 and 3 remain unchanged. The patterns of Figure 7.7(b), redrawn in (a), are no longer possible error patterns. But those of (b) are, with distances 24 and 26 for coding rate 1/2.

Assume, for example, that the couples are inverted (**1** becomes **2** and vice-versa), every other time, before being applied to the vertical encoder. Then the error patterns presented in Figure 7.9(a) no longer exist; for example, although **30002** does represent an RTZ sequence for the encoder considered, **30001** no longer does. Thus, many of the error patterns, in particular the smallest, disappear thanks to the disorder introduced inside the symbols. Figure 7.9(b) gives two examples of patterns that the periodic inversion does not modify. The corresponding distances are high enough (24 and 26 for a rate 1/2) not to pose a problem for small or average block sizes. For long blocks (several thousand bits), additional intersymbol disorder, of low intensity, can be added to the intra-symbol non-uniformity, to obtain even higher minimum distances.

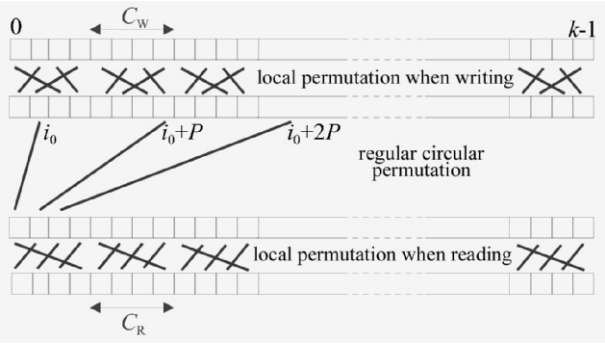

Figure 7.10 – Permutation of the DRP type. This is a circular regular permutation to which local permutations before writing and after reading are added.

Irregular permutations

In this section, we will not describe all the irregular permutations that have been imagined so far, and that have been the subject of numerous publications and several book chapters (see [7.40, 7.34] for example). We prefer to present what seems, for the moment, to be both the simplest and the most efficient type of permutation. These are almost regular circular permutations, called almost regular permutation (ARP)[7.17] or dithered relatively prime (*DRP*) [7.21] permutations, depending on their authors. In all cases, the idea is not to stray too far away from the regular permutation, which is well adapted to simple RTZ error patterns and to instil some small, controlled disorder to counter multiple RTZ error patterns.

Figure 7.10 gives an example, taken from [7.21], of what this small disorder can be. Before the circular regular permutation is performed, the bits undergo local permutation. This permutation is performed in groups of C_W bits. C_W, which is the *writing cycle* disorder, is a divisor of length k of the message. Similarly, a local C_R *reading cycle* permutation is applied before the final reading.

In practice, C_W and C_R can be identical values $C_W = C_R = C$, typically 4 or 8. This way of introducing disorder, in small local fluctuations, does not significantly decrease the accumulated spatial distance, whose maximum value is $\sqrt{2k}$. However, it enables us to suppress the error patterns comparable to those of Figures 7.6(b) and 7.7(c) on condition that the heights and widths of these patterns are not both multiples of C.

Another way to perturb the regular permutation in a controlled way is shown in Figure 7.11. The permutation is represented here in rectangular form, visually more accessible, but it can also be very well applied to circular permutation. One piece of information (bit or symbol) is placed where each row and column cross. With regular permutation, these data are therefore memorized row by row and read column by column. In Figure 7.11, the disorder is introduced by means of four displacement vectors $V_1, \cdots, V_4$ that are applied alternately during reading. These vectors have a small amplitude compared to the dimensions of the permutation matrix.

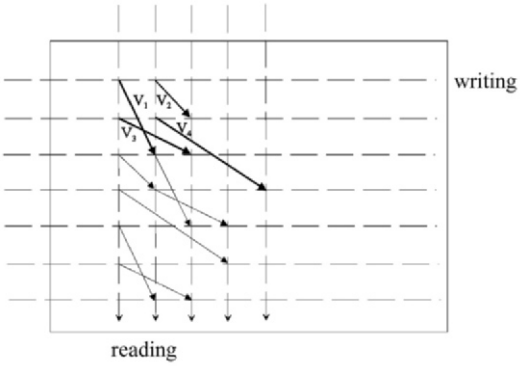

Figure 7.11 – Permutation of the ARP type, following [7.17].

The mathematical model associated with this almost regular permutation, in its circular form, is an extension of (7.4):

$$i = \Pi(j) \equiv Pj \ + Q(j) + i_0 \bmod k \tag{7.14}$$

If we choose

$$Q(j) = A(j)P + B(j) \tag{7.15}$$

where the positive integers $A(j)$ and $B(j)$ are periodic, with cycle C (divisor of k), then these values correspond to the positive shifts applied respectively before and after regular permutation. That is the difference between the permutation shown in Figure 7.10, in which the writing and reading perturbations are performed inside small groups of data and not by shifts.

For the permutation to really be a bijection, parameters $A(j)$ and $B(j)$ are not just any parameters. To ensure the existence of the permutation, there is one sufficient condition: all the parameters have to be multiples of C. This condition is not very restricting in relation to the efficiency of the permutation. (7.15) can then be rewritten in the form:

$$Q(j) = C(\alpha(j)P + \beta(j)) \tag{7.16}$$

where $\alpha(j)$ and $\beta(j)$ are more often than not small integers, with values 0 to 8. In addition, since the properties of a circular permutation are not modified by a simple rotation, one of the $Q(j)$ values can systematically be 0.

Two typical sets of Q values, with cycle 4 and $\alpha = 0$ or 1, are given below:

$$\begin{array}{l} \text{if } j = 0 \mod 4, \text{ then } Q = 0 \\ \text{if } j = 1 \mod 4, \text{ then } Q = 4P + 4\beta_1 \\ \text{if } j = 2 \mod 4, \text{ then } Q = 4\beta_2 \\ \text{if } j = 3 \mod 4, \text{ then } Q = 4P + 4\beta_3 \end{array} \tag{7.17}$$

$$\begin{array}{l} \text{if } j = 0 \mod 4, \text{ then } Q = 0 \\ \text{if } j = 1 \mod 4, \text{ then } Q = 4\beta_1 \\ \text{if } j = 2 \mod 4, \text{ then } Q = 4P + 4\beta_2 \\ \text{if } j = 3 \mod 4, \text{ then } Q = 4P + 4\beta_3 \end{array} \tag{7.18}$$

These models require the knowledge of only four parameters (P, β_1, β_2 and β_3), which can be determined using the procedure described in [7.17]. The utilization of m-binary codes (see Section 7.5), instead of binary codes, simply requires k to be replaced by k/m in Equation (7.14). In particular, the permutations defined for double-binary turbo codes ($m = 2$) of the DVB-RCS, DVB-RCT and WiMax standards are inspired by Equations (7.14) and (7.21).

Quadratic Permutation

Recently, Sun and Takeshita [7.41] proposed a new class of deterministic interleavers based on permutation polynomials (PP) over integer rings. The use of PP reduces the design of interleavers to simply a selection of polynomial coefficients. Furthermore, PP-based turbo codes have been shown to have a) good distance properties [7.38] which are desirable for lowering the error floor and b) a maximum contention-free property [7.43] which is desirable for parallel processing to allow high-speed hardware implementation of iterative turbo decoders.

A. Permutation Polynomials

Before addressing the quadratic PP, we will define the general form of a polynomial and discuss how to verify whether a polynomial is a PP over the ring of integers modulo N, $\mathbb{Z}_N$. Given an integer $N \geq 2$, a polynomial

$$f(x) = a_0 + a_1 + a_2 x 2 + \ldots + a_m x_m \text{ modulo } N \tag{7.19}$$

where the coefficients $a_0, a_1, a_2, \ldots, a_m$ and m are non-negative integers, is said to be permutation polynomial over $\mathbb{Z}_N$ when $f(x)$ permutes $\{0, 1, 2, \ldots, N-1\}$ [7.41]. Since we have modulo N operation, it is sufficient for the coefficients $a_0, a_1, a_2, \ldots, a_m$ to be in $\mathbb{Z}_N$. Let us recall that the formal derivative of the polynomial $f(x)$ is given by

$$f^{'}(x) = a_1 + 2a_2 x + 3a_3 x^2 + \ldots + m a_m x^{m-1} \text{ modulo } N \tag{7.20}$$

To verify whether a polynomial is a PP over $\mathbb{Z}_N$, let us discuss the following three cases a) the case $N = 2^n$, where n is an element of the positive integers $\mathbb{Z}_+$, b) the case $N = p^n$ where p is any prime number, and c) the case where N is an arbitrary element of $\mathbb{Z}_+$.

1. Case I ($N = 2^n$): a theorem in [7.36] states that $f(x)$ is a PP over the integer ring $\mathbb{Z}_2^n$ if and only if 1) a_1 is odd, 2) $a_2 + a_4 + a_6 + \ldots$ is even, and 3) $a_3 + a_5 + a_7 + \ldots$ is even.

   Example 1 for $N = 2^3 = 8 : f(x) = 1 + 5x + x^2 + x^3 + x^5 + 3x^6$ is a PP over $\mathbb{Z}_{N=8}$ because it maps the sequence $\{0, 1, 2, 3, 4, 5, 6, 7\}$ to $\{1, 4, 7, 2, 5, 0, 3, 6\}$. Note that $a1 = 5$ is odd, $a_2 + a_4 + a_6 = 1 + 0 + 3 = 4$ is even, and $a_3 + a_5 = 1 + 1 = 2$ is even.

   Example 2 for $N = 2^3 = 8 : f(x) = 1 + 4x + x^2 + x^3 + x^5 + 3x^6$ is not PP over $\mathbb{Z}_{N=8}$ because it maps the sequence $\{0, 1, 2, 3, 4, 5, 6, 7\}$ to $\{1, 3, 5, 7, 1, 3, 5, 7\}$. Note that $a_1 = 4$ is even.

2. Case II ($N = p^n$): a theorem in [7.33] guarantees that $f(x)$ is a PP modulo pn if and only if $f(x)$ is a PP modulo p and $f^{'}(x) \neq 0$ modulo p, for every integer $x \in \mathbb{Z}_p^n$. Note that the Case I is simply a special case of the Case II because $p = 2$ is a prime number.

   Example 1 for $N = 3^n (p = 3) : f(x) = 1 + 2x + 3x^2$ is a PP over $\mathbb{Z}_3^n$ because $f(0) = 1$ modulo $3 = 1, f(1) = 6$ modulo $3 = 0$, $f(2) = 17$ modulo $3 = 2$, and $f^{'}(x) = 2 + 6x = 2$ modulo $3 = 2$ is a non-zero constant for all $x in \mathbb{Z}_3^n$.

   Example 2 for $N = 3^n (p = 3) : f(x) = 1 + 6x + 3x^2$ is not a PP $\mathbb{Z}_3^n$ because $f^{'}(x) = 6 + 6x = 0$ modulo $3 = 0$ for all $x in \mathbb{Z}_3^n$. For instance, for $N = 3^2 = 9$, $f(x)$ maps the sequence $\{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ to $\{1, 1, 7, 1, 1, 7, 1, 1, 7\}$.

3. Case III (arbitrary N): let $P = \{2, 3, 5, 7, \ldots\}$ be the set of prime numbers. Then, every $N \in \mathbb{Z}_+$ can be factored as $N = \prod_{p \in P} p^{n_{N,p}}$, where all p values are distinct prime numbers, $n_{N,p} \geq 1$ for a certain number of p and $n_{N,p} = 0$ otherwise. For example, if $N = 2500 = 2^2 \times 5^4$, then we have $n_{2500,2} = 2$ and $n_{2500,5} = 4$. A theorem in [7.41] states that for any $N = \prod_{p \in P} p^{n_{N,p}}$, $f(x)$ is a PP modulo N if and only if $f(x)$ is also a PP modulo $p^{n_{N,p}}$,$\forall p$ such that $n_{N,p} \geq 1$. With this theorem, verifying whether a polynomial is a PP modulo N reduces to verifying the polynomial modulo each $p^{n_{N,p}}$ factor of N. For $p = 2$, we use the theorem reported in Case I, which is a simple test on the polynomial coefficients. For $p \neq 2$, we must use the theorem reported in Case II, which cannot be done by simply testing the polynomial coefficients. For an arbitrary N, it is difficult to develop a simple coefficient test to check whether an arbitrary m-degree polynomial $f(x)$ is a PP modulo N. However, for quadratic polynomial ($m = 2$), $f(x) = a_0 + a_1 x + a_2 x^2$, a simple coefficient test have been proposed in [7.39]. Next section will address this coefficient test in details.

B. Quadratic Permutation Polynomials

Since the constant q_0 in the quadratic polynomial $q(x) = q_0 + q_1 x + q_2 x^2$ only causes a "cyclic shif" to the permuted values, we define in this section -without loosing generality- quadratic polynomials as $q(x) = q_1 x + q_2 x^2$. Let us first establish some abbreviations borrowed from [7.43], that we will use throughout this section. We express the fact that b is divisible by a, or a is divisor of b, by $a \,|b$. We also use $a not \,|b$ to express the contrary of $a \,|b$. The greatest common divisor of a and b is denoted by $gcd(a, b)$. Remember that $gcd(a, b) = 1$ indicates that a and b are relatively prime. As we will see in Proposition 1 below, we are mainly interested in the factorization of the coefficient q_2, which can be written according to the previous notation as $q_2 = \prod_{p \in P} p^{n_{q_2,p}}$. The following Proposition 1 provides a necessary and sufficient condition for verifying whether a quadratic polynomial is a PP modulo N.

Proposition 1: Let $N = \prod_{p \in P} p^{n_{N,p}}$. For a quadratic polynomial $q(x) = q_1 x + q_2 x^2$ modulo N to be a PP, the following necessary and sufficient conditions must be satisfied [7.39]

1) Either $2 not \,|N$ or $4 \,|N$ (i.e., $n_{N,2} \neq 1$) $gcd(q_1, N) = 1$ and $q_2 = \prod_{p \in P} p^{n_{q_2,p}}$, $n_{q_2,p} \geq 1$, $\forall p$ such that $n_{N,p} \geq 1$.

2) $2\,|N$ or $4not\,|N$ (i.e., $n_{N,2} = 1$), $q_1 + q_2$ is odd, $gcd(q_1, \frac{N}{2}) = 1$ and $q_2 = \prod_{p\in P} p^{n_{q_2,p}}$, $n_{q_2,p} \geq 1$, $\forall p$ such that $p \neq 2$ and $n_{N,p} \geq 1$.

The statement $q_2 = \prod_{p\in P} p^{n_{q_2,p}}$, $n_{q_2,p} \geq 1$, $\forall p$ such that $n_{N,p} \geq 1$ can be expressed in simple words as follows: *each p factor of N must also be a factor of* q_2. It is important to note that this statement still allows q_2 to have prime factors that differ from all p factors of N.

Example 1: if $N = 36$, then we have case 1) of Proposition 1 because $4\,|N$. All possible values of q_1 are simply the set of numbers that are relatively prime to N. Consequently, $q_1 = \{1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35\}$. Since $36 = 2^2 \times 3^2$ ($p_1 = 2$ and $p_2 = 3$), then 2 and 3 must be a factor of q_2. That is, $q_2 = \big\{(2 \times 3), (2^2 \times 3), (2^3 \times 3), (2 \times 3^2), 5 \times (2 \times 3)\big\} = \{6, 12, 24, 18, 30\}$. As mentioned above, the use of the prime number 5 in $5 \times (2 \times 3)$ does not violate the condition in case 1) of Proposition 1. In total, for $N = 36$ there are $12 \times 5 = 60$ possible quadratic PPs (QPP).

The statement $q_2 = \prod_{p\in P} p^{n_{q_2,p}}$, $n_{q_2,p} \geq 1$, $\forall p$ such that $p \neq 2$ and $n_{N,p} \geq 1$ of case 2) can also be expressed in simple words as follows: *each* $p6 = 2$ *factor of N must also be a factor of* q_2. It is important to note that this statement still allows q_2 to have the prime factor 2 and all prime factors that differ from all p factors of N(q_2 may or may not have 2 as a factor).

Example 2: if $N = 90$, then we have case 2) of Proposition 1 because $2\,|N$ and $4not\,|N$. Since $N = 90 = 2 \times 3^2 \times 5$, all p values that differ from 2 are $p_1 = 3$ and $p_2 = 5$. Therefore, the potential values for q_2 are

$$\big\{(3 \times 5), (3^2 \times 5), (3 \times 5^2), 2 \times (3 \times 5), 2^2 \times (3 \times 5)\big\} = \{15, 45, 75, 30, 60\}$$

Under the condition $gcd(q_1, N/2 = 45) = 1$, the potential values for q_1 are have 120 possible QPPs.

$$\{1, 2, 4, 7, 8, 11, 13, 14, 16, 17, 19, 22, 23, 26, 28, 29, 31, 32, 34, 37, 38, 41, 43, 44, 46,$$
$$47, 49, 52, 53, 56, 58, 59, 61, 62, 64, 67, 68, 71, 73, 74, 76, 77, 79, 82, 83, 86, 88, 89\}$$

Despite the tight conditions imposed by Proposition 1 on q_1 and q_2, the search space of QPPs is still large, especially for medium to large interleavers. Thus, it is desirable to reduce the search space further. A solution is to consider only QPPs that do have a quadratic inverse (for more details on quadratic inverse for QPP, see [7.39]). This solution for reducing the search space is based on the interesting finding reported by Rosnes and Takeshita [7.38], namely, for $32 \leq N \leq 512$ and $N = 1024$, the class of QPP-based interleavers with quadratic inverses are strictly superior (in term of minimum distance) to the class of QPP-based interleavers with no quadratic inverse. Using exhaustive

computer search, Rosnes and Takeshita provided, for turbo codes that use 8 and 16-state constituent codes, a very useful list for the best (in term of minimum distance) QPPs for a wide range of N ($32 \leq N \leq 512$ and $N = 1024$) [7.38]. After discussing a necessary and sufficient condition for verifying whether a quadratic polynomial is a PP modulo N, and providing some examples, let us discuss some properties of QPPs. It is well known that a linear polynomial, $l(x) = l_0 + l_1 x$(or simply $l(x) = l_1 x$), is guaranteed to be a PP modulo N if l_1 is chosen relatively prime to N (i.e., $gcd(l_1, N) = 1$). Consequently, linear permutation polynomials (LPP) always exist for any N, but unfortunately this is not true for QPPs. For example, there are no QPP for $N = 11$ and for $2 \leq N \leq 4096$ there are only 1190 values of N that have QPPs (roughly 29%) [7.44]. A theorem in [7.44] guarantees the existence of QPP for all $N = 8i$, $i \in \mathbb{Z}_+$(i.e., multiples of a typical computer byte size of 8). It is shown in [7.44] that some QPP degenerate to LPP (i.e., there exists an LPP that generates the same permutation over the ring $\mathbb{Z}_N$). A QPP is called reducible if it degenerates to an LPP; otherwise it is called irreducible. For instance, example 1 in Case I of sub-section A could be simply reduced to $f(x) = l + 3x$ modulo 8 to obtain the same permutation. In [7.38], it is shown that some reducible QPPs can achieve better minimum distances than irreducible QPP for some short to medium interleavers. However, for large intereavers, the class of irreducible QPPs are better (in term of minimum distance) than the class of LPP; and if not, that particular length will not have any good minimum distance [7.38].

7.4 Decoding turbo codes

7.4.1 Turbo decoding

Decoding a binary turbo code is based on the schematic diagram of Figure 7.12. The loop allows each decoder to take advantage of all the information available. The values considered at each node of the layout are LLRs, the decoding operations being performed in the logarithmic domain.

The LLR at the output of a decoder of systematic codes can be seen as the sum of two terms: the intrinsic information, coming from the transmission channel, and the extrinsic information, which this decoder adds to the former to perform its correction operation. As the intrinsic information is used by the two decoders (at different instants), it is the extrinsic information produced by each of the decoders that must be transmitted to the other as new information, to ensure joint convergence. Section 7.4.2 details the operations performed to calculate the extrinsic information, by implementing the MAP algorithm or its simplified Max-Log-MAP version.

Because of latency effects, the exchange of extrinsic information, in a digital processing circuit, must be implemented via an iterative process: first decoding

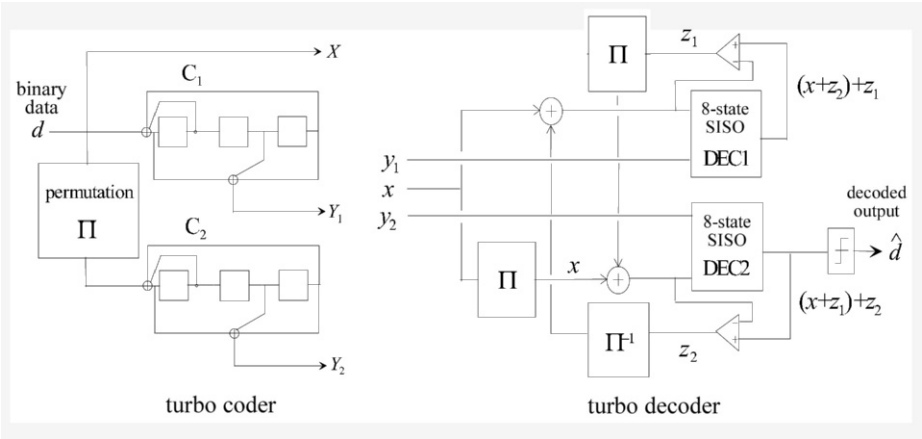

Figure 7.12 – 8-state turbo encoder and schematic structure of the corresponding turbo decoder. The two elementary decoders exchange probabilistic information, called extrinsic information (z)

.

by DEC_1 and putting extrinsic information z_1 in the memory, second decoding by DEC_2 and putting extrinsic information z_2 in the memory (end of the first iteration), again using DEC_1 and putting z_1 in the memory, etc. Different hardware architectures, with more or less great degrees of parallelism, can be envisaged to accelerate the iterative decoding.

If we wanted to decode the turbo code using a single decoder, which would take into account all the possible states of the encoder, for each element of the message decoded, we would obtain one and only one probability of having a binary value equal to 0 or to 1. As for the composite structure of Figure 7.12, it uses two decoders working jointly. By analogy with the result that the single decoder would provide, they therefore need to *converge towards the same decisions, with the same probabilities*, for each of the data considered. That is the fundamental principle of "turbo" processing, which justifies the structure of the decoder, as the following reasoning shows.

The role of a SISO decoder (see Section 7.4.2), is to process the LLRs at its input to try to make them more reliable, thanks to local redundancy (that is, y_1 for DEC1, y_2 for DEC2). The LLR produced by a decoder of binary codes, relative to data d, can be written simply as

$$\mathrm{LLR}_{\mathrm{output}}(d) = \mathrm{LLR}_{\mathrm{input}(d)+z(d)} \tag{7.21}$$

where $z(d)$ is the extrinsic information specific to d. The LLR is improved when z is negative and d is a 0, or when z is positive and d is a 1.

After p iterations, the output of DEC1 is:

$$\mathrm{LLR}^{p}_{\mathrm{output},1}(d) = (x + z_2^{p-1}(d)) + z_1^{p}(d)$$

and the output of DEC2 is

$$\mathrm{LLR}^{p}_{\mathrm{output},2}(d) = (x + z_1^{p-1}(d)) + z_2^{p}(d)$$

If the iterative process converges towards a stable solution, $z_1^p(d) - z_1^{p-1}(d)$ and $z_2^p(d) - z_2^{p-1}(d)$ tend towards zero when p tends towards infinity. Consequently, the two LLRs relative to d become identical, thus satisfying the basic criterion of common probability mentioned above. As for proof of the convergence, it is still being studied further and on this topic we can, for example, consult [7.49, 7.24].

Apart from the permutation and inverse permutation functions, Figure 7.13 details the operations performed during turbo decoding:

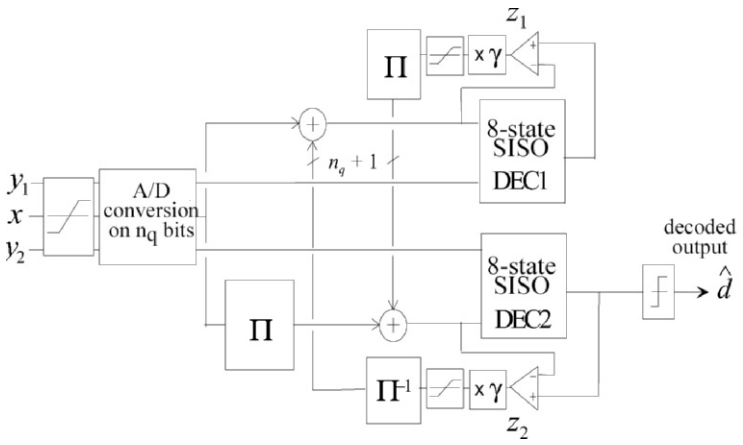

Figure 7.13 – Operations shown (clipping, quantization, attenuation of the extrinsic information) in the turbo decoder of Figure 7.12.

1. Analogue to digital (A/D) conversion transforms the data coming from the demodulator into samples exploitable by the digital decoder. Two parameters are involved in this operation: n_q, the number of quantization bits, and Q, the scale factor, that is, the ratio between the average absolute value of the quantized signal and its maximum absolute value. n_q is fixed to a compromise value between the precision required and the complexity of the decoder. With $n_q = 4$, the performance of the decoder is very close to what we obtain with real samples. The value of Q depends on the modulation, on the coding rate and on the type of channel. It is, for example, larger for a Rayleigh channel than for a Gaussian channel.

2. The role of SISO decoding is to increase the equivalent signal to noise ratio of the LLR, that is, to provide more reliable extrinsic information at output z_{output} than at input (z_{input}). The convergence of the iterative process (see Section 7.6) will depend on the transfer function $\text{SNR}(z_{output}) = G(\text{SNR}(z_{input}))$ of each of the decoders.

   When data is not available at the input of the SISO decoder, due to puncturing for example, a neutral value (analogue zero) is substituted for this missing data.

3. When the elementary decoding algorithm is not the optimal MAP algorithm but a sub-optimal simplified version, the extrinsic information has to undergo some transformations before being used by a decoder:

   - multiplying the extrinsic information by factor γ, lower than 1, guarantees the stability of the looped structure. γ can vary over the iterations, for example, from 0.7 at the beginning of the iterative process, to 1 for the last iteration.
   - clipping the extrinsic information solves both the issue of limiting the size of the memories and that of participating in the stability of the process. A typical value of the maximum dynamics of the extrinsic information is twice the input dynamics of the decoder.

4. Binary decision taking is performed by thresholding at value 0.

   The number of iterations required by turbo decoding depends on the size of the block and on the coding rate. Generally, the larger the decoded block and the slower the convergence, the higher the MHD of the code. The same occurs when the coding rates are low. In practice, we limit the number of iterations to a value between 4 and 10, according to the speed, latency and consumption constraints imposed by the application.

Figure 7.14 gives an example of the performance of a binary turbo code, taken from the UMTS standard [7.3]. We can observe a decrease in packet error rates (PER), just close to the theoretical limit (that is, around 0,5 dB, taking into account the size of the block), but also a fairly pronounced change in slope, due to an MHD that is not very high ($d_{\min} = 26$) for a rate of 1/3.

7.4.2 SISO decoding and extrinsic information

Here we present processing performed in practice in a SISO decoder using the MAP algorithm [7.6] or its simplified version, the Max-log-MAP algorithm, also called the SubMAP algorithm [7.37], to decode RSC m-binary codes and implement iterative decoding. For binary codes and turbo codes, all these equations can be simplified by taking $m = 1$.

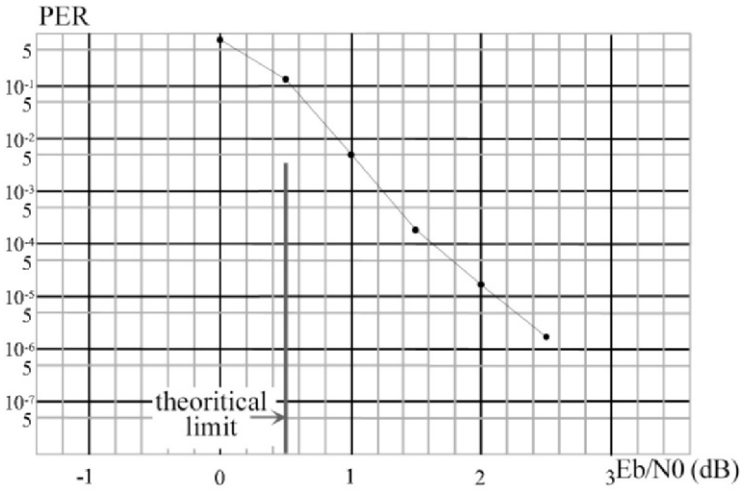

Figure 7.14 – Performance in packet error rates (PER) of the UMTS standard turbo code for $k = 640$ and $R = 1/3$ on a Gaussian channel with 4-PSK modulation. Decoding using the Max-Log-MAP algorithm with 6 iterations.

Notations

A sequence of data $\mathbf{d}$ is defined by $\mathbf{d} \equiv \mathbf{d}_0^{k-1} = (\mathbf{d}_0 \cdots \mathbf{d}_i \cdots \mathbf{d}_{k-1})$, where $\mathbf{d}_i$ is the vector of m-binary data applied at the input of the encoder at instant i: $\mathbf{d}_i = (d_{i,1} \cdots d_{i,l} \cdots d_{i,m})$. The value of $\mathbf{d}_i$ can also be represented by the scalar integer value $j = \sum_{l=1}^{m} 2^{l-1} d_{i,l}$, ranging between 0 and $2^m - 1$ and we can then write $\mathbf{d}_i \equiv j$.

In the case of two or four-phase PSK modulation (2-PSK, 4-PSK), the encoded modulated sequence $\mathbf{u} \equiv \mathbf{u}_0^{k-1} = (\mathbf{u}_0 \cdots \mathbf{u}_i \cdots \mathbf{u}_{k-1})$ is made up of vectors $\mathbf{u}_i$ of size $m + m'$: $\mathbf{u}_i = (u_{i,1} \cdots u_{i,l} \cdots u_{i,m+m'})$, where $u_{i,l} = \pm 1$ for $l = 1 \cdots m + m'$ and m' is the number of redundancy bits added to the m bits of information. The symbol $u_{i,l}$ is therefore representative of a systematic bit for $l \leq m$ and of a redundancy bit for $l > m$.

The sequence observed at the output of the demodulator is denoted $\mathbf{v} \equiv \mathbf{v}_0^{k-1} = (\mathbf{v}_0 \cdots \mathbf{v}_i \cdots \mathbf{v}_{k-1})$, with $\mathbf{v}_i = (v_{i,1} \cdots v_{i,l} \cdots v_{i,m+m'})$. The series of the states of the encoder between instants 0 and k is denoted $\mathbf{S} = \mathbf{S}_0^k = (\mathbf{S}_0 \cdots \mathbf{S}_i \cdots \mathbf{S}_k)$. The following is based on the results presented in the chapter on convolutional codes.

Decoding following the Maximum *A Posteriori* (MAP) criterion

At each instant i, the weighted (probabilistic) estimates provided by the MAP decoder are the 2^m *a posteriori* probabilities (APP) $\Pr(\mathbf{d}_i \equiv j \mid \mathbf{v})$, $j = 0 \cdots 2^m - 1$. The corresponding hard decision, $\hat{\mathbf{d}}_i$, is the binary representation of value j that maximizes the APP.

Each APP can be expressed as a function of the joint likelihoods $p(\mathbf{d}_i \equiv j, \mathbf{v})$:

$$\Pr(\mathbf{d}_i \equiv j \mid \mathbf{v}) = \frac{p(\mathbf{d}_i \equiv j, \mathbf{v})}{p(\mathbf{v})} = \frac{p(\mathbf{d}_i \equiv j, \mathbf{v})}{\sum_{l=0}^{2^m-1} p(\mathbf{d}_i \equiv l, \mathbf{v})} \tag{7.22}$$

In practice, we calculate the joint likelihoods $p(\mathbf{d}_i \equiv j, \mathbf{v})$ for $j = 0 \cdots 2^m - 1$ then each APP is obtained by normalization.

The trellis representative of a code with memory ν has 2^ν states, taking their scalar value s in $(0, 2^\nu - 1)$. The joint likelihoods are calculated from the recurrent *forward* $\alpha_i(s)$ and *backward* probabilities $\beta_i(s)$ and the branch likelihoods $g_i(s', s)$:

$$p(\mathbf{d}_i \equiv j, \mathbf{v}) = \sum_{(s',s)/\mathbf{d}_i(s',s) \equiv j} \beta_{i+1}(s)\, \alpha_i(s')\, g_i(s', s) \tag{7.23}$$

where $(s', s)/\mathbf{d}_i(s', s) \equiv j$ denotes the set of transitions from state to state $s' \to s$ associated with the m-binary j. This set is, of course, always the same in a trellis that is invariant over time.

The value $g_i(s', s)$ is expressed as:

$$g_i(s', s) = \Pr{}^a(\mathbf{d}_i \equiv j, \mathbf{d}_i(s', s) \equiv j).p(\mathbf{v}_i \mid \mathbf{u}_i) \tag{7.24}$$

where $\mathbf{u}_i$ is the set of systematic and redundant information symbols associated with transition $s' \to s$ of the trellis at instant i and $\Pr^a(\mathbf{d}_i \equiv j, \mathbf{d}_i(s', s) \equiv j)$ is the *a priori* probability of transmitting the m-tuple of information and that this would correspond to transition $s' \to s$ at instant i. If transition $s' \to s$ does not exist in the trellis for $\mathbf{d}_i \equiv j$, then $\Pr^a(\mathbf{d}_i \equiv j, \mathbf{d}_i(s', s) \equiv j) = 0$, otherwise the transition is given by the source statistics (usually uniform, in practice).

In the case of a Gaussian channel with binary inputs, value $p(\mathbf{v}_i \mid \mathbf{u}_i)$ can be written:

$$p(\mathbf{v}_i \mid \mathbf{u}_i) = \prod_{l=1}^{m+m'} \left(\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(v_{i,l} - u_{i,l})^2}{2\sigma^2} \right) \right) \tag{7.25}$$

where σ^2 is the variance of the additive white Gaussian noise. In practice, we keep only the terms that are specific to the transition considered and that are

not eliminated by division in the expression (7.22):

$$p'(\mathbf{v}_i \mid \mathbf{u}_i) = \exp\left(\frac{\sum_{l=1}^{m+m'} v_{i,l} \cdot u_{i,l}}{\sigma^2}\right) \tag{7.26}$$

The forward and backward recurrent probabilities are calculated as follows:

$$\alpha_i(s) = \sum_{s'=0}^{2^\nu - 1} \alpha_{i-1}(s')\, g_{i-1}(s', s) \quad \text{for } i = 1 \cdots k \tag{7.27}$$

and:

$$\beta_i(s) = \sum_{s'=0}^{2^\nu - 1} \beta_{i+1}(s')\, g_i(s, s') \quad \text{for } i = k-1 \cdots 0 \tag{7.28}$$

To avoid problems of precision or of overflow in the representation of these values, they have to be normalized regularly. The initialization of the recursions depends on the knowledge or not of the state of the encoder at the beginning and at the end of encoding. If the initial state $\mathbf{S}_0$ of the encoder is known, then $\alpha_0(\mathbf{S}_0) = 1$ and $\alpha_0(s) = 0$ for any other state, otherwise all the $\alpha_0(s)$ are initialized to the same value. The same rule is applied for the final state $\mathbf{S}_k$. For circular codes, initialization is performed automatically after the prologue step, which starts from identical values for all the states of the trellis.

In the context of iterative decoding, the composite decoder uses two elementary decoders exchanging *extrinsic probabilities*. Consequently, the basic decoding brick described above must be reconsidered in order to:

1. take into account an extrinsic probability, $\Pr^{ex}(\mathbf{d}_i \equiv j \mid \mathbf{v}')$, in expression (7.24), calculated by the other elementary decoder of the composite decoder, from its own input sequence $\mathbf{v}'$,
2. produce its own extrinsic probability $\Pr^{ex}(\mathbf{d}_i \equiv j \mid \mathbf{v}')$ that will be used by the other elementary decoder.

In practice, for each value of j, $j = 0 \cdots 2^m - 1$:

1. in expression (7.24), the *a priori* probability $\Pr^a(\mathbf{d}_i \equiv j, \mathbf{d}_i(s', s) \equiv j)$ is replaced by the modified *a priori* probability $\Pr^{@}(\mathbf{d}_i \equiv j, \mathbf{d}_i(s', s) \equiv j)$, having for its expression, to within one normalization factor:

$$\Pr^{@}(\mathbf{d}_i \equiv j, \mathbf{d}_i(s', s) \equiv j) = \Pr^a(\mathbf{d}_i \equiv j, \mathbf{d}_i(s', s) \equiv j).\Pr^{ex}(\mathbf{d}_i \equiv j \mid \mathbf{v}') \tag{7.29}$$

1. $\Pr^{ex}(\mathbf{d}_i \equiv j \mid \mathbf{v})$ is given by:

$$\Pr^{ex}(\mathbf{d}_i \equiv j \mid \mathbf{v}) = \frac{\sum_{(s',s)/\mathbf{d}_i(s',s)\equiv j} \beta_{i+1}(s)\,\alpha_i(s')\,g_i^*(s',s)}{\sum_{(s',s)} \beta_{i+1}(s)\,\alpha_i(s')\,g_i^*(s',s)} \tag{7.30}$$

The terms $g_i^*(s',s)$ are non-zero if $s' \rightarrow s$ corresponds to a transition of the trellis and are then inferred from the expression of $p(\mathbf{v}_i \mid \mathbf{u}_i)$ by eliminating the systematic part of the information. In the case of a transmission over a Gaussian channel with binary inputs and starting from the simplified expression (7.26) of $p'(\mathbf{v}_i \mid \mathbf{u}_i)$, we have:

$$g_i^*(s',s) = \exp\left(\frac{\sum_{l=m+1}^{m+m'} v_{i,l}u_{i,l}}{\sigma^2}\right) \tag{7.31}$$

The simplified Max-Log-MAP or SubMAP algorithm

Decoding following the MAP criterion requires a large number of operations, including calculating exponentials and multiplications. Re-writing the decoding algorithm in the logarithmic domain simplifies the processing. The weighted estimations provided by the decoder are then values proportional to the logarithms of the APPs, called Log-APP logarithms, denoted L:

$$L_i(j) = -\frac{\sigma^2}{2} \ln \Pr(\mathbf{d}_i \equiv j \mid \mathbf{v}) \;, \quad j = 0 \cdots 2^m - 1 \tag{7.32}$$

We define $M_i^{\alpha}(s)$ and $M_i^{\beta}(s)$ the forward and backward metrics relative to node s at instant i, and $M_i(s',s)$, the branch metric relative to the $s' \rightarrow s$ transition of the trellis at instant i by:

$$\begin{aligned} M_i^{\alpha}(s) &= -\sigma^2 \ln \alpha_i(s) \\ M_i^{\beta}(s) &= -\sigma^2 \ln \beta_i(s) \\ M_i(s',s) &= -\sigma^2 \ln g_i(s',s) \end{aligned} \tag{7.33}$$

Introduce values $A_i(j)$ and B_i calculated as:

$$A_i(j) = -\sigma^2 \ln \left[\sum_{(s',s)/\mathbf{d}_i(s',s)\equiv j} \beta_{i+1}(s)\alpha_i(s')g_i(s',s)\right] \tag{7.34}$$

$$B_i = -\sigma^2 \ln \left[\sum_{(s',s)} \beta_{i+1}(s)\alpha_i(s')g_i(s',s)\right] \tag{7.35}$$

$L_i(j)$ can then be written, by reference to (7.22) and (7.23), as follows:

$$L_i(j) = \frac{1}{2}\left(A_i(j) - B_i\right) \tag{7.36}$$

Expressions (7.34) and (7.35) can be simplified by applying the so-called Max-Log approximation:

$$\ln(\exp(a) + \exp(b)) \approx \max(a, b) \tag{7.37}$$

For $A_i(j)$ we get:

$$A_i(j) \approx \min_{(s',s)/\mathbf{d}_i(s',s)\equiv j} \left(M_{i+1}^{\beta}(s) + M_i^{\alpha}(s') + M_i(s',s)\right) \tag{7.38}$$

and for B_i:

$$B_i \approx \min_{(s',s)} \left(M_{i+1}^{\beta}(s) + M_i^{\alpha}(s') + M_i(s',s)\right) = \min_{l=0\cdots 2^m-1} A_i(l) \tag{7.39}$$

and finally we get:

$$L_i(j) = \frac{1}{2}\left(A_i(j) - \min_{l=0\cdots 2^m-1} A_i(l)\right) \tag{7.40}$$

Note that these values are always positive or equal to zero.

Introduce the values L^a proportional to the logarithms of the *a priori* probabilities $\Pr^a$:

$$L_i^a(j) = -\frac{\sigma^2}{2} \ln \Pr^a(\mathbf{d}_i \equiv j) \tag{7.41}$$

Branch metrics $M_i(s', s)$ can be written, according to (7.24) and (7.33):

$$M_i(s', s) = 2L_i^a(\mathbf{d}(s', s)) - \sigma^2 \ln p(\mathbf{v}_i \mid \mathbf{u}_i) \tag{7.42}$$

If the statistic of the *a priori* transmission of the m-tuples $\mathbf{d}_i$ is uniform, term $2L_i^a(\mathbf{d}(s', s))$ can be omitted from the above relation since it is the same value that is used in all the branch metrics. In the case of a transmission over a Gaussian channel with binary inputs, we have according to (7.26):

$$M_i(s', s) = 2L_i^a(\mathbf{d}(s', s)) - \sum_{l=1}^{m+m'} v_{i,l} \cdot u_{i,l} \tag{7.43}$$

The forward and backward metrics are then calculated from the following recurrence relations:

$$M_i^{\alpha}(s) = \min_{s'=0\cdots 2^{\nu}-1} \left(M_{i-1}^{\alpha}(s') - \sum_{l=1}^{m+m'} v_{i-1,l} \cdot u_{i-1,l} + 2L_{i-1}^a(\mathbf{d}(s', s))\right) \tag{7.44}$$

$$M_i^{\beta}(s) = \min_{s'=0\cdots 2^{\nu}-1} \left(M_{i+1}^{\beta}(s') - \sum_{l=1}^{m+m'} v_{i,l} \cdot u_{i,l} + 2L_i^a(\mathbf{d}(s,s') \right) \tag{7.45}$$

Applying the Max-Log-MAP logarithm in fact amounts to performing two Viterbi decodings, in the forward and backward directions. That is the reason why it is also called the *dual Viterbi* algorithm.

If the initial state of the encoder, $\mathbf{S}_0$, is known, then $M_0^{\alpha}(\mathbf{S}_0) = 0$ and $M_0^{\alpha}(s) = +\infty$ for any other state, otherwise all the $M_0^{\alpha}(s)$ are initialized to the same value. The same rule is applied for the final state. For circular codes, all the metrics are initialized to the same value at the beginning of the prologue.

Finally, taking into account (7.38) and replacing $M_i(s',s)$ by its expression (7.43), we obtain:

$$A_i(j) = \min_{(s',s)/\mathbf{d}_i(s',s)\equiv j} \left(M_{i+1}^{\beta}(s) + M_i^{\alpha}(s') - \sum_{l=1}^{m+m'} v_{i,l} \cdot u_{i,l} \right) + 2L_i^a(j) \tag{7.46}$$

The hard decision taken by the decoder is the value of j, $j = 0\cdots 2^m - 1$, which minimizes $A_i(j)$. Let us denote this value j_0. According to (7.40), $L_i(j)$ can be written:

$$L_i(j) = \frac{1}{2}\left[A_i(j) - A_i(j_0)\right] \text{ pour } j = 0\cdots 2^m - 1 \tag{7.47}$$

We note that the presence of coefficient σ^2 in definition (7.32) of $L_i(j)$ allows us to ignore the knowledge of this parameter for computing the metrics and hence for all the decoding. This is an important advantage of the Max-Log-MAP method over the MAP method.

In the context of iterative decoding, term $L_i^a(j)$ is modified in order to take into account extrinsic information $L_i^*(j)$ coming from the other elementary decoder:

$$L_i^{@}(j) = L_i^a(j) + L_i^*(j) \tag{7.48}$$

On the other hand, the extrinsic information produced by the decoder is obtained by eliminating in $L_i(j)$ the terms containing the direct information about $\mathbf{d}_i$, that is, the intrinsic and *a priori* information:

$$\begin{aligned} L_i^*(j) = \tfrac{1}{2}\Bigg[& \min_{(s',s)/\mathbf{d}_i(s',s)\equiv j} \left(M_{i+1}^{\beta}(s) + M_i^{\alpha}(s') - \sum_{l=m+1}^{m+m'} v_{i,l} \cdot u_{i,l} \right) \\ & - \min_{(s',s)/\mathbf{d}_i(s',s)\equiv j_0} \left(M_{i+1}^{\beta}(s) + M_i^{\alpha}(s') - \sum_{l=m+1}^{m+m'} v_{i,l} \cdot u_{i,l} \right) \Bigg] \end{aligned} \tag{7.49}$$

The expression of $L_i(j)$ can then be formulated as follows:

$$L_i(j) = L_i^*(j) + \frac{1}{2}\sum_{l=1}^{m} v_{i,l} \cdot \left[u_{i,l}\big|_{\mathbf{d}_i\equiv j} - u_{i,l}\big|_{\mathbf{d}_i\equiv j_0} \right] + \left[L_i^{@}(j) - L_i^{@}(j_0) \right] \tag{7.50}$$

This expression shows that extrinsic information $L_i^*(j)$ can, in practice, be deduced from $L_i(j)$ by simple subtraction. Factor $\frac{1}{2}$ in definition (7.32) of $L_i(j)$ allows us to obtain a weighted decision and extrinsic information $L_i^*(j)$ on the same scale as the noisy samples $v_{i,l}$.

7.4.3 Practical considerations

The simplest way to perform turbo decoding is totally sequential and uses the following operations, here founded on the Max-Log-MAP algorithm and repeated as many times as necessary:

1. Backward recursion for code C_2 (Figure 7.12), calculation and memorization of metrics $M_i^\beta(s)$, $i = k-1, ..., 0$ and $s = 0, ..., 2^\nu - 1$,
2. Forward recursion for code C_2, calculation of metrics $M_i^\alpha(s)$, $i = 0, ...,$ $k-1$ and $s = 0, ..., 2^\nu - 1$. Calculation and memorization of the extrinsic information,
3. Backward recursion for code C_1, calculation and memorization of metrics $M_i^\beta(s)$, $i = k-1, ..., 0$ and $s = 0, ..., 2^\nu - 1$,
4. Forward recursion for code C_1, calculation of metrics $M_i^\alpha(s)$, $i = 0, ...,$ $k-1$ and $s = 0, ..., 2^\nu - 1$. Calculation and memorization of the extrinsic information. Binary decisions (at the last iteration).

The first practical problem lies in the memory necessary to store metrics $M_i^\beta(s)$. Processing the coded messages of $k = 1000$ bits, for example, with 8-state decoders and quantization of the metrics on 6 bits, at first sight requires a storage capacity of 48000 bits for each decoder. In sequential operation (alternate processing of C_1 and C_2), this memory can, of course, be used by the two decoders in turn. The technique used to greatly reduce this memory is that of the *sliding window*. It involves (Figure 7.15) replacing all the backward processing, from $i = k-1$ to 0, by a succession of partial forward processings, from $i = i_F$ to 0, then from $i = 2i_F$ to i_F, from $i = 3i_F$ to $2i_F$ etc., where i_F is an interval of some tens of trellis sections. Each partial backward processing includes a "prologue" (dotted line), that is, a step without memorization whose aim is to estimate as correctly as possible the accumulated backward metrics in positions i_F, $2i_F$, $3i_F$, etc. The parts shown by a solid line correspond to the phases during which these metrics are memorized. The same memory can be used for all the partial backward recursions. The forward recursion is performed without any discontinuity.

The process greatly reduces the storage capacity necessary which, in addition, becomes independent of the length of the messages. The drawback lies in the necessity to perform the additional operations – the prologues – that can increase the total calculation complexity by 10 to 20 %. However, these prologues can be

avoided after the first iteration if the estimates of the metrics at the boundary indices are put into memory to be used as departure points for the calculations of the following iteration.

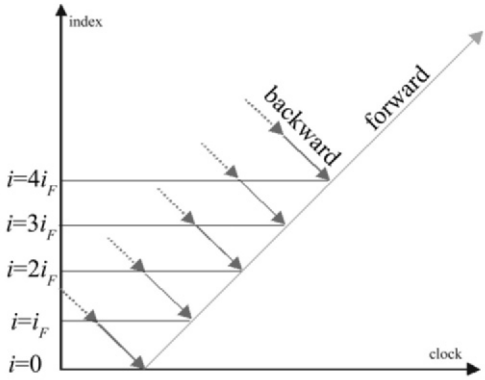

Figure 7.15 – Operation of the forward and backward recursions when implementing the MAP algorithm with a sliding window.

The second practical problem is that of the speed and latency of decoding. The extent of the problem depends of course on the application and on the ratio between the decoding circuit clock and the data rate. If the latter is very high, the operations can be performed by a single machine, in the sequential order presented above. In specialized processors of the DSP (*digital signal processor*) type, cabled co-processors may be available to accelerate the decoding. In dedicated circuits of the ASIC (*application-specific integrated circuit*) type, acceleration of the decoding is obtained by using parallelism, that is, multiplying the number of arithmetical operators, if possible without increasing the capacity of the memories required to the same extent. Then, problems of access to these memories are generally posed.

Note first that only knowledge of permutation $i = \Pi(j)$ is necessary for implementation of the iterative decoding and not that of inverse permutation Π^{-1}, as could be wrongly assumed from the schematic diagrams of Figures 7.12 and 7.13. Consider, for example, two SISO decoders working in parallel to decode the two elementary codes of the turbo code and based on two dual-port memories for the extrinsic information (Figure 7.16). The DEC1 decoder associated with the first code produces and receives the extrinsic information in the natural order i. The DEC2 decoder associated with the second code works according to index j but writes and recovers its data at addresses $i = \Pi(j)$. Knowledge of Π^{-1}, which could pose a problem depending on the permutation model selected, is therefore not required.

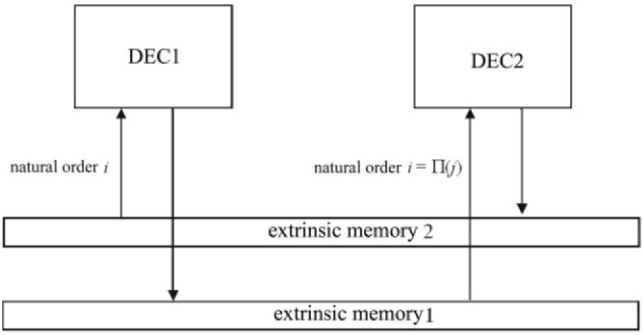

Figure 7.16 – Implementing turbo decoding does not require explicit knowledge of Π^{-1}

In addition, the two extrinsic information memories can be merged into a single one, observing that extrinsic information that has just been read and exploited by a decoder no longer has to be retained. It can thus be replaced immediately afterwards by another datum, which can be the extrinsic information output from the same decoder. Figure 7.17 illustrates this process, which imposes a slight hypothesis: working indices i and j have the same parity and permutation $i = \Pi(j)$ inverses the parity. For example, with the permutation defined by (7.4), this hypothesis is satisfied if the departure index i_0 is odd and the length of the message k is even.

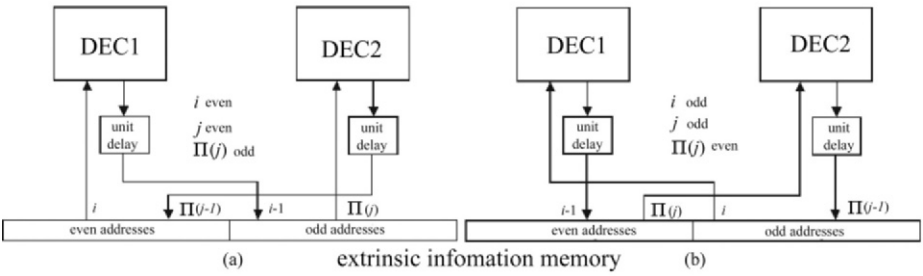

Figure 7.17 – In practice, the storage of the extrinsic information uses only a single memory.

The extrinsic information memory is divided in two pages corresponding to the sub-sets of the even and odd addresses. Access to these two pages, in a dual-port memory, is alternated regularly. In Figure 7.17(a), in the even cycles, DEC1 reads at an even address and writes the extrinsic information produced during the previous cycle, via a buffer memory with unit delay, to an odd address. Meanwhile, DEC2 reads at an even address and writes to an odd address. In

Figure 7.17(b), during the odd cycles, the accesses to the reading-writing pages are exchanged.

To further increase the degree of parallelism in the iterative decoder, the forward and backward recursion operations can also be tackled inside each of the two decoders (DEC1 and DEC2). This can be easily implemented by considering the diagram of Figure 7.15.

Finally, depending on the permutation model used, the number of elementary decoders can be increased beyond two. Consider for example the circular permutation defined by (7.14) and (7.16), with cycle $C = 4$ and k a multiple of 4.

The congruences of j and $\Pi(j)$ modulo 4, are periodic. Parallelism with degree 4 is then possible following the principle described in Figure 7.18 [7.17]. For each forward or backward recursion (these also can be done in parallel), four processors are used. At the same instant, these processors process data whose addresses have different congruences modulo 4. In the example in the figure, the forward recursion is considered and we assume that $k/4$ is also a multiple of 4. Then, we have first processor begin at address 0, the second at address $k/4 + 1$, the third at address $k/2 + 2$ and finally the fourth at address $3k/4 + 3$. At each instant, as the processors advance by one place each time, the congruences modulo 4 of the addresses are always different. Addressing conflicts are avoided via a router that directs the four processors towards four memory pages corresponding to the four possible congruences. If $k/4$ is not a multiple of 4, the departure addresses are no longer exactly 0, $k/4+1$, $k/2+2$, $3k/4+3$ but the process is still applicable.

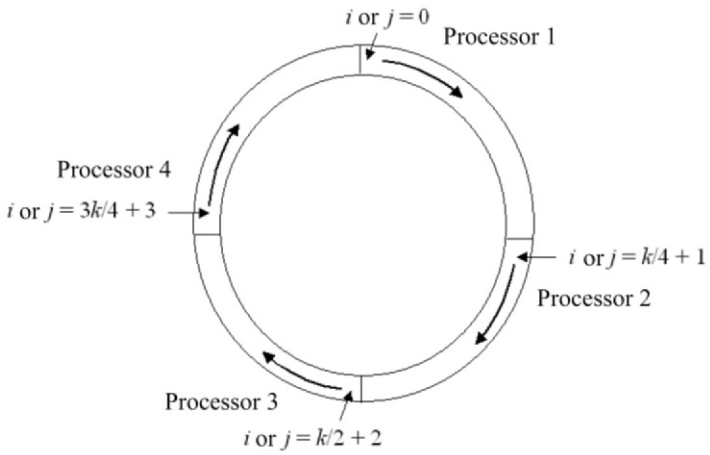

Figure 7.18 – The forward recursion circle is divided into 4 quadrants.

Whatever the value of cycle C, higher degrees of parallelism of value pC, can be implemented. Indeed, any multiple of C, the basic cycle in the permutation, is also a cycle in the permutation, on condition that pC is a divisor of k. That is, j modulo pC and $\Pi(j)$ modulo pC are periodic on the circle of length k, which can then be cut into pC fractions of equal length. For example, a degree 64 parallelism is possible for a value of k equal to 2048.

However, whatever the degree of parallelism, a minimum latency is unavoidable: the time required for receiving a packet and putting it into the buffer memory. While this packet is being put into memory, the decoder works on the information contained in the previous packet. If this decoding is performed in a time at least equal to the memorization time, then the total decoding latency is at maximum twice this memorization time. The level of parallelism in the decoder is adjusted according to this objective, which may be a constraint in certain cases.

For further information about the implementation of turbo decoders, of all the publications on this topic, [7.47] is a good resource.

7.5 m-binary turbo codes

$m-$binary turbo codes are built from recursive systematic convolutional (RSC) codes with m binary inputs ($m \geq 2$). There are at least two ways to build an $m-$binary convolutional code: either from the Galois field $\mathbf{F}_{2^m}$, or from the Cartesian product $(\mathbf{F}_2)^m$. Here, we shall only deal with the latter, which is more convenient. Indeed, a code elaborated in $\mathbf{F}_{2^m}$, with a memory ν, has $2^{\nu m}$ possible states, whereas the number of states of the code defined in $(\mathbf{F}_2)^m$, with the same memory, can be limited to 2^ν.

The advantages of the m-binary construction compared to the classical ($m = 1$) turbo code scheme, are varied: better convergence of the iterative process, larger minimum distances, less puncturing, lower latency, robustness towards the sub-optimality of the decoding algorithm, in particular when the MAP algorithm is simplified into its Max-Log-MAP version [7.23].

The case $m = 2$ has already been adopted in the European standards for the return path in digital video broadcasting via the satellite network and in the terrestrial network [7.2, 7.1] as well as in the IEEE 802.16 standard [7.5]. Combined with the circular trellis technique, these 8-state turbo codes, called double-binary turbo codes, offer good average performance and great flexibility in adapting to different block sizes and different rates, whilst retaining reasonable decoding complexity.

7.5.1 $m-$binary RSC encoders

Figure 7.19 presents the general structure of an $m-$binary RSC encoder. It uses a pseudo-random generator with code memory ν and generator matrix $\mathbf{G}$ (size

$\nu \times \nu$). The input vector $\mathbf{d}$ with m components is connected to the different possible nodes thanks to a grid of interconnections whose binary matrix, size $\nu \times m$, is denoted $\mathbf{C}$. The vector $\mathbf{T}$ applied to the ν possible taps of the register at instant i, is given by:

$$\mathbf{T}_i = \mathbf{C}.\mathbf{d}_i \tag{7.51}$$

with $\mathbf{d}_i = (d_{1,i} \dots d_{m,i})$.

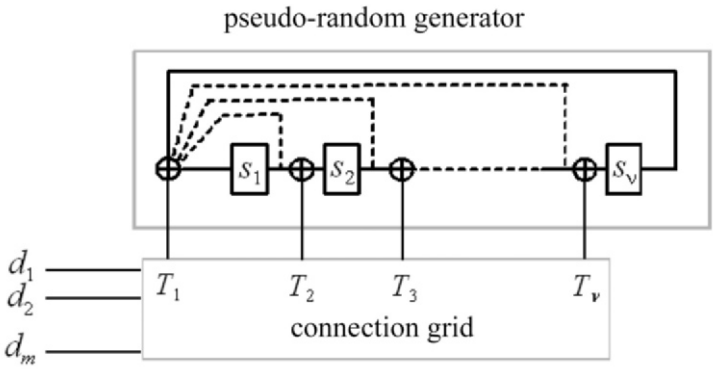

Figure 7.19 – General structure of an m-binary RSC encoder with code memory ν. The time index is not shown.

If we wish to avoid parallel transitions in the trellis of the code, condition $m \leq \nu$ must be respected and matrix $\mathbf{C}$ must be full rank. Except for very particular cases, this encoder is not equivalent to an encoder with a single input on which we would successively present $d_1, d_2, \cdots, d_m$. An $m-$binary encoder is therefore not decomposable generally.

The redundant output of the machine (not shown in the figure) is calculated at instant i according to the expression:

$$y_i = \sum_{j=1\dots m} d_{j,i} + \mathbf{R}^{\mathbf{T}}\mathbf{S}_i \tag{7.52}$$

where $\mathbf{S}_i$ is the state vector at instant i and $\mathbf{R}^{\mathbf{T}}$ is the transposed redundancy vector. The p-th component of $\mathbf{R}$ equals 1 if the p-th component of $\mathbf{S}_i$ is used in the construction of y_i and 0 otherwise. We can show that y_i can also be written as:

$$y_i = \sum_{j=1\dots m} d_{j,i} + \mathbf{R}^{\mathbf{T}}\mathbf{G}^{-\mathbf{1}}\mathbf{S}_{i+1} \tag{7.53}$$

on condition that :

$$\mathbf{R}^{\mathbf{T}}\mathbf{G}^{-\mathbf{1}}\mathbf{C} \equiv \mathbf{0} \tag{7.54}$$

Expression (7.52) ensures, first, that the Hamming weight of vector $(d_{1,i}, d_{2,i}, \cdots, d_{m,i}, y_i)$ is at least equal to 2 when we leave the reference path

("all zero" path), in the trellis. Indeed, inverting any component of $\mathbf{d}_i$ modifies the value of y_i. Second, expression (7.53) indicates that the Hamming weight of the same vector is also at least equal to 2 when we return to the reference path. In conclusion, relations (7.52) and (7.53) together guarantee that the free distance of the code, whose rate is $R = m/(m+1)$, is at least equal to 4, whatever m.

7.5.2 m-binary turbo codes

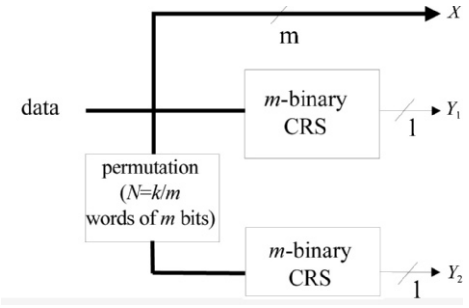

Figure 7.20 – m-binary turbo encoder.

We consider a parallel concatenation of two $m-$binary RSC enoders associated with a permutation as a function of N words of m bits ($k = mN$) (Figure 7.20). The blocks are encoded twice by this two-dimensional code, whose rate is $m/(m+2)$. The circular trellis principle is adopted to enable encoding of the blocks without a termination sequence and without edge effects.

The advantages of this construction compared to classical turbo codes are the following :

- **Better convergence**. This advantage, observed first in [7.9], commented in [7.16] and in a different way in [7.23], can be explained by a lower density of errors on each of the two dimensions of the iterative process. Take relation (7.8) that provides the upper bound of the accumulated spatial distance for a binary code and adapt it to an m-binary code:

$$\sup S_{\min} = \sqrt{\frac{2k}{m}} \tag{7.55}$$

For a coding rate R, the number of parity bits produced by the sequence with accumulated length $\sup S_{\min}$ is:

$$n_{parity}(\sup S_{\min}) = \left(\frac{1-R}{R}\right)\frac{m}{2}\sup S_{\min} = \left(\frac{1-R}{R}\right)\sqrt{\frac{mk}{2}} \tag{7.56}$$

Thus, replacing an ($m = 1$) binary turbo code by an ($m = 2$) double-binary code, the number of parity bits in the sequence considered is multiplied by $\sqrt{2}$, although the accumulated spatial distance has been reduced by the same ratio. Because the parity bits are local information for the two elementary decoders (and are therefore not a source of correlation between them), increasing the number of the former improves convergence. To increase m beyond 2 slightly improves behaviour concerning correlation but the effects are less visible than when passing from $m = 1$ to $m = 2$.

- **Larger minimum distances**. As explained above, the number of parity bits produced by the RTZ sequences of input weight 2 is increased by using m-binary codes. The same is true for all the simple RTZ sequences defined in Section 7.3.2. The number of parity bits for these sequences is at least equal to $n_{parity}(\sup S_{\min})$. The corresponding Hamming distances are therefore even higher than those obtained with binary codes and contribute even less to the MHD of the turbo code. As for the distances associated with multiple RTZ patterns, which are generally those that fix the MHD, they depend on the quality of the permutation implemented (see Section 7.3.2).

- **Less puncturing**. To obtain coding rates greater than $m/(m + 1)$ from the encoder of Figure 7.20, it is not necessary to suppress as many redundancy symbols as with a binary encoder. The performance of elementary codes is improved by this, as Figure 7.21 shows. This figure compares the correction capability of convolutional codes of rates 2/3 and 6/7, in the binary ($m = 1$) and double-binary ($m = 2$) versions.

- **Reduced latency**. From the point of view of encoding as well as decoding, the latency (that is, the number of clock cycles necessary for the processing) is divided by m since the data are processed in groups of m bits. However, it may happen that the critical path of the decoding circuit is increased compared to the case $m = 1$ as more data are to be considered in a clock cycle. Parallelism solutions, such as those proposed in [7.35], can help to increase the frequency of the circuit.

- **Robustness of the decoder**. For binary turbo codes, the difference in performance between the MAP algorithm and its simplified versions or between the MAP algorithm and the SOVA algorithm, vary from 0.2 to 0.6 dB, depending on the size of the blocks and the coding rates. This difference is divided by two when we use double-binary turbo codes and can be even lower for $m > 2$. This favourable (and slightly surprising) property can be explained as follows: for a block of a given size (k bits), the lower the number of steps in the trellis, the closer the decoder is to the Maximum Likelihood (ML) decoder, whatever the algorithm on which

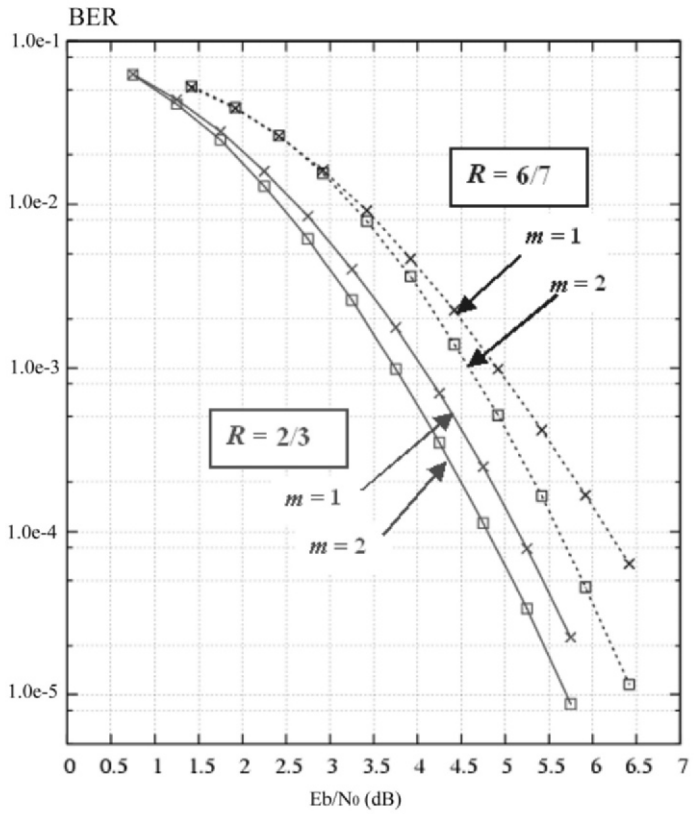

Figure 7.21 – Performance of simple binary ($m = 1$) and double-binary ($m = 2$) convolutional codes, for $R = 2/3$ and $R = 6/7$.

it is based. Ultimately, a trellis reduced to a single step and therefore containing all the possible codewords is equivalent to an ML decoder.

8-state double-binary turbo code

Figure 7.22(a) gives some examples of performance obtained with the turbo code of [7.2], for a rate 2/3. The parameters of the constituent encoders are:

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

The permutation function uses both inter- and intrasymbol interference. In particular, we can observe:

- good average performance for this code whose decoding complexity remains very reasonable (around 18,000 gates per iteration plus the memory);
- a certain coherence concerning the variation of performance with block size (in agreement with the curves of Figures 3.6, 3.9, 3.10). The same coherence could also be observed for the variation of performance with coding rate;
- quasi-optimality of decoding with low error rates. The theoretical asymptotic curve for 188 bytes has been calculated from the sole knowledge of the minimum distance of the code (that is, 13 with a relative multiplicity of 0.5) and not from the total spectrum of the distances. In spite of this, the difference between the asymptotic curve and the curve obtained by simulation is only 0.2 dB for a PER of 10^{-7}.

16-state double-binary turbo code

The extension of the previous scheme to 16-state elementary encoders allows the minimum distances to be greatly increased. We can, for example, choose:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

For the rate 2/3 turbo code, again with blocks of 188 bytes, the minimum distance obtained is equal to 18 (relative multiplicity of 0.75) instead of 13 for the 8-state code. Figure 7.22(b) shows the gain obtained for low error rates: around 1 dB for a PER of 10^{-7} and 1.4 dB asymptotically, considering the respective minimum distances. We can note that the convergence threshold is almost the same for 8-state and 16-state decoders, the curves being practically identical for a PER greater than 10^{-4}. The theoretical limit (TL), for $R = 2/3$ and for a blocksize of 188 bytes, is 1.7 dB. The performance of the decoder in this case is: TL + 0.9 dB for a PER of 10^{-4} and TL + 1.3 dB for a PER of 10^{-7}. These intervals are typical of what we obtain in most rate and blocksize configurations.

Replacing 4-PSK modulation by 8-PSK modulation, in the so-called pragmatic approach, gives the results shown in Figure 7.22(b), for blocks of 188 and 376 bytes. Here again, good performance of the double-binary code can be observed, with losses compared to the theoretical limits (that are around 3.5 and 3.3 dB, respectively) close to those obtained with 4-PSK modulation. Associating turbo codes with different modulations is described in Chapter 10.

For a particular system, the choice between an 8-state or 16-state turbo code depends, apart from the complexity desired for the decoder, on the target error

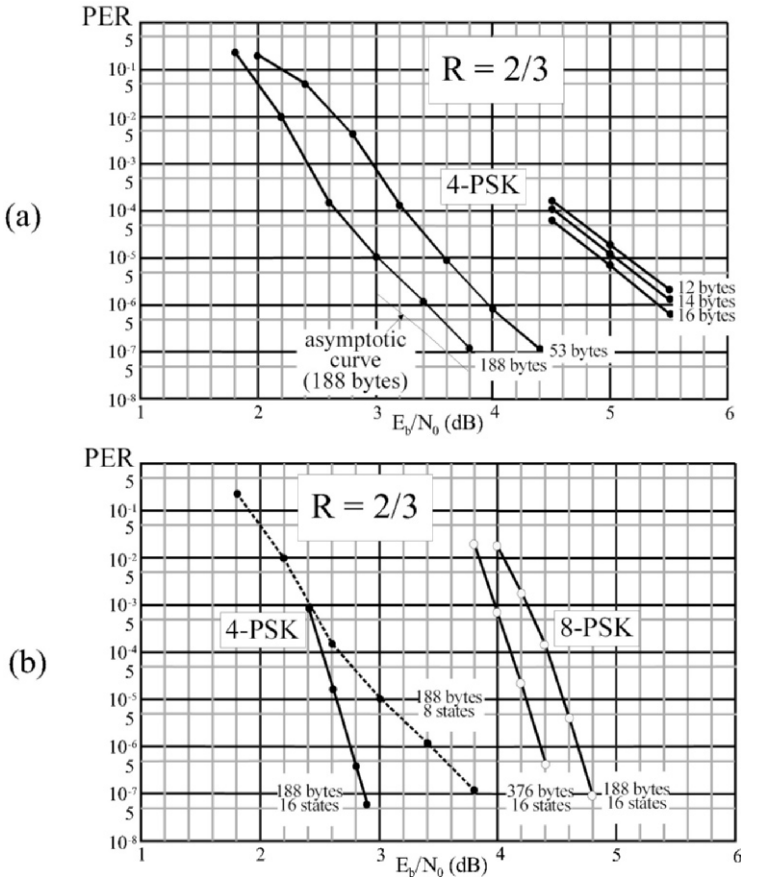

Figure 7.22 – (a) PER performance of a double-binary turbo code with 8 states for blocks of 12, 14, 16, 53 and 188 bytes. 4-PSK, AWGN noise and rate 2/3. Max-Log-MAP decoding with input samples of 4 bits and 8 iterations. (b) PER performance of a double-binary turbo code with 16 states for blocks of 188 bytes (4-PSK and 8-PSK) and 376 bytes (8-PSK), AWGN noise and rate 2/3. Max-Log-MAP decoding with input samples of 4 bits (4-PSK) or 5 bits (8-PSK) and 8 iterations.

rates. To simplify, let us say that an 8-state turbo code suffices for PERs greater than 10^{-4}. This is generally the case for transmissions having the possibility of repetitions (ARQ: Automatic Repeat reQuest). For lower PERs, typical of broadcasting or of mass memory applications, the 16-state code is highly preferable.

7.6 Analysis tools

7.6.1 Theoretical performance

Figure 1.6 shows two essential parameters allowing the performance of an error correcting code and its decoder to be evaluated:

- the *asymptotic gain* measuring the behaviour of the coded system at low error rates. This is mainly dictated by the MHD of the code (see Section 1.5). A low value of the MHD leads to a great change in the slope (*flattening*) in the error rate curve. When the asymptotic gain is reached, the BER(E_b/N_0) curve with coding becomes parallel to the curve without coding.
- the *convergence threshold* defined as the signal to noise ratio from which the coded system becomes more efficient than the non-coded transmission system;

In the case of turbo codes and the iterative process of their decoding, it is not always easy to estimate the performance either of the asymptotic gain or of the convergence. Methods for estimating or determining the minimum distance proposed by Berrou *et al.* [7.18], Garello *et al.* [7.27] and Crozier *et al.* [7.22] are presented in the rest of this chapter. The EXIT diagram method proposed by ten Brink [7.46] to estimate the convergence threshold is also introduced.

7.6.2 Asymptotic behaviour

Determining the performance of error correcting codes with low error rates by simulation requires high calculation power. It is, however, possible to estimate this performance when the MHD $d_{\min}$ and the multiplicity are known (see Section 1.5). Thus, the packet error rate with high signal to noise ratio E_b/N_0 is given by the first term of the *union bound* (UB). The expression of the UB is described by relation (3.21), and estimation of the PER, given by Equation (1.16), is shown again here:

$$\mathrm{PER} \approx \frac{1}{2} N(d_{\min}) \operatorname{erfc}\left(\sqrt{R d_{\min} \frac{E_b}{N_0}}\right) \tag{7.57}$$

where $N(d_{\min})$, the multiplicity, represents the number of codewords at the minimum distance.

The minimum distance of a code is not, in the general case, simple to determine except if the number of codewords is low enough for us to make an exhaustive list of them, or if particular properties of the code enable us to establish an analytical expression of this value (for example, the minimum distance

of a product code is equal to the product of the minimum distances of the constituent codes). In the case of convolutional turbo codes, the minimum distance is not obtained analytically; the only methods proposed are based on the total or partial [7.28] enumeration of codewords whose input weight is lower than or equal to the minimum distance. These methods are applicable in practice only for small sizes blocksizes and small minimum distances, which is why they will not be described here.

Error impulse method

This method, proposed by Berrou *et al.* [7.18], is not based on the analysis of the properties of the code but on the correction capacity of the decoder. Its principle, illustrated in Figure 7.23, involves superposing on the input sequence of the decoder an error impulse whose amplitude A_i is increased until the decoder no longer knows how to correct it.

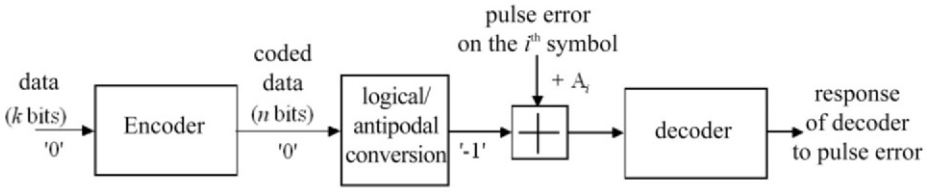

Figure 7.23 – Schematic diagram of the error impulse method.

The code considered being linear, the sequence transmitted is assumed to be the "all zero" sequence. The coding operation then produces codewords that also contain only zeros. These are next converted into real values equal to -1. If this succession of symbols was directly applied at the decoder, the latter would not encounter any difficulty in retrieving the original message since the transmission channel is perfect.

The proposed method involves adding an error impulse to the i-th symbol ($0 \leq i \leq k-1$) of the information sequence (systematic part), that is, transforming a "-1" symbol into a symbol having a positive value equal to $-1+A_i$. If amplitude A_i is high enough, the decoder does not converge towards the "all zero" word. Let us denote A_i^* the maximum amplitude of the impulse in position i such that the decoded codeword is the "all zero" word. It is shown in [7.18] that, if the decoder performs maximum likelihood decoding, *impulse distance* $d_{imp} = \min_{i=0,\cdots,k-1}(A_i^*)$ is also the minimum distance d_{min} from the code.

It is generally not necessary to test all the positions of the sequence. For a shift invariant code (which is the case of convolutional codes), it suffices to apply the error impulse to just one position of the datablock. For a code presenting a periodicity of period P, it is necessary to test P positions. This method is appli-

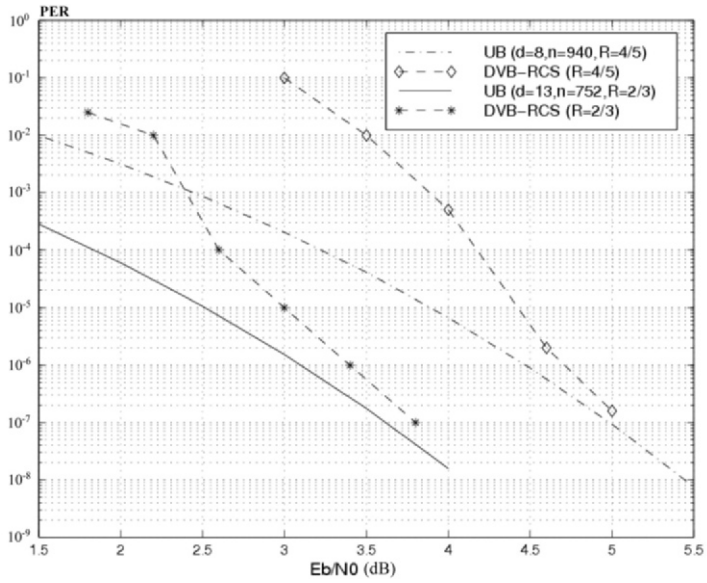

Figure 7.24 – Measured and estimated PER (UB) of the DVB-RCS turbo code for the transmission of MPEG (188 bytes) blocks with coding rates 2/3 and 4/5. 4-PSK modulation and Gaussian channel.

cable to any linear code, for any blocksize and any coding rate, and it requires only a few seconds to a few minutes calculation on an ordinary computer, the calculation time being a linear function of the blocksize or of its period P.

When the decoding is not maximum likelihood, this method is no longer rigorous and produces only an estimation of the minimum distance. In addition, the multiplicity of the codewords at distance $d_{\min}$ is not provided and Equation (7.57) cannot be applied without particular hypotheses about the properties of the code. In the case of turbo codes, two realistic hypotheses can be formulated to estimate multiplicity: a single codeword at distance A_i^* has its i-th information bit at 1 (unicity), and the A_i^* values corresponding to all positions i come from distinct codewords (non-overlapping).

An estimation of the PER is then given by:

$$\text{PER} \approx \frac{1}{2} \sum_{i=0}^{k-1} \text{erfc}(\sqrt{RA_i^* \frac{E_b}{N_0}}) \tag{7.58}$$

The first hypothesis (unicity) under-evaluates the value of the error rate, unlike the second (non-overlapping) that over-evaluates it and, overall, the two effects compensate each other. As an example, Figure 7.24 compares the measured performance of the DVB-RCS turbo code, for two coding rates, with their estimate

deduced from (7.58). The parameters obtained by the error impulse method are:

- $d_{\min} = 13$ and $n(d_{\min}) = 752$ for $R = 2/3$
- $d_{\min} = 8$ and $n(d_{\min}) = 940$ for $R = 4/5$

For packet error rates of 10^{-7}, less than 0.2 dB separates the measured and estimated curves.

Modified error impulse method

The approach of Garello *et al.* [7.27] is similar to the error impulse method presented above. It involves placing an impulse in row i in the "all zero" codeword. This time, the amplitude of the impulse is high enough for the decoder not to converge towards the "all zero" codeword but towards another sequence that contains a 1 in position i. In addition, Gaussian noise is added to the input sequence of the decoder, which tends to help the latter converge towards the concurrent word having the lowest weight. This is what often happens when the level of noise is well adjusted. In all cases, the weight of the codeword provided by the decoder is an upper limit of the minimum distance of all the codewords containing a 1 in row i. The minimum distance and the multiplicity are estimated by sweeping all the positions. This algorithm works very well for small and average distances.

Double error impulse method

The method proposed by Crozier *et al.* [7.22] is an improvement of the previous method, at the expense of higher computation time. It involves placing a first high level impulse at row i and a second at row j to the right of i and such that $j - i < r$. The upper limit of r is $2D$ where D is an upper bound of the distance to be evaluated. Then, decoding is applied similar to that described above but with a stronger probability of obtaining a codeword at the minimum distance. The calculation time is increased by a ratio r.

7.6.3 Convergence

A SISO decoder can be seen as a processor that transforms one of its input values, the LLR of the extrinsic information used as *a priori* information, into an output extrinsic LLR. In iterative decoding, the characteristics of the extrinsic information provided by decoder 1 depend on the extrinsic information provided by decoder 2 and vice-versa. The degree of dependency between the input and output extrinsic information can be measured by the mutual information (MI).

The idea implemented by ten Brink [7.46] is to follow the exchange of extrinsic information through the SISO decoders working in parallel on a diagram,

called an *EXtrinsic Information Transfer* (EXIT) chart. To elaborate the EXIT chart, it is necessary to know the transfer characteristics of the extrinsic information of each SISO decoder used in the decoding. This section shows how to establish the transfer function of the extrinsic information for a SISO decoder, then construct the EXIT chart, and finally analyse the convergence of the iterative decoder.

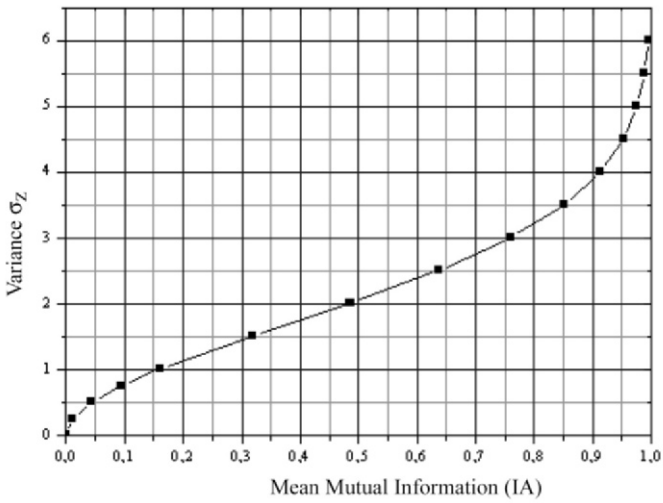

Figure 7.25 – Variation of variance σ_z as a function of the the mutual information IA

Transfer function for a SISO decoder of extrinsic information

a. Definition of the mutual information (MI)

If the weighted extrinsic information z on the coded binary element $x \in \{-1, +1\}$ follows a conditional probability density $f(z|x)$, the MI $I(z, x)$ measures the quantity of information provided on average by z on x and equals

$$\mathrm{I}(z,x) = \frac{1}{2} \sum_{x=-1,+1} \int_{-\infty}^{+\infty} f(z|x) \times \log_2 \left[\frac{2f(z|x)}{f(z|-1) + f(z|+1)} \right] dz \qquad (7.59)$$

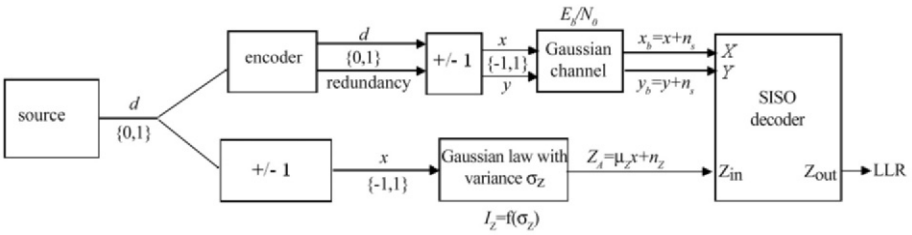

Figure 7.26 – Algorithm for determining the transfer function $IE = T(IA, Eb/N_0)$

b. Definition of the a priori *mutual information*

Hypotheses:

- Hyp. 1: when the interleaving is large enough, the distribution of the input extrinsic information can be approximated by a Gaussian distribution after a few iterations.
- Hyp. 2: probability density $f(z|x)$ satisfies the exponential symmetry condition, that is, $f(z|x) = f(-z|x)exp(-z)$.

The first hypothesis allows the *a priori* LLR Z_A of a SISO decoder to be modelled by a variable having independent Gaussian noise n_z, with variance σ_z and expectation μ_z, applied to the transmitted information symbol x according to the expression

$$Z_A = \mu_z x + n_z$$

The second hypothesis imposes $\sigma_z^2 = 2\mu_z$. The amplitude of the extrinsic information is therefore modelled by the following distribution:

$$f(\lambda|x) = \frac{1}{\sqrt{4\pi\mu_z}} \exp\left[-\frac{(\lambda - \mu_z x)^2}{4\mu_z}\right] \tag{7.60}$$

From (7.59) and (7.60), observing that $f(z|\ 1) = f(-z|-1)$, we deduce the *a priori* mutual information:

$$\mathrm{IA} = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{4\pi\mu_z}} \exp\left[-\frac{(\lambda - \mu_z)^2}{4\mu_z}\right] \times \log_2\left[\frac{2}{1+\exp(-\lambda)}\right] d\lambda$$

or again

$$\mathrm{IA} = 1 - \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_z} \exp\left[-\frac{(\lambda - \sigma_z^2/2)^2}{2\sigma_z^2}\right] \times \log_2\left[1+\exp(-\lambda)\right] d\lambda \tag{7.61}$$

We can note that $\lim_{\sigma_z \to 0} \text{IA} = 0$ (the extrinsic information does not provide any information about datum x) and that $\lim_{\sigma_z \to +\infty} \text{IA} = 1$ (the extrinsic information perfectly determines datum x).
IA is an increasing monotonous function of σ_z; it is therefore invertible. Function $\sigma_z = f(\text{IA})$ is shown in Figure 7.25.

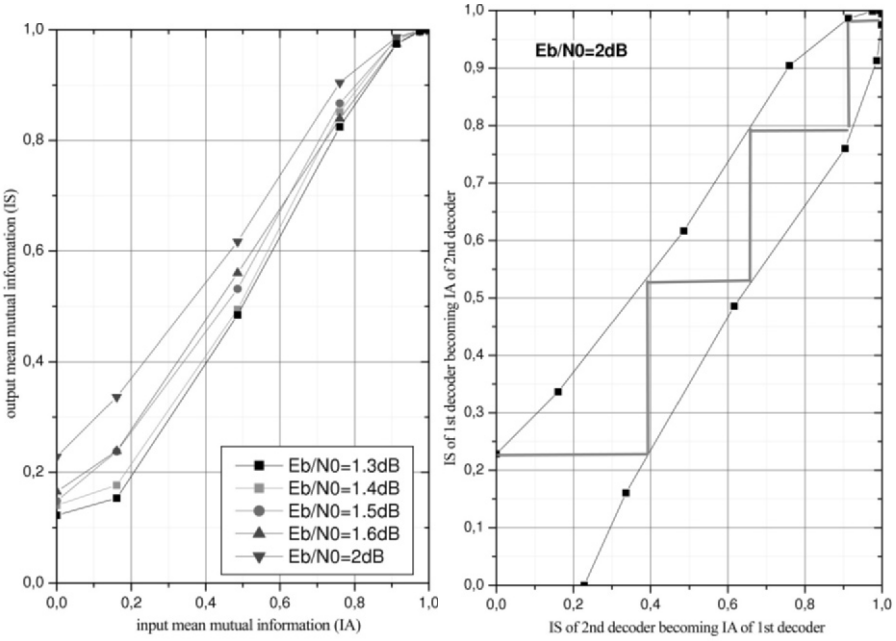

Figure 7.27 – (a) Transfer characteristic of the extrinsic information for a 16-state binary encoder, with rate 2/3 and MAP decoding with different Eb/N0.
(b) EXIT chart and trajectory for the corresponding turbo code, with pseudo-random interleaving on 20,000 bits, for $E_b/N_0 = 2dB$.

c. Calculation of the output mutual information

Relation (7.59) allows the calculation of the mutual information IS linked with the extrinsic information produced by the SISO decoder:

$$\text{IS} = \frac{1}{2} \sum_{x=-1,+1} \int_{-\infty}^{+\infty} f_s(z|x) \times \log_2 \left[\frac{2 f_s(z|x)}{f_s(z|-1) + f_s(z|+1)} \right] dz \qquad (7.62)$$

We can note that $\text{IS} \in [0, 1]$.

The distribution f_s is not Gaussian. It is therefore necessary to use a digital calculation tool to determine it, which is the great drawback of this method.

If we view the MI of output IS as a function of IA and of the signal to noise ratio E_b/N_0, the transfer function of the extrinsic information is defined by:

$$IE = T(IA, E_b/N_0) \tag{7.63}$$

d. Practical method to obtain the transfer function of the extrinsic information

Figure 7.26 shows the path taken to establish the transfer characteristic of the extrinsic information of a SISO decoder.

- step 1: Generation of the pseudo random message d to be transmitted; at least 10000 bits are necessary for the statistical properties to be representative.
- step 2: Encoding the data with rate R then 2-PSK modulation of the signal; the systematic and redundancy data both belong to the alphabet {-1,+1}.
- step 3: Application of a Gaussian noise with signal to noise ratio E_b/N_0 (in dB), with variance

$$\sigma = \sqrt{\frac{1}{2} \cdot \frac{10^{-0,1 \times E_b/N_0}}{R}}$$

- step 4: Application to the data transmitted (stored in a file) of normal law $N(\mu_z, \sigma_z)$ corresponding to the mutual information IA desired (see Figure 7.25) to obtain the distribution of *a priori* extrinsic information.
- step 5: Initialization of the SISO decoder with the *a priori* LLRs (it might be necessary, depending on the decoding algorithm chosen, to transform the LLRs into probabilities).
- step 6: Recovering the LLRs at the output of the SISO decoder (corresponding to one half-iteration of the decoding process), in a file.
- step 7: Utilization of digital calculation software to evaluate IS (relation (7.62)).
    - Trace the histograms of the LLR distributions output as a function of the bit transmitted (hence the necessity to store this information in two files).
    - Evaluate the integral by the trapeze method.

- The result is the MI of output IS corresponding to the MI of input IA.

e. An example

The simulations were performed on a 16-state binary turbo code with rate 2/3, with a pseudo-random interleaving of 20,000 bits. The decoding algorithm is the MAP algorithm. Figure 7.27(a) shows the relation between IS and IA as a function of the signal to noise ratio of the Gaussian channel.

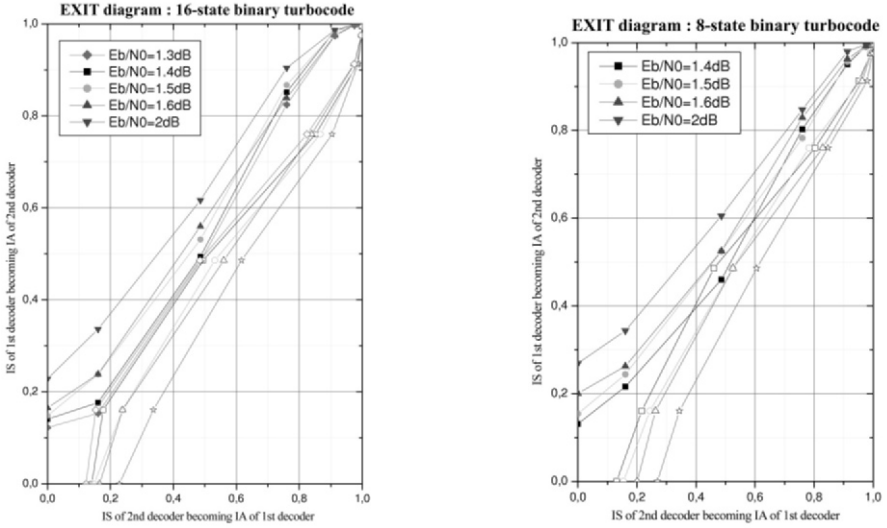

Figure 7.28 – EXIT charts for different E_b/N_0 in the case of binary turbo codes, rate 2/3, pseudo-random interleaving of 20,000 bits, (a) 16-state and (b) 8-state MAP decoding.

EXIT chart

The extrinsic information transfer characteristic is now known for a SISO decoder. In the case of iterative decoding, the output of decoder 1 becomes the input of decoder 2 and vice versa. Curves IS1 = $f(I$A1 = IS2) and IS2 = f(IA2 = IS1), identical to one symmetry if the SISO decoders are the same, are placed on the same graph as shown in Figure 7.27(b). In the case of a high enough signal to noise ratio (here 2 dB), the two curves do not have any intersection outside the point of coordinates (1,1) which materializes the knowledge of the received message. Starting from null mutual information, it is then possible to follow the exchange of extrinsic information along the iter-

ations. In the example of Figure 7.27(b), arrival at point (1,1) is performed in 3.5 iterations.

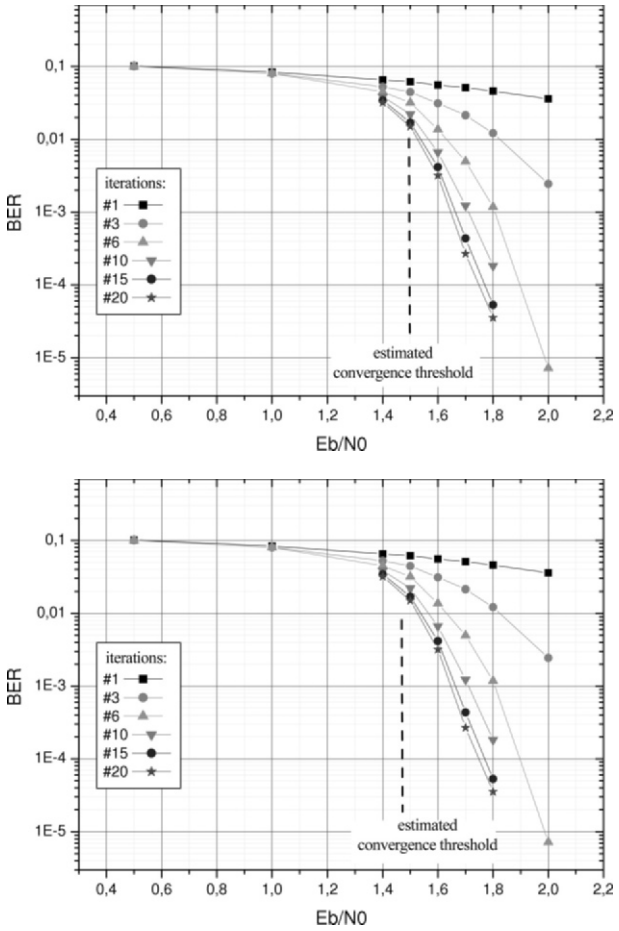

Figure 7.29 – Binary error rates of a 16-state (a) and 8-state (b) binary turbo code with rate 2/3, with pseudo-random interleaving of 20000 bits. MAP decoding with 1, 3, 6, 10, 15 and 20 iterations and comparison with the convergence threshold estimated by the EXIT method.

When the signal to noise ratio is too low, as in case $E_b/N_0 = 1.4$ dB in Figure 7.28(b), the curves have intersection points other than point $(1, 1)$. The iterative process starting from null MI at the input will therefore not be able to lead to a perfectly determined message. The minimum signal to noise ratio for which there is no intersection other than point (1,1) is the convergence threshold of the turbo encoder. In the simulated example, this convergence can be esti-

mated at around 1.4 dB for 16-state (Figure 7.28(a)) and 8-state (Figure 7.28(b)) binary turbo codes.

Figure 7.29 shows the performance of 16-state and 8-state binary turbo codes as a function of the number of iterations, and compares them with the convergence threshold estimated by the EXIT chart method.

Bibliography

[7.1] Interaction channel for digital terrestrial television. DVB, ETSI EN 301 958, V1.1.1, pp. 28-30, Aug. 2001.

[7.2] Interaction channel for satellite distribution systems. DVB, ETSI EN 301 790, V1.2.2, pp. 21-24, Dec. 2000.

[7.3] Multiplexing and channel coding (fdd). 3GPP Technical Specification Group, TS 25.212 v2.0.0, June 1999.

[7.4] Recommendations for space data systems. telemetry channel coding. Consultative Committee for Space Data Systems, BLUE BOOK, May 1998.

[7.5] Ieee standard for local and metropolitan area networks, IEEE Std 802.16a, 2003. Available at http://standards.ieee.org/getieee802/download/802.16a-2003.pdf.

[7.6] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, IT-20:284–287, March 1974.

[7.7] G. Battail. Weighting of the symbols decoded by the viterbi algorithm. *Annals of Telecommunications*, 42(1-2):31–38, Jan.-Feb. 1987.

[7.8] G. Battail. Coding for the gaussian channel: the promise of weighted-output decoding. *International Journal of Satellite Communications*, 7:183–192, 1989.

[7.9] C. Berrou. Some clinical aspects of turbo codes. In *Proceedings of 3rd International Symposium on turbo codes & Related Topics*, pages 26–31, Brest, France, Sept. 1997.

[7.10] C. Berrou, P. Adde, E. Angui, and S. Faudeuil. A low complexity soft-output viterbi decoder architecture. In *Proceedings of IEEE International Conference on Communications (ICC'93)*, pages 737–740, GENEVA, May 1993.

[7.11] C. Berrou, C. Douillard, and M. Jézéquel. Designing turbo codes for low error rates. In *IEE colloquium : turbo codes in digital broadcasting – Could it double capacity?*, pages 1–7, London, Nov. 1999.

[7.12] C. Berrou, C. Douillard, and M. Jézéquel. Multiple parallel concatenation of circular recursive convolutional (crsc) codes. *Annals of Telecommunications*, 54(3-4):166–172, March-Apr. 1999.

[7.13] C. Berrou and A. Glavieux. Reflections on the prize paper: "near optimum error correcting coding and decoding: turbo codes". *IEEE IT Society Newsletter*, 48(2):136–139, June 1998. www.ieeeits.org/publications/nltr/98_jun/reflections.html.

[7.14] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: turbo-codes. In *Proceedings of IEEE International Conference on Communications (ICC'93)*, pages 1064–1070, GENEVA, May 1993.

[7.15] C. Berrou and M. Jézéquel. Frame-oriented convolutional turbo-codes. *Electronics letters*, 32(15):1362–1364, July 1996.

[7.16] C. Berrou and M. Jézéquel. Non binary convolutional codes for turbo coding. *Electronics Letters*, 35(1):39–40, Jan. 1999.

[7.17] C. Berrou, Y. Saouter, C. Douillard, S. Kerouédan, and M. Jézéquel. Designing good permutations for turbo codes: towards a single model. In *Proceedings of IEEE International Conference on Communications (ICC'04)*, Paris, France, June 2004.

[7.18] C. Berrou, S. Vaton, M. Jézéquel, and C. Douillard. Computing the minimum distance of linear codes by the error impulse method. In *Proceedings of IEEE Global Communication Conference (Globecom'2002)*, pages 1017–1020, Taipei, Taiwan, Nov. 2002.

[7.19] E. Boutillon and D. Gnaedig. Maximum spread of d-dimensional multiple turbo codes. *IEEE Transanction on Communications*, 53(8):1237–1242, Aug. 2005.

[7.20] A. R. Calderbank. The art of signaling: fifty years of coding theory. *IEEE Transactions on Information Theory*, 44(6):2561–2595, Oct. 1998.

[7.21] S. Crozier and P. Guinand. Distance upper bounds and true minimum distance results for turbo-codes with drp interleavers. In *Proceedings of 3rd International Symposium on turbo codes & Related Topics*, pages 169–172, Brest, France, Sept. 2003.

[7.22] S. Crozier, P. Guinand, and A. Hunt. Computing the minimum distance of turbo-codes using iterative decoding techniques. In *Proceedings of the 22nd Biennial Symposium on Communications*, pages 306–308, Kingston, Ontario, Canada, 2004,May 31-June3.

[7.23] C. Douillard and C. Berrou. Turbo codes with rate-m/(m+1) constituent convolutional codes. *IEEE Trans. Commun.*, 53(10):1630–1638, Oct. 2005.

[7.24] L. Duan and B. Rimoldi. The iterative turbo decoding algorithm has fixed points. *IEEE Transactions on Information Theory*, 47(7):2993–2995, Nov. 2001.

[7.25] P. Elias. Error-free coding. *IEEE Transactions on Information Theory*, 4(4):29–39, Sept. 1954.

[7.26] R. G. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, IT-8:21–28, Jan. 1962.

[7.27] R. Garello and A. Vila Casado. The all-zero iterative decoding algorithm for turbo code minimum distance computation. In *Proceedings of IEEE International Conference on Communications (ICC'2004)*, pages 361–364, Paris, France, 2004.

[7.28] R. Garello, P. Pierloni, and S. Benedetto. Computing the free distance of turbo codes and serially concatened codes with interleavers: Algorithms and applications. *IEEE Journal on Selected Areas in Communications*, May 2001.

[7.29] K. Gracie and M.-H. Hamon. Turbo and turbo-like codes: Principles and applications in telecommunications. In *Proceedings of the IEEE*, volume 95, pages 1228–1254, June 2007.

[7.30] J. Hagenauer and P. Hoeher. Concatenated viterbi-decoding. In *Proceedings of International Workshop on Information Theory*, pages 136–139, Gotland, Sweden, Aug.-Sept. 1989.

[7.31] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. In *Proceedings of IEEE Global Communications Conference (Globecom'89)*, pages 1680–1686, Dallas, Texas, USA, Nov. 1989.

[7.32] J. Hagenauer and P. Hoeher. Turbo decoding with tail-biting trellises. In *Proceedings of Signals, Systems and Electronics, URSI Int'l Symposium*, pages 343–348, Sept.-Oct. 1998.

[7.33] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers.* Oxford Univ. Press, Oxford, UK, 1979.

[7.34] C. Heegard and S. B. Wicker. Chapter 3. In *Turbo coding.* Kluwer Academic Publishers, 1999.

[7.35] H. Lin and D. G. Messerschmitt. Algorithms and architectures for concurrent viterbi decoding. In *Proceedings of IEEE International Conference on Communications (ICC'89)*, pages 836–840, Boston, June 1989.

[7.36] R. L. Rivest. Permutation polynomials modulo 2w. *Finite Fields Their Applications*, 7:287–292, Apr. 2001.

[7.37] P. Robertson, P. Hoeher, and E. Villebrun. Optimal and suboptimal maximum a posteriori algorithms suitable for turbo decoding. *European Transactions on Telecommunication*, 8:119–125, March-Apr. 1997.

[7.38] E. Rosnes and O. Y. Takeshita. Optimum distance quadratic permutation polynomial-based interleavers for turbo codes. In *Proceedings of IEEE International Symposyum on Information Theory (ISIT'07)*, pages 1988–1992, Seattle, Washington, July 2006.

[7.39] J. Ryu and O. Y. Takeshita. On quadratic inverses for quadratic permutation polynomials over integer rings. *IEEE Transactions On Information Theory*, 52:1254–1260, March 2006.

[7.40] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe. Interleaver design for turbo codes. *IEEE Journal on Selected Areas in Commununications*, 19(5):831–837, May 2001.

[7.41] J. Sun and O. Y. Takeshita. Interleavers for turbo codes using permutation polynomials over integer rings. *IEEE Transactions On Information Theory*, 51:101–119, Jan. 2005.

[7.42] Y. V. Svirid. Weight distributions and bounds for turbo-codes. *European Transactions on Telecommunication*, 6(5):543–55, Sept.-Oct. 1995.

[7.43] O. Y. Takeshita. On maximum contention-free interleavers and permutation polynomials over integer rings. *IEEE Transactions On Information Theory*, 52:1249–1253, March 2006.

[7.44] O. Y. Takeshita. Permutation polynomial interleavers: an algebraic-geometric perspective. *IEEE Transactions On Information Theory*, 53:2116–2132, June 2007.

[7.45] R. M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, IT-271:533–547, Sept. 1981.

[7.46] S. ten Brink. Convergence behavior of iteratively decoded parallel concatenated codes. *IEEE Transactions On Communications*, 49(10), Oct. 2001.

[7.47] Z. Wang, Z. Chi, and K. K. Parhi. Area-efficient high-speed decoding schemes for turbo decoders. *IEEE Trans. VLSI Systems*, 10(6):902–912, Dec. 2002.

[7.48] C. Weiss, C. Bettstetter, and S. Riedel. Code constuction and decoding of parallel concatenated tail-biting codes. *IEEE Comm. Letters*, 47(1):366–386, Jan. 2001.

[7.49] Y. Weiss and W. T. Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, Feb. 2001.