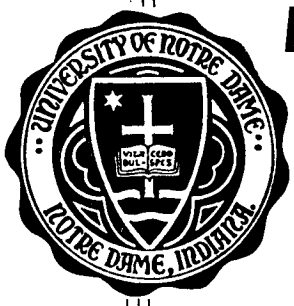
# ANNUAL STATUS REPORT

## Error Control Techniques for Satellite and Space Communications
## NASA Grant Number NAG5-557

Principal Investigator:
Daniel J. Costello, Jr.

December 1995






*Department of*
**ELECTRICAL ENGINEERING**

UNIVERSITY OF NOTRE DAME, NOTRE DAME, INDIANA

## Summary of Progress

In this report, we will focus on the results obtained during the PI's recent sabbatical leave at the Swiss Federal Institute of Technology (ETH) in Zurich, Switzerland, from January 1, 1995 through June 30, 1995. During this time period the PI and his postdoctoral research associate, Dr. Lance C. Perez, supervised one semester project and one diploma project involving three ETH students: Dieter Arnold, Guido Meyerhans, and Jan Seghers. Both projects investigated various properties of TURBO codes, a new form of concatenated coding that achieves near channel capacity performance at moderate bit error rates. The semester project, entitled "The Realization of the Turbo-Coding System" and authored by Mr. Arnold and Mr. Meyerhans, involved a thorough simulation study of the performance of TURBO codes and verified the results claimed by previous authors. A copy of the final report for this project is included as Appendix A to this report. The diploma project, entitled "On the Free Distance of Turbo Codes and Related Product Codes" and authored by Mr. Seghers, includes an analysis of TURBO codes and an explanation for their remarkable performance. A copy of the final report for this project is included as Appendix B to this report. Recently, the PI presented a paper on this research at the 1995 Allerton Conference [8]. As a result of this presentation, we have been invited to submit a paper on TURBO codes to the special issue of the IEEE Transactions on Information Theory on "Codes and Complexity" scheduled for publication in November 1996. The following sections contain a brief summary of this research.

# 1 Introduction

The discovery of TURBO codes and the near capacity performance reported by the discoverers [1] has stimulated significant research effort to fully understand this new coding scheme [2]-[6]. In this paper, the performance of TURBO codes is explained in terms of the code's distance spectrum. These results explain both the near capacity performance of the TURBO codes and the observed "error floor" for moderate and high signal-to-noise ratios (SNR's).

# 2 Performance Bounds

In order to make clear the distinction between TURBO codes and convolutional codes, it is useful to consider these codes as block codes. To this end, the inputs sequences are restricted to sequences of length $N$, where $N$ corresponds to the size of the interleaver in the TURBO encoder. With finite length input sequences of length $N$, a $(2, 1, \nu)$ convolutional code may be viewed as a block code with $2^N$ codewords of length $2(\nu + N)$.

The bit error rate (BER) performance of the convolutional code with maximum likelihood (ML) decoding is upper bounded by

$$P_b \leq \sum_{i=1}^{2^N} \frac{w_i}{N} \frac{1}{2} erfc\left(\sqrt{d_i \frac{RE_b}{N_0}}\right)$$

where $w_i$ and $d_i$ are the information weight and total Hamming weight, respectively, of the $i^{th}$ codeword. Collecting codewords of the same total Hamming weight and defining the average information weight per codeword as

$$\tilde{w}_d = \frac{w_d}{N_d},$$

where $w_d$ is the total information weight of all codewords of weight $d$ and $N_d$ is the number of codewords of weight $d$, yields

$$P_b \leq \sum_{d=d_{free}}^{\infty} \frac{N_d}{N} \frac{\tilde{w}_d}{2} erfc\left(\sqrt{d \frac{RE_b}{N_0}}\right).$$

If a convolutional code has $N_d^0$ codewords of weight $d$ caused by an information sequence $x(D)$ whose first one occurs at time 0, then it also has a codeword of weight $d$ caused by the information sequence $Dx(D)$, a codeword of weight $d$ caused by the information sequence $D^2x(D)$, and so on. Thus, as the length of the information sequences increases we have

$$\lim_{N \to \infty} \frac{N_d}{N} = N_d^0$$

and

$$\lim_{N \to \infty} \tilde{w}_d = \frac{w_d}{N_d} = \frac{w_d^0}{N_d^0}$$

and the bound on the BER of a convolutional code with ML decoding becomes

$$P_b \leq \sum_{d=d_{free}}^{\infty} \frac{w_d^0}{2} erfc\left(\sqrt{d \frac{RE_b}{N_0}}\right),$$

which is the standard union bound for ML decoding. For this reason, most efforts to find good convolutional codes have focused on finding codes that maximize the free distance $d_{free}$ and minimize the multiplicity $N_d$ for a given rate and constraint length.

The performance of the TURBO code with ML decoding is also bounded by

$$P_b \leq \sum_{i=1}^{2^N} \frac{w_i}{N} \frac{1}{2} erfc\left(\sqrt{d_i \frac{RE_b}{N_0}}\right)$$

where $w_i$ and $d_i$ are the information weight and total Hamming weight, respectively, of the $i^{th}$ codeword. However, in the TURBO encoder the pseudorandom interleaver maps the sequence $x(D)$ to $x'(D)$ and the sequence $Dx(D)$ to a sequence $x''(D)$ that is different from $Dx'(D)$ with very high probability. Thus, the input sequences $x(D)$ and $Dx(D)$ produce different codewords with different Hamming weights.

Collecting codewords of the same total Hamming weight, the bound on the BER for TURBO codes becomes

$$P_b \leq \sum_{d=d_{free}}^{\infty} \frac{N_d}{N} \frac{\tilde{w}_d}{2} erfc\left(\sqrt{d \frac{RE_b}{N_0}}\right).$$

For TURBO codes with pseudorandom interleavers, $N_d \tilde{w}_d$ is much less than $N$ and

$$\lim_{N \to \infty} \frac{\tilde{w}_d N_d}{N} << 1,$$

where

$$\frac{\tilde{w}_d N_d}{N}$$

is called the *effective multiplicity* of codewords of weight $d$.

# 3 Asymptotic Performance

In this section, the performance of TURBO codes for high SNR's is addressed using the bounds developed in the previous section. In order to make the points more concrete, the performance of the TURBO code is compared to the performance of a maximum free distance (MFD) $(2, 1, 14)$ convolutional code using simulations and analytical results. Figure 1 shows simulation results for the $(2, 1, 14)$ code with soft-decision Viterbi decoding and the TURBO code of [1] with iterative decoding and $N = 65536$.

It is well known that for high SNR's, the performance of a convolutional code with ML decoding is determined by the *free distance asymptote*. That is, for high SNR's the first term of the union bound is dominant. For the MFD $(2, 1, 14)$ code the first term in the bound is given by

$$\frac{w^0_{free}}{2} erfc\left(\sqrt{d_{free} \frac{RE_b}{N_0}}\right),$$

where $d_{free} = 18$, $N^0_{free} = 18$, and $w^0_{free} = 137$. In Figure 2, simulation results for this code are shown along with the free distance asymptote and it is clear that as the SNR gets large the asymptote accurately predicts the performance of the code.

For the TURBO code, the first term in the union bound is given by

$$\frac{N_{free}}{N} \frac{\tilde{w}_{free}}{2} erfc\left(\sqrt{d_{free} \frac{RE_b}{N_0}}\right),$$

where $N$ is the size of the interleaver. For $N = 65536$, is has been found [7] that the TURBO code has $N_{free} = 3$ and $\tilde{w} = 2$. (This may vary slightly depending on which pseudorandom interleaver is used.) In Figure 3, the free distance asymptote is plotted together with simulation results. Once again, the asymptote is seen to be a good estimate of the performance of the code for high SNR's. Thus, it can be concluded that the 'error floor' of TURBO codes is due to their relatively low free distance and the corresponding free distance asymptote.

It can be shown [7] that as the interleaver size $N$ increases, the number of free distance codewords approaches a constant $K_{free}$ that is much less than $N$. That is,

$$\lim_{N \to \infty} N_{free} = K_{free}.$$

(This is not true for rectangular interleavers!) Thus, the 'error floor' can be lowered by increasing the size of the interleaver. Simulation results have shown that this is indeed the case.

# 4 Performance at Low SNR's

Having demonstrated that TURBO codes have a relatively small free distance, their excellent performance at low SNR's may seem even more surprising. In this section, the performance of TURBO codes at low SNR's is explained by examining the code's distance spectrum.

Returning to the MFD $(2, 1, 14)$ convolutional code and Figure 2, it is clear that for low and moderate SNR's there is a significant gap between the free distance asymptote and the simulated performance. That is, the real coding gain is less than the asymptotic coding gain. The $(2, 1, 14)$ code has the following distance spectrum

| $d$ | $N_d^0$ | $w_d^0$ |
|---|---|---|
| 18 | 33 | 187 |
| 20 | 136 | 1034 |
| 22 | 835 | 7857 |
| 24 | 4787 | 53994 |
| 26 | 27941 | 361762 |
| 28 | 162513 | 2374453 |
| 30 | 945570 | 15452996 |
| 32 | 5523544 | 99659236 |

where $N_d^0$ is the number of paths of weight $d$ and $w_d^0$ is the total information weight. In Figure 4, the contribution of each term in the distance spectrum to the overall performance of the code is plotted and compared to simulation results. It is easily seen that for SNR's less than 2.5 dB the free distance asymptote is the dominant term in the performance. This is due to the rapid growth of the multiplicities in the distance spectrum which causes the high weight paths to become dominant for low and moderate SNR's.

In view of the analysis of the $(2, 1, 14)$ code, it is reasonable to suggest that the excellent performance of the TURBO code at low and moderate SNR's is due to a relatively "thin" distance spectrum. That is, the TURBO code is able to follow the free distance asymptote at lower SNR's because the multiplicities of higher weight codewords are small enough that the free distance asymptote remains the dominant term in the bound. The distance spectrum of TURBO codes is the result of a process called "spectral thinning"

in which the interleaver effectively moves many lower weight codewords to higher weight codewords. This theory is supported by simulation results and actual calculations of the distance spectrum of some simple TURBO codes [7].

In Figure 5, simulation results are shown along with the contributions to the union bound of the first five terms of the distance spectrum of the TURBO code. (The distance spectrum of the TURBO was calculated assuming only information sequences of weight two.) Note that the free distance asymptote remains the dominant term in the bound even for low SNR's. Thus, the excellent performance of TURBO codes at low SNR's is a result of their relatively thin distance spectrum which enables the code to follow the free distance asymptote for low and moderate SNR's.

# 5 Primitive TURBO Codes

For sufficiently large interleavers, it can be shown that the first several terms of the distance spectrum are determined solely by information sequences of weight 2 [7]. As the size of the interleaver increases, the number of terms in the distance spectrum determined solely by weight 2 information sequences increases. Consequently, for a given interleaver size, the performance of TURBO codes can be improved by choosing the feedback polynomial in the encoder such that weight 2 information sequences generate high weight codewords.

Using primitive polynomials as the feedback polynomial in the encoder causes weight 2 information sequences to generate high weight codewords, thereby increasing the free distance of the TURBO code for larger interleaver sizes and improving the performance at moderate to high SNR's. Figure 6 shows simulation results for two TURBO codes with an interleaver size of 400 and memory $\nu = 4$ encoders. As expected, the TURBO code with the primitive feedback polynomial performs better at moderate and high SNR's due to the increased free distance.

# 6 Conclusion

The excellent performance of TURBO codes may be explained in terms of the distance spectrum of the code. The 'error floor' observed in simulations of TURBO codes is a manifestation of the free distance asymptote. Since TURBO codes have relatively low free distances, the free distance asymptote dominates performance at moderate and low error rates. The 'error floor' may be lowered by increasing the size of the interleaver. The exceptional performance of TURBO codes at low SNR's is due to 'spectral thinning' and the resultant ability of the code to follow the free distance asymptote at moderate and low SNR's. Finally, for a fixed interleaver size, the performance can be improved by using primitive feedback polynomials in the encoder. This increases the free distance of the overall code and results in better performance at moderate to high SNR's.

# References


[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "New Shannon limit error-correcting coding and decoding: Turbo-codes", Proc. 1993 IEEE Int. Conf. on Comm., Geneva, Switzerland, pp. 1064-1070, 1993.

[2] S. Benedetto and G. Montorsi, "Perfornace evaluation of TURBO-codes." *IEEE Electronics Letters*, **Vol. 31**, No. 3, pg. 163, February 2, 1995.

[3] S. Benedetto and G. Montorsi, "Average performance of parallel concatenated block codes", *Electronics Letters*, **Vol. 31**, No. 3, pg. 156, February 2, 1995.

[4] S. Benedetto and G. Montorsi, *Unveiling TURBO-codes: some results on parallel concatenated coding schemes*, Dipartimento di Elettronica, Politecnico di Torine, January 17, 1995.

[5] J. D. Andersen, *The TURBO Coding Scheme*, Report IT-146, Technical University of Denmark, June 1994.

[6] P. Robertson, "Illuminating the Structure of Parallel Concatenated Recursive Systematic (TURBO) Codes", Proc. GLOBECOM '94, Vol. 3, pp. 1298-1303, San Francisco, California, November 1994.

[7] J. Seghers, *On the Free Distance of TURBO Codes and Related Product Codes*, Final Report, Diploma Project SS 1995, Number 6613, Swiss Federal Institute of Technology, Zurich, Switzerland, August 1995.

[8] J. Seghers, L. C. Perez, and D. J. Costello, Jr., "On Selecting Code Generators for Turbo Codes", *Proc. Allerton Conf. on Communication, Control, and Computing*, Monticello, IL, October 1995.

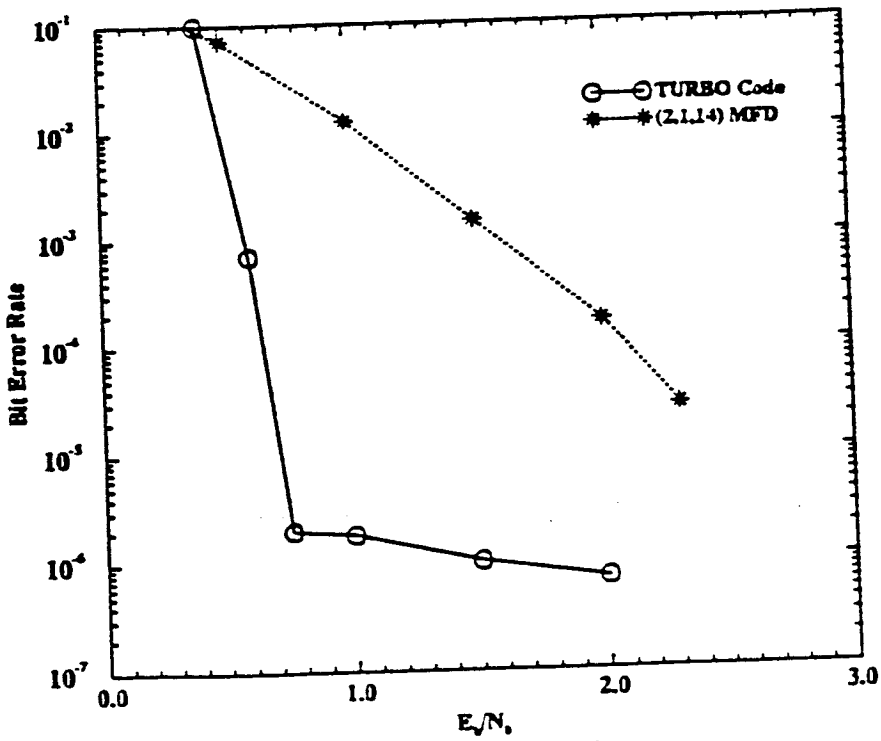



Figure 1: Simulation results for the TURBO code and a (2,1,14) convolutional code.

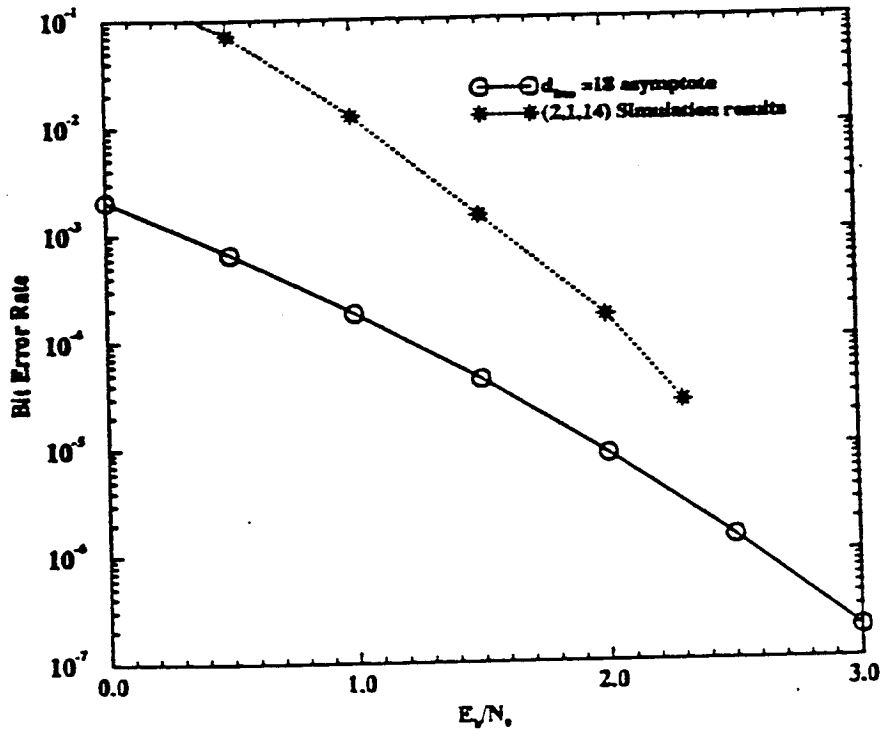



Figure 2: Asymptotic performance of the $(2, 1, 14)$ code.

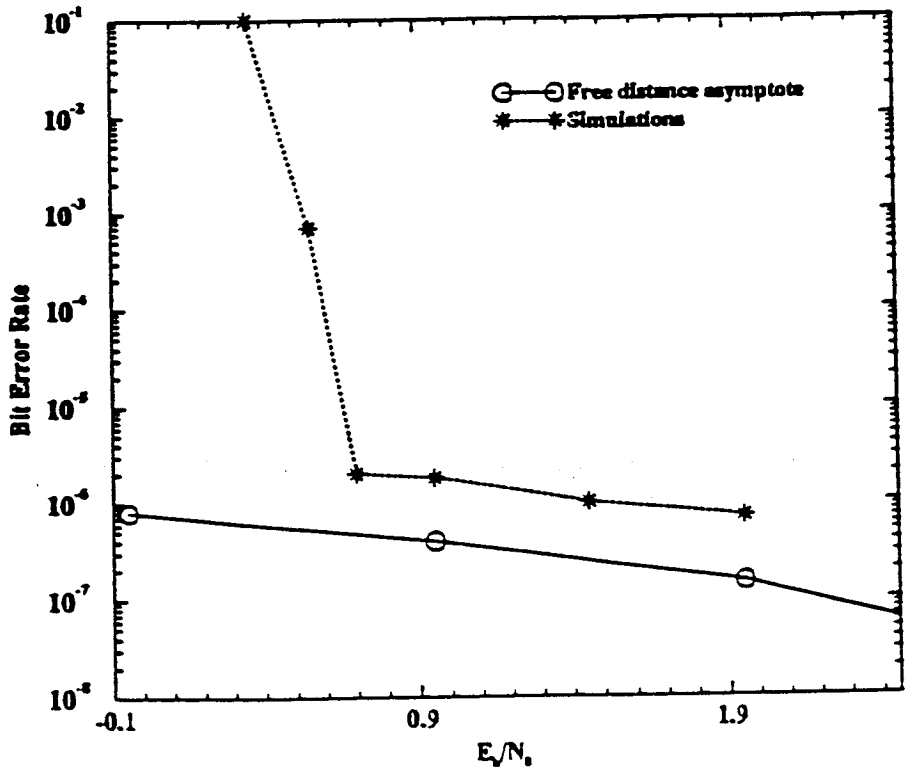



Figure 3: Asymptotic performance of the TURBO code.

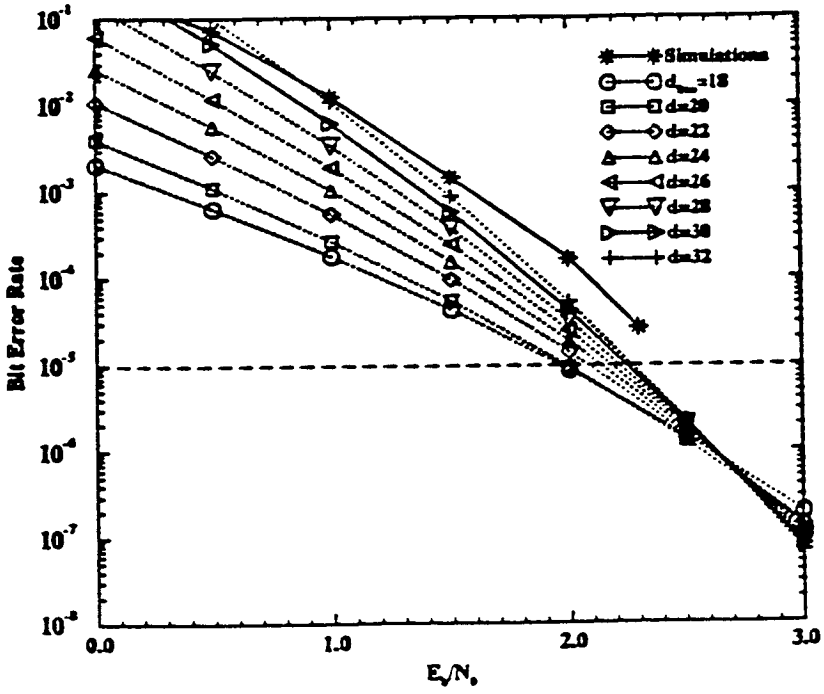



Figure 4: Decomposed performance of the MFD $(2, 1, 14)$ code.

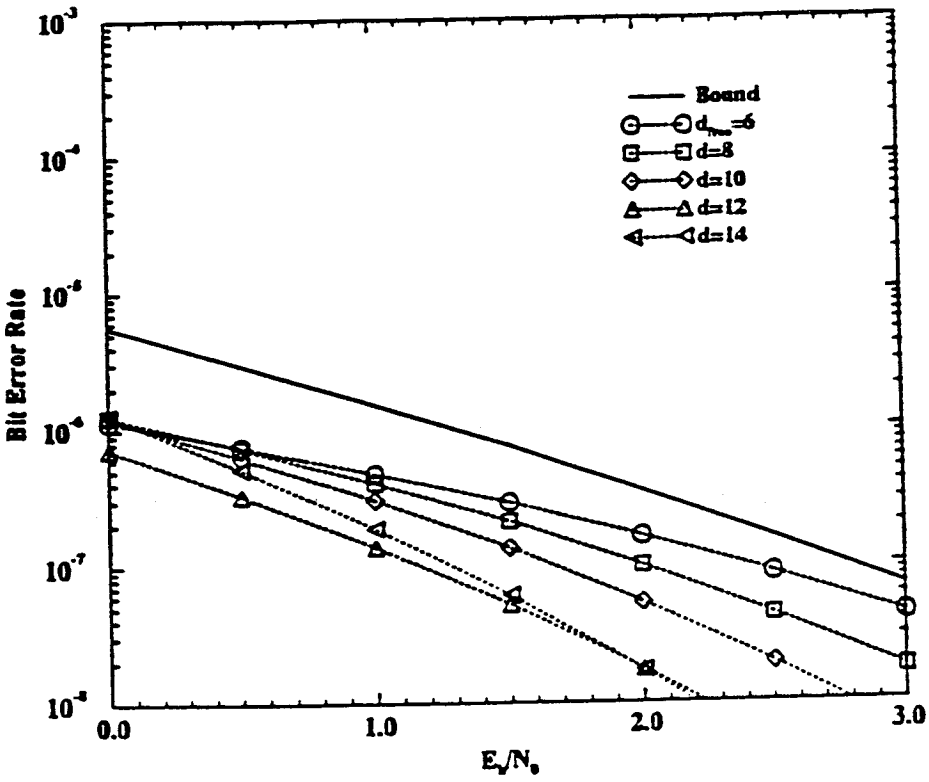



Figure 5: Decomposed performance of the TURBO code.

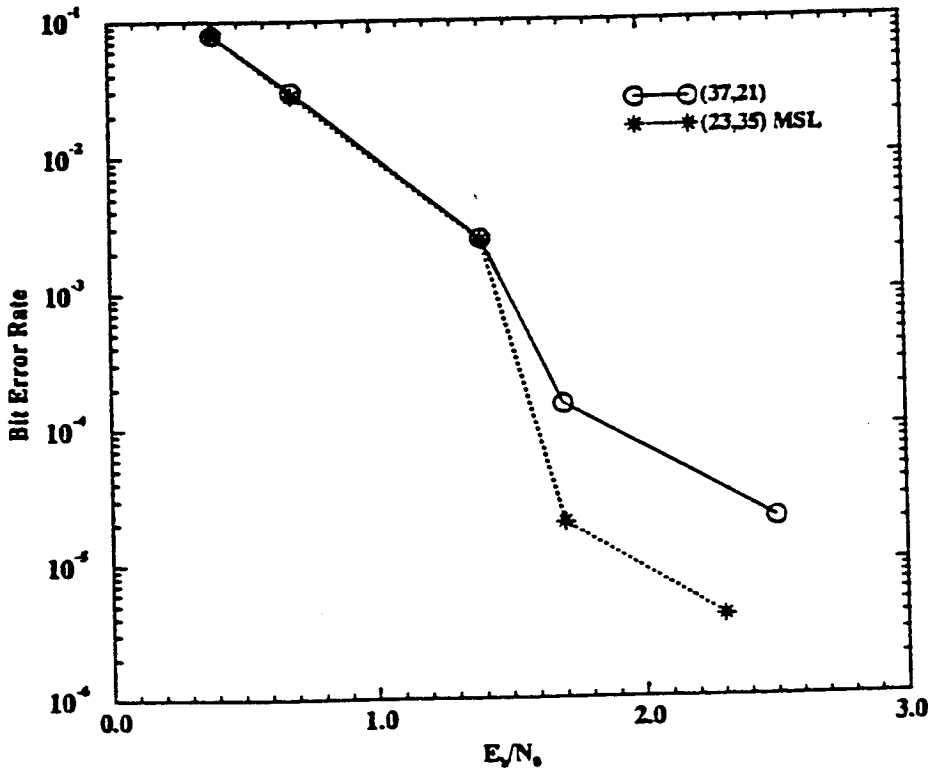



Figure 6: Performance of two TURBO codes with different feedback polynomials.

# Appendix A

## The Realization of the Turbo-Coding System

ETH Eidgenössische Technische Hochschule Zürich
Ecole polytechnique federale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut fur Signal- und Informationsverarbeitung

Semester Project

# The Realization of the Turbo-Coding System

Dieter Arnold and Guido Meyerhans
July 14th, 1995

Professors:
Prof. Dr. J. L. Massey
Prof. Dr. D. Costello

Advisors:
Dr. Lance C. Perez
Beat Keusch
Jossy Sayir

# Table of contents

# Zusammenfassung

1993 präsentierte eine französische Forschergruppe ein neues Codierungsverfahren, genannt Turbo Code. Dieses erlaubt zuverlässige Datenkommunikation auch für Signal-/Rauschverhältnisse sehr nahe bei der Shannon Limit. Dieser Bericht erklärt zuerst die Struktur des Turbo Code. Später werden Wege aufgezeigt, welche die Leistungsfähigkeit des Codes vergrössern. Dieses wird erreicht durch Änderungen an den Encodern und Verbesserungen am Interleaver. Zum Schluss wird noch die Klasse von Interleavern definiert, welche es erlaubt, beide Encoder des Turbo Code Encoders im gleichen Zustand enden zu lassen.

# Abstract

In 1993 a coding scheme called Turbo Coding was proposed by a French research group. It allows almost reliable data communication close to the Shannon limit. The structure of the Turbo Code is explained. Methods that achieve better performance through modifications of the encoders and the interleaver are discussed. Finally, the class of interleavers which allows both encoders of the Turbo Code to end in the same state will be defined.

Zurich, July 14th 1995

Dieter Arnold

Guido Meyerhans

# 1 Introduction

## 1.1 Overview

Chapter 1 is dedicated to an overview of the report and the problems addressed in this project. After this introductionary chapter, a description of the Turbo Coding System (TCS) as presented in [1] and its mathematical background are given in Chapter 2.

Chapter 3 is dedicated to convolutional codes. Some general aspects of recursive convolutional encoders and their structure will lead directly to some rules for finding improved encoders for the TCS. Bounds for these improved encoders, which result in better performance than the encoder used in [1], are given at the end of the chapter.

Interleavers influence the performance of a TCS in two ways. First, it is not possible to terminate both component encoders of the TCS at the same time with a typical interleaver. In order to terminate both encoders, interleavers have to be constructed according to special rules. Second, in order to achieve high performance the ability to move the elements in the interleaver has to be maximized. In chapter 4, it will be shown how compromises can be achieved for these contradictory demands.

Simulations were carried out with three different kinds of quantization. The performance of the TCS with these different quantization schemes is explained in chapter 5.

In chapter 6, improvements on the TCS are presented. These improvements follow directly from the knowledge obtained from the three previous chapters.

The report ends with chapter 7, where general comments and concluding remarks are made.

For a detailed presentation of the simulation results the interested reader is asked to look at appendix A. Simulation results are presented in appendix B.

## 1.2 Tasks

The following pages contain a copy of the tasks given at the beginning of this semester project.

# 2 Introduction to Turbo Codes

The aim of this chapter is to give an introduction to Turbo Codes and a basic explanation of their performance. To this end, we make use of the Turbo Coding System (TCS) presented in [1]. Throughout this chapter we follow closely the paper of P. Robertson [2] to clarify many questions that arose in the original Turbo Code paper [1].

## *2.1 The MAP algorithm*

In this section, a complete derivation of the Bahl, et al., maximum-a-posteriori (MAP) algorithm is given. This derivation is important for two reasons.
First, the MAP algorithm has to be modified to be used with the systematic feedback encoders of the Turbo Coding System. Second, the modified MAP algorithm presented in [1] is unnecessarily complicated. A simpler version was presented in [6], which issued throughout the literature on Turbo Codes.

### 2.1.1 Why and how the MAP algorithm has to be modified for recursive encoders

The MAP Algorithm is derived in its most general form in [5]. Where ist application to nonrecursive encoders was also discussed. Nonrecursive codes are characterized by the following features:

1) The input is equal to the first element of the state S at time k, that is
$$d_k = S_0(k) \qquad \text{where } S_k = (S_0(k), S_1(k), \ldots, S_{M-1}(k)).$$
2) To transfer from any state at time k-1 to a particular state $S_k = m$ at time k it is required that the input bits become all zeroes or all ones. In other words, the states can be divided into two subsets according to the inputs required for a transition from time k-1 to time k (Figure 2.1)

Recursive encoders do not have these features. Since the state at time k, $S_k$, is defined by the input at time k, $d_k$, and the state at time k-1, $S_{k-1}$, it is possible to reach state $S_k$ with a one or a zero input depending on the previous state $S_{k-1}$. As a consequence of this, the states of a systematic feedback encoder cannot be divided into subsets based on the input (Figure 2.1). As a further consequence, the *a posteriori* probability cannot be calculated as

$$P(d_k = 0 | R_1^N) = \frac{\sum_{S_k \in A_k^0} P(S_k = m, R_1^N)}{P(S_N = 0, R_1^N)} \quad (A_k^0 = \text{subset of all states with } S_0(k) = d_k) \qquad (2.1)$$

In addition the decoding rule

$$d_k = \begin{cases} 0 \text{ if } P(d_k + 0 | R_1^N) \geq 0.5 \\ 1 \text{ otherwise} \end{cases} \qquad (2.2)$$

cannot be applied as in [5], where $R_1^N$ denotes the received bits from time 1 to N (= end of transmission).

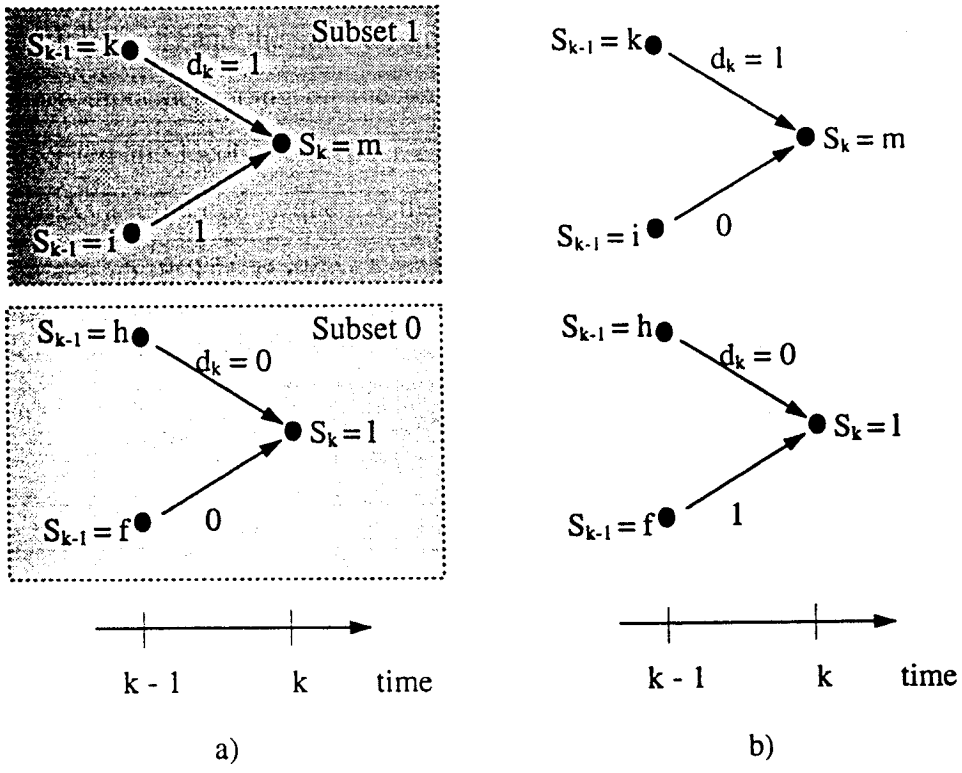



Figure 2.1: Example of a trellis of a non-recursive code (a) and a recursive code (b)

Instead, for recursive encoders the logarithm of the ratio of the *a posteriori* probability (APP) of each information bit being a 1 to the APP of it being a 0 is calculated. That is,

$$\Lambda_k = \Lambda(d_k) = \log \frac{P(d_k = 1 | R_1^N)}{P(d_k = 0 | R_1^N)} \tag{2.3}$$

is calculated and the decoding rule

$$d_k = \begin{cases} 1, \Lambda_k > 0 \\ 0, \Lambda_k \leq 0 \end{cases} \tag{2.4}$$

### 2.1.2 Derivation of the extrinsic information

The expression for $\Lambda_k$, given in [5] as

$$\Lambda(d_k) = \log \frac{P(d_k = 1 | R_1^N)}{P(d_k = 0 | R_1^N)} = \log \frac{\sum_m \sum_{m'} \gamma_1(R_k, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)}{\sum_m \sum_{m'} \gamma_0(R_k, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)} \tag{2.5}$$

where $d_k$ denotes the data input into the encoder at time k, will now be derived.

First, define

$$\alpha_k(m) = P(S_k = m | R_1^k) = P(S_k = m, R_1^k) \cdot \frac{1}{P(R_1^k)} \tag{2.6}$$

$$\beta_k(m) = P(R_{k+1}^N | S_k = m) \cdot \frac{1}{P(R_{k+1}^N | R_1^k)} \tag{2.7}$$

$$\gamma_i(R_k, m, m') = P(d_k = i, R_k, S_k = m, | S_{k-1} = m') \tag{2.8}$$

where state m at time k is denoted by $S_k = m$.

Recall Bayes rule,

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

and the expressions for the conditional probability,

$$P(B|A) = \frac{P(A, B)}{P(A)}$$

Combining these two equalities we obtain the following well known relation

$$P(A, B) = P(A|B) \cdot P(B) \tag{2.9}$$

which turns out to be very useful for the following derivation.

**Proof:**

In order to prove (2.5) one can simply prove

$$P(d_k = i|R_1^N) = \sum_m \sum_{m'} \gamma_i(R_k, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m) \qquad (2.10)$$

To do so, the left side is first expanded over all states m = 0...M-1 and then (2.9) is applied. This results in

$$P(d_k = i|R_1^N) = \sum_m P(d_k = i, S_k = m|R_1^N) = \sum_m P(d_k = i, S_k = m, R_1^N) \cdot \frac{1}{P(R_1^N)}$$

Since the observations are independent of time, this can be rewritten as

$$\sum_m P(d_k = i, S_k = m, R_1^k, R_{k+1}^N) \cdot \frac{1}{P(R_1^k, R_{k+1}^N)}$$

Applying (2.9) to the numerator and the denominator gives

$$\frac{\sum_m P(R_{k+1}^N|d_k = i, S_k = m, R_1^k) \cdot P(d_k = i, S_k = m, R_1^k)}{P(R_{k+1}^N|R_1^k) \cdot P(R_1^k)}.$$

Using the property that events after time k are not influenced by the observation of $R_1^k$ and bit $d_k$ if the state $S_k$ is known (i.e. Markov property of the source), this can be simplified to

$$\sum_m \underbrace{\frac{P(R_{k+1}^N|S_k = m)}{P(R_{k+1}^N|R_1^k)}}_{\beta_k(m)} \cdot \frac{P(d_k = i, S_k = m, R_1^k)}{P(R_1^k)}, \qquad (2.11)$$

where the first term is recognized as $\beta_k(m)$.

The second term can be rewritten as

$$\frac{P(d_k = i, S_k = m, R_1^k)}{P(R_1^k)} = \frac{P(d_k = i, S_k = m, R_1^{k-1}, R_k)}{P(R_1^{k-1}, R_k)} =$$

$$= \frac{P(d_k = i, S_k = m, R_k|R_1^{k-1})}{P(R_k|R_1^{k-1})} \cdot \frac{P(R_1^{k-1})}{P(R_1^{k-1})}. \qquad (2.12)$$

The first equality comes from the time independence of the observation and the second from (2.9). The second term can now be canceled. Notice that expanding the denominator over all possible states m = 0...M-1 at time k-1 and all possible inputs i = 0, gives

$$P(R_k|R_1^{k-1}) = \sum_m \sum_i P(d_k = i, S_k = m, R_k|R_1^{k-1}). \qquad (2.13)$$

So the proof continues simply with the numerator which can be expanded over all states m'= 0...M-1 at time k-1 as follows,

$$P\left(d_k = i, S_k = m, R_k \middle| R_1^{k-1}\right) = \sum_{m'} P\left(d_k = i, S_k = m, S_{k-1} = m', R_k \middle| R_1^{k-1}\right) \qquad (2.14)$$

Again using the time independence of the observation results in

$$\sum_{m'} P\left(d_k = i, S_k = m, S_{k-1} = m', R_k, R_1^{k-1}\right) \cdot \frac{1}{P\left(R_1^{k-1}\right)}$$

and with (2.9) this becomes

$$= \sum_{m'} P\left(d_k = i, S_k = m, R_k \middle| S_{k-1} = m', R_1^{k-1}\right) \cdot \underbrace{\frac{P\left(S_{k-1} = m', R_1^{k-1}\right)}{P\left(R_1^{k-1}\right)}}_{\alpha_{k-1}(m')} =$$

$$= \sum_{m'} P\left(d_k = i, S_k = m, R_k = m \middle| S_{k-1} = m', R_1^{k-1}\right) \cdot \alpha_{k-1}(m') =$$

Now the substitution of the second term with $\alpha_{k-1}(m')$ and the first term with $\gamma_k(Rk,m,m')$ gives

$$= \sum_{m'} \underbrace{P(d_k = i, S_k = m, R_k = m \middle| S_{k-1} = m')}_{\gamma_i(R_k,m,m')} \cdot \alpha_{k-1}(m') =$$

$$= \sum_{m'} \gamma_i\left(R_k, m, m'\right) \cdot \alpha_{k-1}(m').$$

Equation (2.11) can now be expressed with (2.13) as

$$\frac{P\left(d_k = i, S_k = m, R_k \middle| R_1^{k-1}\right)}{P\left(R_k \middle| R_1^{k-1}\right)} = \frac{\sum_{m'} \gamma_i(R_k, m, m')\alpha_{k-1}(m')}{\sum_{m}\sum_{i}\sum_{m'} \gamma_i(R_k, m, m')\alpha_{k-1}(m')}.$$

Interchanging the sums over i and m' is allowed since $\alpha_{k-1}(m)$ does not depend on i, yielding

$$= \frac{\sum_{m'} \gamma_i(R_k, m, m')\alpha_{k-1}(m')}{\sum_{m}\sum_{m'}\sum_{i} \gamma_i(R_k, m, m')\alpha_{k-1}(m')} \qquad (2.15)$$

Combing (2.11) with (2.10) gives

$$P\left(d_k = i \middle| R_1^N\right) = \frac{\sum_{m}\sum_{m'} \gamma_i(R_k, m, m')\alpha_{k-1}(m')\beta_k(m)}{\sum_{m}\sum_{m'}\sum_{i} \gamma_i(R_k, m, m')\alpha_{k-1}(m')} \qquad (2.16)$$

Finally, setting i =0 and i =1 and dividing gives

$$\Lambda(d_k) = \log \frac{P\left(d_k = 1 \middle| R_1^N\right)}{P\left(d_k = 0 \middle| R_1^N\right)} = \log \frac{\sum_{m}\sum_{m'} \gamma_1(R_k, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)}{\sum_{m}\sum_{m'} \gamma_0(R_k, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)} \qquad (2.17)$$

since the denominator of (2.16) is in both cases equal. ❑

**Difference with the original Turbo Code paper [1]**

In the original Turbo Code paper [1], equation (2.14) was also expanded over all possible previous data inputs $d_{k-1} = 0,1$. This leads to $\alpha_k^i(m)$ depending also on the input $d_{k-1} = i$. This is unnecessary, as shown in [7], since $\alpha_k^0(m) + \alpha_k^1(m)$ is equal to the $\alpha_k(m)$ defined in (2.6).

### 2.1.3 Recursive calculation of $\alpha_k(m)$ and $\beta_k(m)$.

The recursive calculation of $\alpha_k(m)$ and $\beta_k(m)$ is now derived.

$\alpha_k(m)$ was defined in (2.6) as

$$\alpha_k(m) = P\left(S_k = m \middle| R_1^k\right) = P\left(S_k = m, R_1^k\right) \cdot \frac{1}{P\left(R_1^k\right)}.$$

The second term of (2.11) can be obtained by expanding $\alpha_k(m)$ as defined in (2.6) over the possible inputs

$$\alpha_k(m) = \sum_i \frac{P\left(d_k = i, S_k = m, R_1^k\right)}{P\left(R_1^k\right)}.$$

Substituting the term in the sum with (2.15) results in a forward recursion for $\alpha_k(m)$ given by

$$\alpha_k(m) = \frac{\sum_{m'} \sum_i \gamma_i(R_k, m, m') \alpha_{k-1}(m')}{\sum_m \sum_{m'} \sum_i \gamma_i(R_k, m, m') \alpha_{k-1}(m')} \qquad (2.18)$$

with the initial conditions $\alpha_0(0) = 1$ and $\alpha_0(k) = \alpha_n(k) = 0, \forall\, k \neq 0$ and $\forall\, n \neq 0$.

Expanding $\beta_k(m)$ as defined in (3.2) over all possible states m = 0...M-1 and all possible inputs i = 0,1 at time k+1 gives

$$\beta_k(m) = \frac{\sum_{m'}\sum_{i} P\left(d_{k+1} = i, S_{k+1} = m', R_{k+2}^N, R_{k+1} \middle| S_k = m\right)}{P\left(R_{k+1}^N \middle| R_1^k\right)}$$

where again the time independence of the observations was used.

Using Bayes's rule, the numerator becomes

$$\sum_{m'}\sum_{i} P\left(R_{k+2}^N \middle| S_{k+1} = m'\right) \cdot \underbrace{P\left(d_{k+1} = i, S_{k+1} = m', R_{k+1} \middle| S_k = m\right)}_{\gamma_i\left(R_{k+1}^N, m, m'\right)}$$

The backward recursion for $\beta_k(m)$ is then given by

$$\beta_k(m) = \frac{\sum_{m'}\sum_{i=0}^{1} \gamma_i(R_{k+1}, m, m') \cdot \beta_{k+1}(m')}{\sum_{m}\sum_{m'}\sum_{i=0}^{1} \gamma_i(R_{k+1}, m, m') \cdot \alpha_k(m')} \qquad (2.19)$$

with the initial conditions $\beta_N(0) = 1$ and $\beta_N(k) = \beta_n(k) = 0$ ,$\forall\ k \neq 0$ and $\forall\ n \neq 0$.

### 2.1.4 The additive character of the MAP algorithm

Looking at (2.17), (2.18) and (2.19) it is clear that the transition probability $\gamma_i$ is the most important term.

Equation (2.9) can be rewritten as

$$\gamma_i(R_k, m', m) = p\left(R_k \middle| d_k = i, S_k = m, S_{k-1} = m'\right) \cdot P\left(d_k = i \middle| S_k = m, S_{k-1} = m'\right) \cdot P\left(S_k = m \middle| S_{k-1} = m'\right)$$

As will be show in section 2.2.3, $R_k$ consists of $y_k^s$, the systematic information sequence, and $y_k^p$, the parity sequence[1], which are independent of each other. The first term can therefore be written as a product of the two received bits. That is,

$$p\left(R_k \middle| d_k = i, S_k = m, S_{k-1} = m'\right) = p\left(y_k^s \middle| d_k = i, S_k = m, S_{k-1} = m'\right) \cdot p\left(y_k^p \middle| d_k = i, S_k = m, S_{k-1} = m'\right)$$

---

[1] In 2.2.1 we shall see that $x_k^p$ consists itself of two parity sequences.

where $p(\dashv) = \frac{1}{2\sqrt{\sigma}} e^{\frac{-(y_k^{s/p} - om_k^{s/p}(i,m',m))^2}{2\sigma^2}}$ is the Gaussian distribution (we consider here the channel to have additive white Gaussian noise, see 2.2.3), and

$$om_k^{s/p}(i,m',m)$$

is the modulator output (see 2.2.1.d).

Since $y_k^s$ is independent of the states m, it can be taken out of the logarithm directly. Defining the *a priori* probability

$$P(S_k = m | S_{k-1} = m') = \frac{e^{\bar{\Lambda}(d_k)}}{1 + e^{\bar{\Lambda}(d_k)}} \text{ when } P(d_k = 1 | S_k = m, S_{k-1} = m') = 1 \text{ and}$$

$$P(S_k = m | S_{k-1} = m') = 1 - \frac{e^{\bar{\Lambda}(d_k)}}{1 + e^{\bar{\Lambda}(d_k)}} \text{ when } P(d_k = 0 | S_k = m, S_{k-1} = m') = 1$$

leads to

$$\Lambda(d_k) = \underbrace{\frac{1}{2 \cdot \sigma^2} \cdot y_k^s}_{K_c \cdot y_k^s} + \bar{\Lambda}(d_k) + \underbrace{\log \frac{\sum_m \sum_{m'} \gamma_1^e(y_k^p, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)}{\sum_m \sum_{m'} \gamma_0^e(y_k^p, m', m) \cdot \alpha_{k-1}(m') \cdot \beta_k(m)}}_{\Lambda_e(d_k)}$$

(2.20)

with

$$\gamma_i^e(y_k^p, m', m) = p(y_k^p | d_k = i, S_k = m, S_{k-1} = m') \cdot P(d_k = i | S_k = m, S_{k-1} = m').$$

The first term in (2.20) is the received channel output multiplied with a constant $K_c$ depending on the channel. The second term is the *a priori* probability of the bit k being a 1 or a 0. The third term, the so called extrinsic information, can be considered as the updated *a priori* information for a possible second decoder. Together, these terms give *the a posteriori* probability $\Lambda(d_k)$. Figure 2.2 shows the structure of a soft input/soft output decoder.

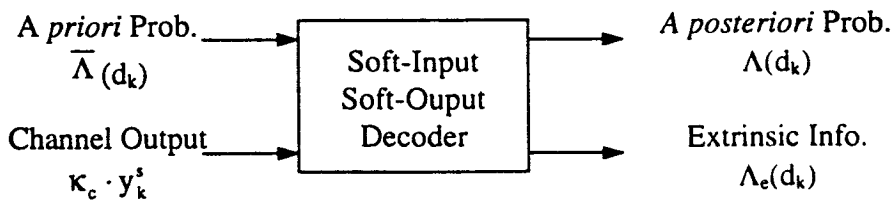



Figure 2.2: Structure of a Soft-Input/Soft-Output decoder

## 2.2 The Turbo-Coding System

In this section, we discuss the three main parts into which a communication system can be divided: Namely the transmitter, the channel and the receiver as shown in Figure 2.3.

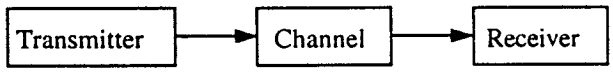



Figure 2.3 - The three main parts of the Turbo-Coding system

### 2.2.1 Transmitter

The transmitter itself can be divided into five parts: the source, the encoder, the puncturer and the modulator as shown in Figure 2.4


Figure 2.4: Transmitter

#### a. Source

The source produces the message to be transmitted. For the simulations reported here, a source generating a random binary sequence was used. In order to avoid difficulties caused

by a poor random number generator the Data Enscritption Standard (DES) random-generator from [17] was used.

### b. Encoder

The encoder consists of an interleaver and two binary, recursive, systematic, convolutional encoders (RSC), which henceforth are referred to as component encoders 1 and 2. One therefore operates in GF(2) and all additions are EXOR operations.

A non-uniform interleaver of size 64000, which is filled up row by row and read pseudo-randomly (Fig. 2.5), is used in the TCS. Although interleavers are well known, they are of central importance in a TCS. Chapter 4 is dedicated to a more detailed presentation of interleavers and their influence in a TCS.

Input

[1, 2, 3,..., 12] →

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |

Output

→ [7, 2, 3, 11, 1, 12, 4, 5, 6, 9, 8, 10]

Figure 2.5: Nonuniform interleaver

As proposed in [1], the component encoders are identical, although this is not necessary in general. These component encoders are defined by their rate R, the number of states ($M=2^m$) determines the constraint length $n_A = (m + 1)$ and visa-versa, a feedback generator $G_1$ and a feedforward generator $G_2$. The first component encoder operates directly on the information to be transmitted, while the second component encoder is fed with the interleaved information sequence. The output of the first component encoder consists of the systematic sequence and the parity sequence, while the output of the second component encoder consists of the parity sequence only. The systematic sequence of the second component encoder is not transmitted (Figure 2.6).

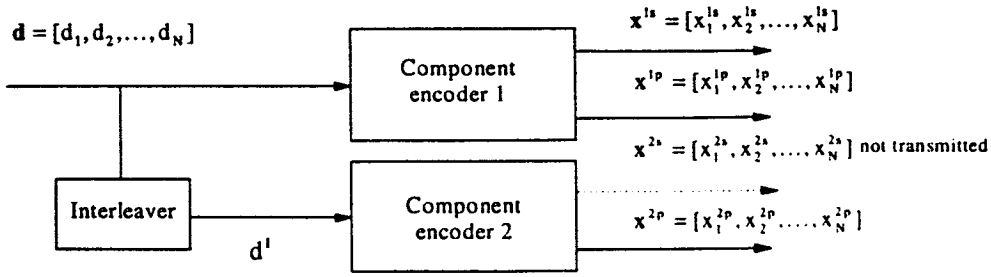



Figure 2.6: Encoder

As mentioned before, the same encoders and interleaver as in [1] have been implemented. The two component encoders have constraint length $n_A = 5$, generators $G_1 = 37$ and $G_2 = 21$ and are used in a parallel concatenation (Figure 2.7).

The data sequence $\mathbf{d}$ is the input. The first output sequence $\mathbf{x}^{1s}$ is equal to the input sequence because of the systematic structure of the first component encoder. The second output sequence $\mathbf{x}^{1p}$ is the parity sequence from the first component encoder. This is given by

$$\mathbf{x}^{1p} = [x_1^{1p}, x_2^{1p}, \ldots, x_k^{1p}, \ldots, x_N^{1p}] \text{ with } x_k^{1p} = \sum_{i=1}^{M} g_i^f \cdot a_{k-i}^1 \text{ and } a_k^1 = d_k + \sum_{i=1}^{M} g_i^b \cdot a_{k-i}^1$$

where $G_2=(g_1^f, g_2^f, g_3^f, g_4^f) = (1,0,0,1)$ is the feedforward generator and $G_1=(g_1^b, g_2^b, g_3^b, g_4^b) = (1,1,1,1)$ is the feedback generator.

The third output sequence $\mathbf{x}^{2p}$ is the second parity sequence, generated by the second component encoder. The parity sequence $\mathbf{x}^{2p}$ is obtained in the same way as $\mathbf{x}^{1p}$ with the difference that the data $\mathbf{d}^I$ on which the second component encoder operates is interleaved. Thus,

$$\mathbf{x}^{2p} = [x_1^{2p}, x_2^{2p}, \ldots, x_k^{2p}, \ldots, x_N^{2p}] \text{ with } x_k^{2p} = \sum_{i=1}^{M} g_i^f \cdot a_{k-i}^2 \text{ and } a_k^2 = d_k^I + \sum_{i=1}^{M} g_i^b \cdot a_{k-i}^2 \ ,$$

where again $(g_1^f, g_2^f, g_3^f, g_4^f) = (1,0,0,1)$, $(g_1^b, g_2^b, g_3^b, g_4^b) = (1,1,1,1)$.

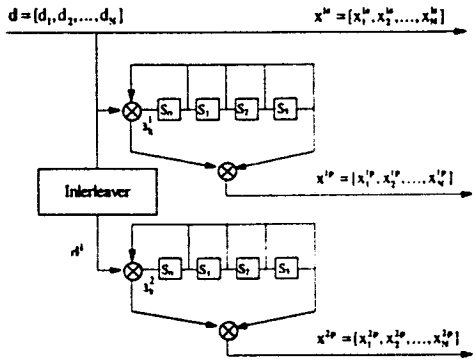



Figure 2.7: Encoder

The encoder produces three bits, $x_k^{1s}, x_k^{1p}$ and $x_k^{2p}$, for each input bit, and thus the rate of the encoder is 1/3.

Both component encoders are assumed to start in the all zero state: i.e. $a_l^1$ and $a_l^2$ are zero.

Due to the interleaver the termination of both component encoders at the same time turns out to be difficult. At this point, only the first component encoder is terminated and the second is left 'open'. As a consequence, the initial condition of $\beta_N(k)$ in the second, non-terminating trellis has to be changed from (2.19) to

$$\beta_N(k) = \alpha_N(k), \forall\ k = 0...M-1. \tag{2.21}$$

A solution to this problem was not given in [1]. However, it will be shown in chapter 4 that terminating both trellises is possible for certain classes of interleavers.

## c. Puncturer

In order to achieve a higher global rate than R = 1/3 the parity sequences $\mathbf{x}^{1p}$ and $\mathbf{x}^{2p}$ are punctured according to a puncturing matrix **P**. For a global rate of R = 1/2, **P** is chosen as

$$\mathbf{P} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.22}$$

and the punctured 'parity bit' sequence is

$$\mathbf{x}^{pp} = [x_1^{1p}, x_2^{2p}, \ldots, x_{N-1}^{1p}, x_N^{2p}].$$

This means that $x_{i.}^{1p}$ is sent at every odd time and $x_{i.}^{2p}$ at every even time k.

## d. Modulator

The modulator assigns to every bit of the incoming sequences $\mathbf{x}^{pp}$, $\mathbf{x}^{1s}$ a value corresponding to its level. We chose the same modulator as in [1] which assigns a +1 in case of a 1 and -1 in case of a 0 according to (2.22)

$$\begin{aligned} xm_k^{pp} &= 2 \cdot x_k^{pp} - 1 \\ xm_k^{1s} &= 2 \cdot x_k^{1s} - 1 \end{aligned} \tag{2.23}$$

where $xm_k^{pp}$ and $xm_k^{1s}$ are the outputs of the modulator at time k.

Figure 2.8 gives a detailed overview over the transmitter

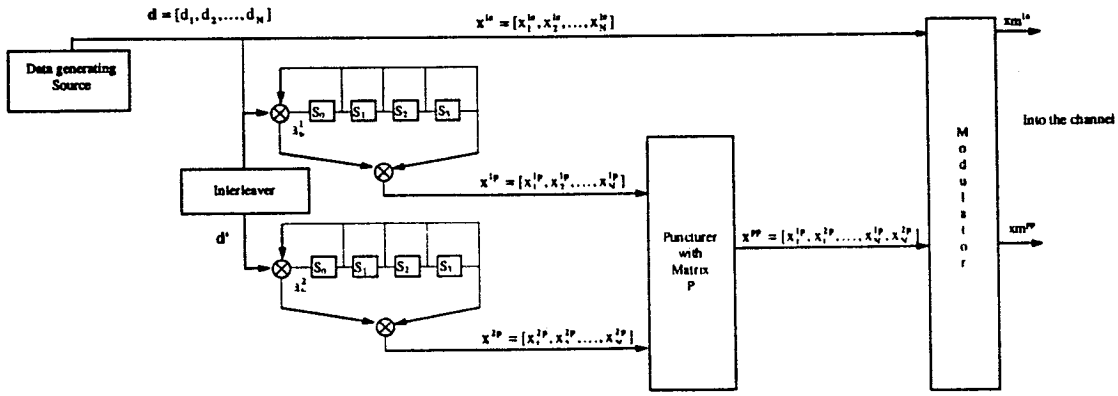



Figure 2.8: Detail view of the transmitter

### 2.1.1 Channel

The channel for $\mathbf{x}^{1s}$ and $\mathbf{x}^{pp}$ is the additive white Gaussian noise (AWGN) channels where an independent noise sample $n^{1,2}$ with zero mean and variance $\sigma_{1,2}^2 = No_{1,2} / 2$ is added to each transmitted bit. The channel inputs are denoted by $\mathbf{x}$ and the channel outputs by $\mathbf{y}$ (Figure 2.9).

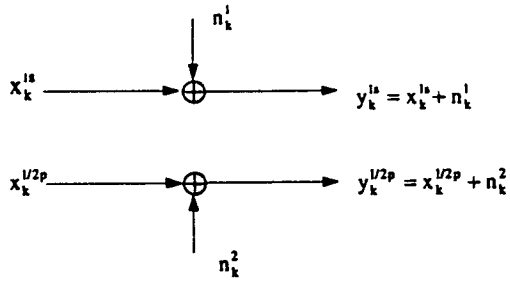


Figure 2.9: Channel model

### 2.1.2 Receiver

The receiver provides the Turbo Code with its name and its performance. Analogous to the transmitter, the receiver can be divided into a depuncturer, a decoder and a sink. The demodulator is part of the decoder (Figure 2.10).


Figure 2.10: Receiver

#### a. Depuncturer

The depuncturer separates the (demultiplexed) parity bit sequence into the two parity bit sequences $\mathbf{y}^{1p}$ and $\mathbf{y}^{2p}$ and substitutes a zero for the punctured bits (Figure 2.11).

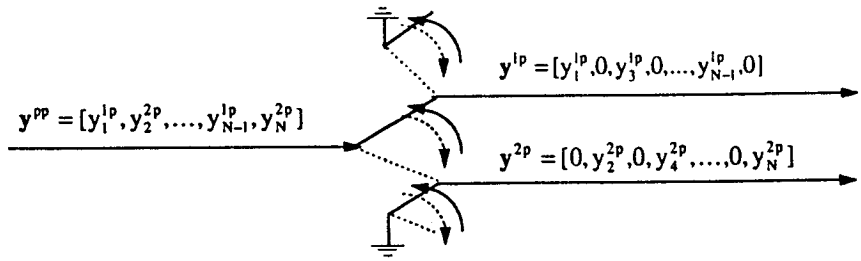



Figure 2.11: Depuncturer

#### b. Decoder

The decoder consists of two deinterleavers, two interleavers, a final harddecision decoder and two softinput/softoutput decoders which are denoted as component decoder 1 and 2. Both component decoders use the modified MAP algorithm.

The first component decoder is fed with the *a priori* probability $\Lambda^{1in}$, the information sequence $\mathbf{y}^{1s}$ and the parity sequence $\mathbf{y}^{1p}$. The result of the first component decoder is $\Lambda^{1out}$, the extrinsic information which is independent of the *a priori* information and the systematic information for the corresponding bit $d_k$. At the output of the first decoder, the *a posteriori* probability (APP) $\Lambda^{1out}$, is not of interest, since the decoder has not made use of $\mathbf{y}^{2p}$ which may improve the APP.

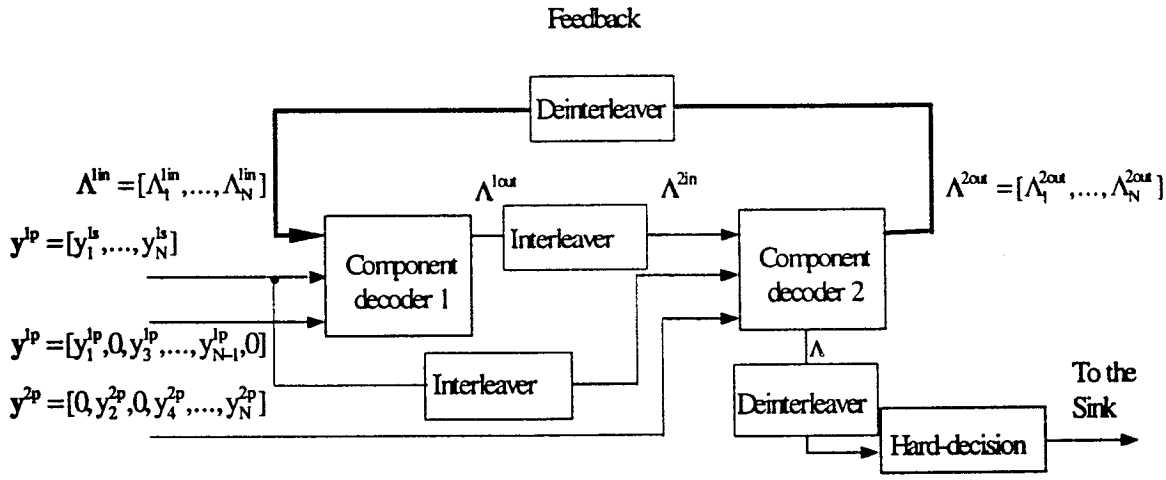



Figure 2.12: Decoder

The second component decoder is fed with $\Lambda^{2in}$ (which is equal to $\Lambda^{1out}$ deinterleaved) as the *a priori* probability, $\mathbf{x}^{1s}$ deinterleaved and $\mathbf{x}^{2p}$. The deinterleavers are necessary to ensure that $\mathbf{x}^{1s}$, $\Lambda^{2in}$ at time k corresponds to $\mathbf{x}^{2p}$ at time step k. The second component decoder produces the new extrinsic information $\Lambda^{2out}$ and the *a posteriori* probability $\Lambda$, since both parity bits are now used.

This procedure is known as iterative decoding. The two component decoders are suboptimal because the first component decoder uses only half of the available redundant information ($\mathbf{x}^{1p}$ but not $\mathbf{x}^{2p}$). As a consequence, an improvement in performance can be achieved through the use of a feedback loop ($\Lambda^{1in} = \Lambda^{2out}$ ). In the simulations up to 18 iterations were used as proposed in [1]. This means that for every information bit $d_k$ both

component decoders are used 18 times. The term *Turbo* Codes is given for this iterative decoding scheme using feedback with reference to the turbo engine principle.
After 18 iterations, the APP of the second component decoder is led to the final hard-decision decoder which decodes $\mathbf{d}^e$ according to (2.4).

### c. Sink

In the simulation program, the sink is simply represented by a procedure for counting the decoding errors.

The achieved results with the algorithm are shown in Figure 2.13. The performance is the same like presented in [1].

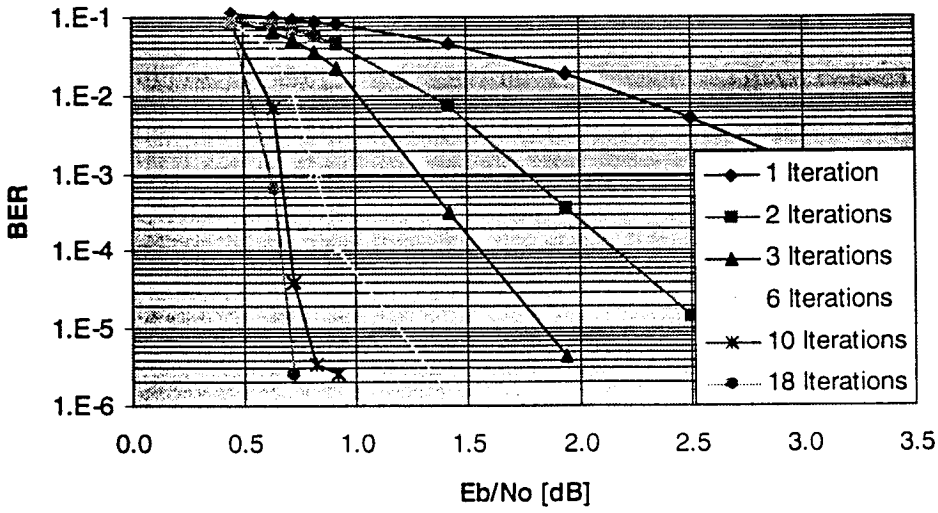



Figure 2.13: BER of Turbo Code with a 65536 bit interleaver

Some modifications on the encoder and interleavers are made in the following chapter. But the structure of the algorithm will not be changed.

# 3 Convolutional Codes

The turbo code uses a recursive encoder to produce its parity check bits. This chapter discusses the structure of the trellis used in the turbo-code, which is important to understand design considerations for interleavers.

## 3.1 General aspects of recursive convolutional codes

The turbo codes are realized with convolutional encoders in systematic feedback form. Such encoders always have a feedback and a forward path as shown in Figure 3.1. The encoder is given by its generator polynomials in octal numbers, e.g. (37,21). The first number defines the feedback and the second one the forward path.

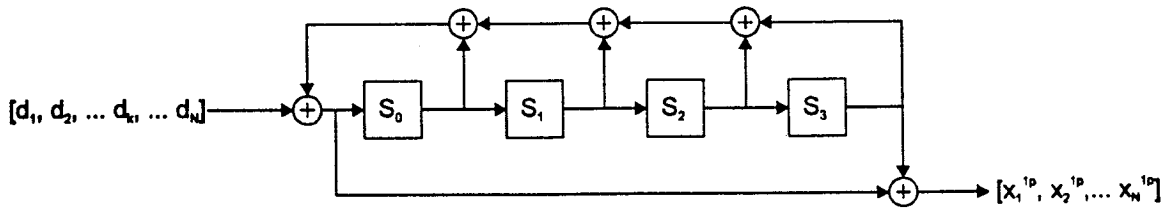



Figure 3.1: Standard turbo encoder (37,21)

The constraint length is defined as

$$n_A = m+1, \tag{3.1}$$

where the memory order m is the length of the shift register. The constraint length is not as imprtant for recursive encoders, as it is for feedforward ones. Another parameter, the so called cycle length $n_{cycle}$ is more important for recursive codes. The cycle length is best defined by considering the encoder output sequence for a *base* information sequence. A base sequence is an information bit sequence in which only one bit of the vector is equal to 1, e.g. d=[0,0,0,1,0,0,0,..,0,0]. The corresponding output sequence shown in Figure 3.2, has a repetitive structure. The length of this repetitive sequence is called the cycle length.

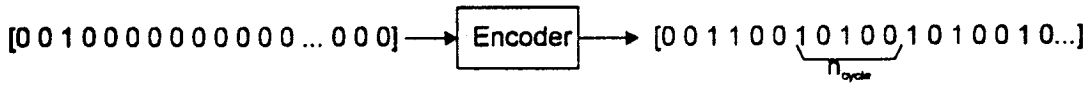



Figure 3.2: Output vector of a base sequence

The cycle length is given by the feedback path of the encoder and is bounded by the memory order by

$$m \leq n_{cycle} \leq 2^m - 1 \qquad (3.2)$$

In the case of the turbo code with generators (37,21), the cycle length is $n_{cycle} = m + 1$.

The cycle length is important in the design of the interleaver, as will be shown in chapter 3.3 and chapter 4.

## 3.2 Structure of the Turbo Code feedback shift register

The structure of the systematic feedback encoder is important in the design of good interleavers and in the achievement of better performance. As described in the previous chapter, a recursive encoder has a repetitive output sequence. A similar cyclic process exists in the content of the encoder's memory.

Figure 3.3 shows the typical structure of the turbo code encoders, where all feedback taps are equal to 1.

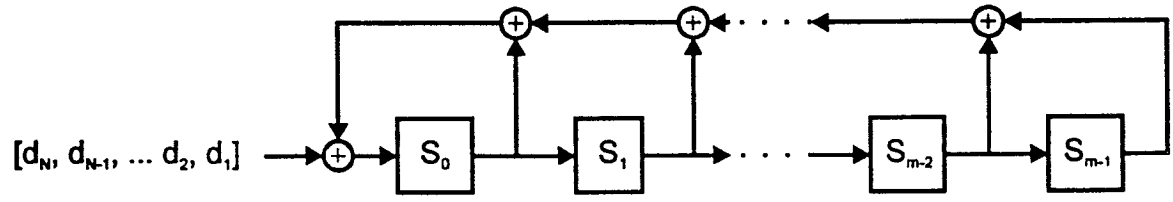



Figure 3.3: Structure of a recursive encoder, where all feedback taps are set

The content of the memory of the encoder shown in Figure 3.3 is given by

$$S_0(n) = \sum_{k=1}^{n} l_k(n) \cdot d_k$$

$$\text{with } l_k(n) = \big((n-k) \bmod (m+1) = 0\big) \oplus \big((n-k) \bmod (m+1) = 1\big) \qquad (3.3)$$

$$= \big((n-k) \bmod (m+1) = 0\big) \oplus \big((n-k-1) \bmod (m+1) = 0\big)$$

and where

$$S_{i+1}(n) = S_i(n-1) \qquad (3.4)$$

m: Memory order of encoder

$S_0(n)..S_{m-1}(n)$: Content of the memory cell at time n

dk: encoder input bit at time k

**Example:**

The information sequence for an encoder with feedback polynomial (37) is d=[0,1,1,0,1,0,0,0,0,1,1,1]. According to (3.3) and (3.4) this will lead to the following contents of the encoder memory:

| n | $d_n$ | $S_0(n)$ | $S_1(n)$ | $S_2(n)$ | $S_3(n)$ |
|---|---|---|---|---|---|
| ≤0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 | 0 |
| 3 | 1 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 1 | 0 |
| 5 | 1 | 1 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 |
| 8 | 0 | 0 | 1 | 1 | 1 |
| 9 | 0 | 1 | 0 | 1 | 1 |
| 10 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 0 | 1 | 0 |
| 12 | 1 | 1 | 1 | 0 | 1 |

**Induction proof of (3.3)**

Initialization condition: $l_k(n) = 0 \;\forall\, n \le 0$ or $n < k$, $S_0(n) = 0 \;\forall\, n \le 0$ and because of (3.4): $S_i(n) = 0 \;\; \forall\, n \le i$ and the first incoming value: $S_0(1) = d_1$,

Initialization test:

$l_1(1) = \big((1-1) \bmod (m+1) = 0\big) \oplus \big((1-1) \bmod (m+1) = 1\big) = 1 \oplus 0 = 1 \Rightarrow$ Assumption is true for the Initialization

Recursion:

$$S_0(n) = \left(\sum_{j=0}^{m-1} S_j(n-1)\right) + 1 \cdot d_n$$

$$\overset{(3.4)}{=} \left(\sum_{j=0}^{m-1} S_0(n-1-j)\right) + 1 \cdot d_n$$

$$= \left(\sum_{j=0}^{m-1} \sum_{k=1}^{n-1-j} l_k(n-1-j) \cdot d_k\right) + 1 \cdot d_n$$

$$= \left(\sum_{j=0}^{m-1} \sum_{k=1}^{n-1} l_k(n-1-j) \cdot d_k\right) - \left(\sum_{j=1}^{m-1} \sum_{k=n-j}^{n-1} \underbrace{l_k(n-1-j)}_{=0,\text{ see below}} \cdot d_k\right) + 1 \cdot d_n$$

$$= \left(\sum_{k=1}^{n-1} \sum_{j=0}^{m-1} l_k(n-1-j) \cdot d_k\right) + 1 \cdot d_n$$

$$= \left(\sum_{k=1}^{n-1} \big((n-1-k) \bmod (m+1) = 0\big) \oplus \big((n-2-k) \bmod (m+1) = 0\big) \cdots \right.$$

$$\cdots \oplus \big((n-2-k) \bmod (m+1) = 0\big) \cdots$$

$$\cdots \big((n-1-(m-2)-k-1) \bmod (m+1) = 0\big) \oplus \big((n-1-(m-1)-k) \bmod (m+1) = 0\big)$$

$$\left. \oplus \big((n-1-(m-1)-k-1) \bmod (m+1) = 0\big)\right) \cdot d_k + 1 \cdot d_n$$

$$= \left(\sum_{k=1}^{n-1} \big((n-k-1) \bmod (m+1) = 0\big) \oplus \big((n-k) \bmod (m+1) = 0\big) \cdot d_k\right)$$

$$+ \big((n-n-1) \bmod (m+1) = 0\big) \oplus \big((n-n) \bmod (m+1) = 0\big) \cdot d_n$$

$$= \sum_{k=1}^{n} \big((n-k-1) \bmod (m+1) = 0\big) \oplus \big((n-k) \bmod (m+1) = 0\big) \cdot d_k$$

Remark:

$$\sum_{j=1}^{m+1}\sum_{k=n-j}^{n+1} \underbrace{l_k(n-1-j)}_{=0,\text{ see below}} \cdot d_k = 0\text{, because}$$

$l_k(n) = 0\ \forall n < k$, here: $n-1-j < n-j$ and $n-1-j < n$

□

The corresponding cycle length for recursive encoders where all feedback taps are set is $n_{cycle} = m+1$. This can cosily be seen for a base sequence. The contents of the encoder memory are equal every m+1 time units and therefore the cycle length has to be m+1.

## 3.3 Improved trellis for the Turbo Code scheme

The choice of the generator polynomials for a given memory order m influences the performance of the Turbo Code.
Calculations of the distance spectrum as well as simulations for pseudo random interleavers have shown that the cycle length of the linear feedback shift register should be $2^m - 1$. This maximal cycle length can be achieved with a primitive feedback polynomial. In a subsequent chapter, there is an explanation of why it is crucial to use maximum cycle length shift registers.

The feedback shift register of a maximum cycle length code has another advantage. According to [16] it is not possible for a base information sequence that all of the ones of the output are on even or odd positions. This means that it is not possible to cancel all the ones with puncturing. This would reduce the distance dramatically.

While the feedback path is mainly responsible for the cycle length, the feedforward polynomial has to to be choosen such that catastrophic encoders are avoided. If the case of catastrophic encoders is excluded, it is not so important which feedforeward generator is used.

### 3.3.1 Distance spectrum

A change of the feedback path has an influence on the distance spectrum of the code. Unfortunately, it is very difficult to calculate the whole distance spectrum of the Turbo Code system given in [1], because the calculation time for such big interleavers is very high. Therefore, the effects for interleavers of size 32 are considered to illustrate the basic principles.

The following figure shows the distance spectrum for an encoder with memory three and with different feedback polynomials for an interleaver of size 32. As described above, the best results can be achieved with maximum cycle length feedback shift registers.

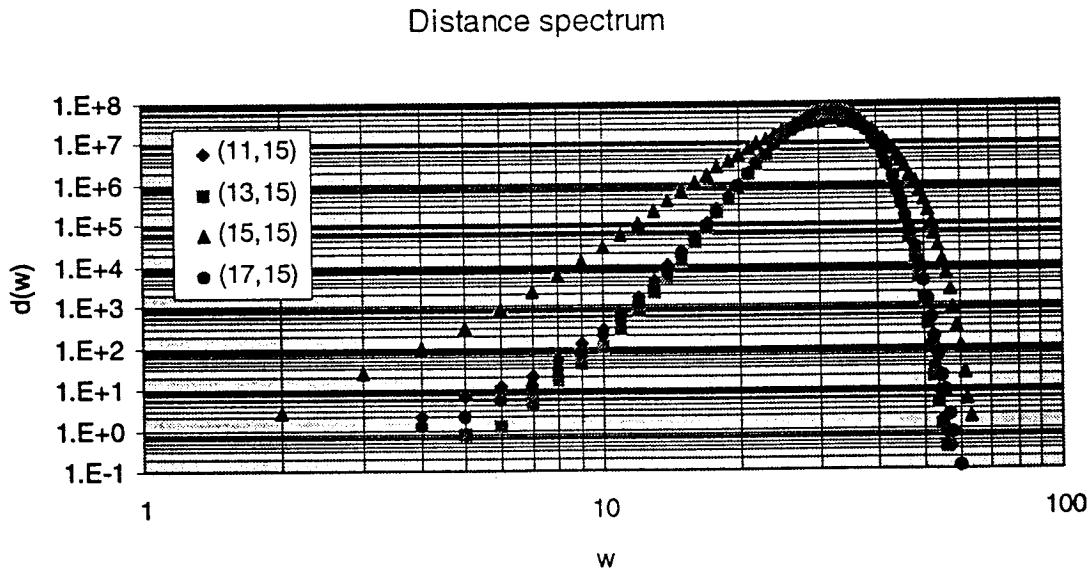



Figure 3.4 - Distance spectrum of turbo codes with different feedback paths (Blocksize: 32, Iterations: 10)

# 4 Interleaver

Interleavers are commonly used on channels with memory so that they appear to the channel decoder as memoryless. In most coding systems, they have no influence on the performance, if memoryless channels are used. This is not the case in the turbo coding scheme. In this scheme, the interleaver is an integral part of the encoder and can dramatically effect the performance even on memoryless channels.

The interleaving process scrambles the order of the code symbols $d_n$ in time while the deinterleaving process unscrambles the recovered symbol stream into the original sequence.

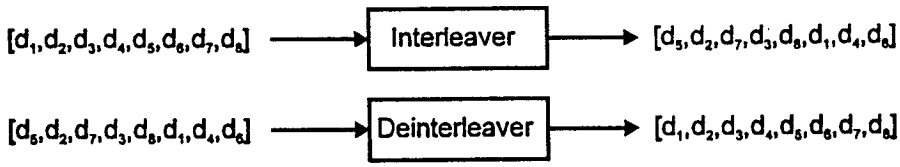



Figure 4.1: Interleaving process

This chapter discusses different kinds of interleavers and gives some rules for building good interleavers for use with Turbo Codes.

## 4.1 Interleaver design considerations

The original turbo code [1] uses a pseudo-random interleaver. For this class of interleavers, the only condition to the interleaver function G(i) is that the transformation is bijective.

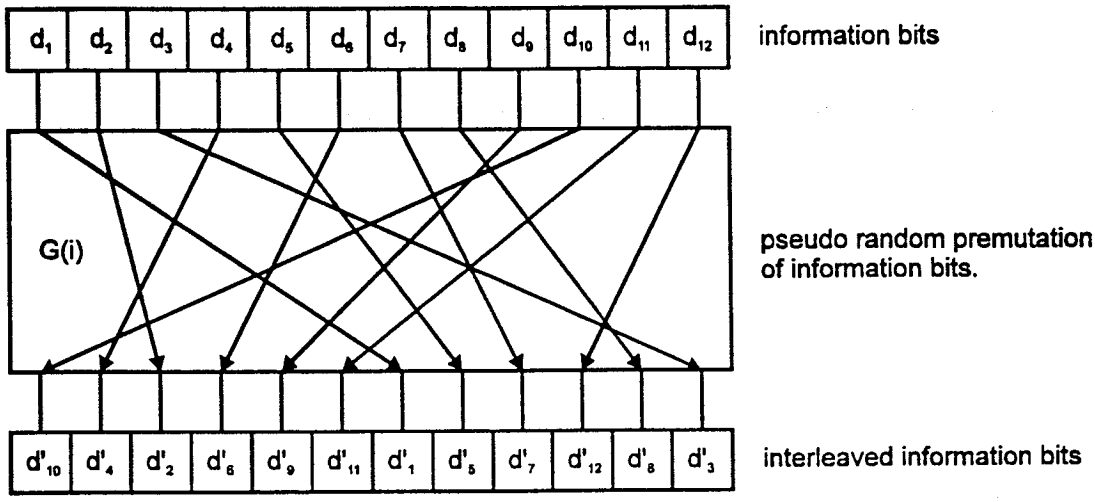



Figure 4.2: Pseudorandom Interleaver

The transformation itself is a pseudorandom process, which is defined once and does not change during the encoding / decoding process. Pseudorandom interleavers follow no special rules and can therefore result in a transformation, which leads to a low Hamming distance. The following figure shows the error correcting capability of the Turbo Code, using a typical pseudorandom interleaver of size 400.

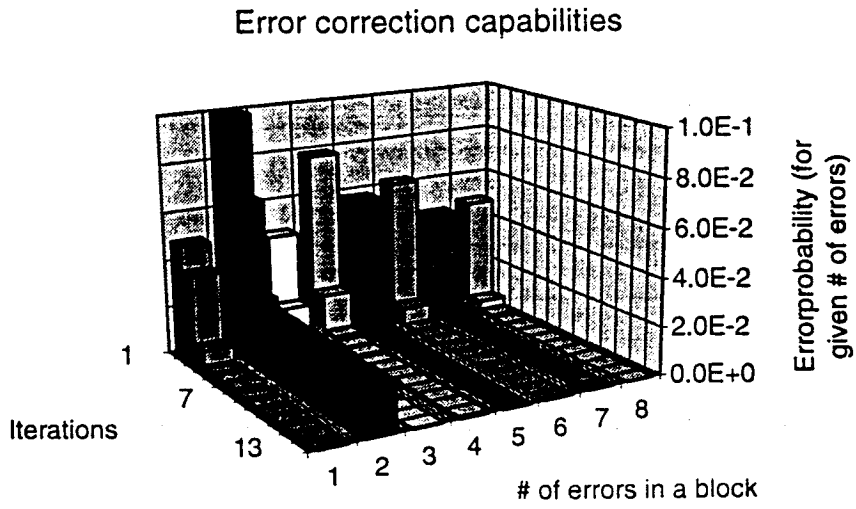



Figure 4.3: Random Interleaver (Blocksize: 400, $E_b/N_0$: 1.9 dB)

Most of the errors can be corrected through the iteration process. However, there are some errors which cannot corrected and 'fall through'. It is significant that error patterns of odd weight seldom occur and that when an error occurs it is with a high probability a two bit error. A brief look at the bit error probability on different inlerleaver / block positions shows that the bit errors are concentrated in certain positions.

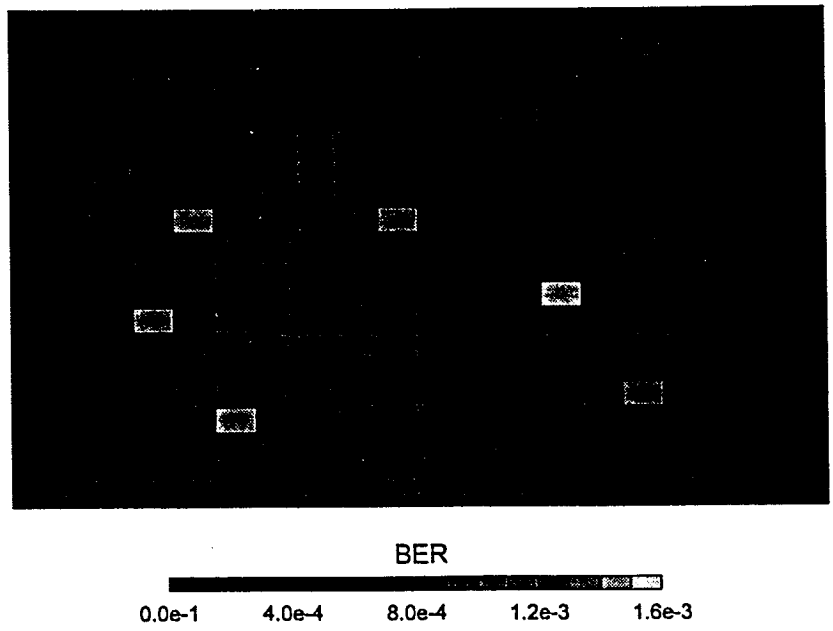



Figure 4.4: BER on different bit-positions within a block

The positions with a high bit error rate are given by a special interleaver symmetry, which leads to a small weight for certain codewords. An information bit sequence with just two ones is an example for such a case.

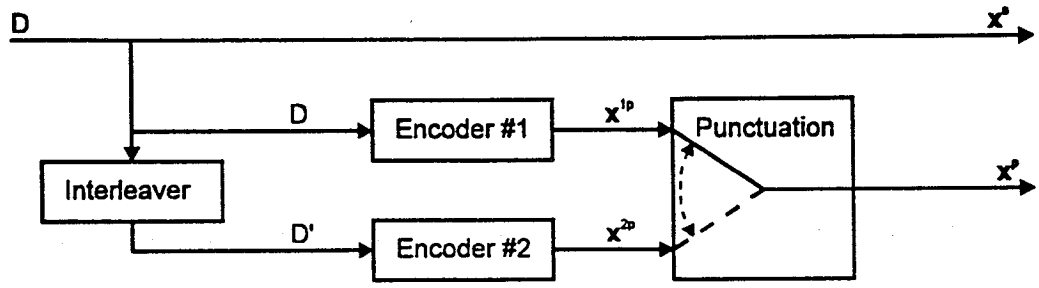



Figure 4.5: Encoder structure

If $d_i$ and $d_{i+cycle}$ are equal to one and all other bits are zero, then $x^{1p}$, the output of encoder 1, has only two ones after puncturing. A worst case interleaver will just move both information bits by n. Then there will be only two parity check bits in $x^{2p}$, the output of encoder 2. The resulting distance from the all zero codeword is only 6.

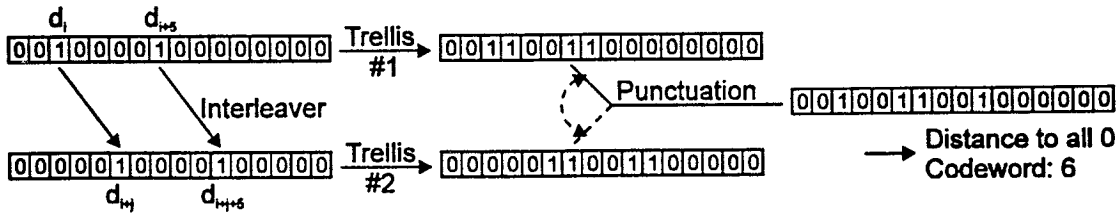



Figure 4.6: Turbo-code with small free distance

A turbo code using the bad interleaver shown in Figure 4.6 has poor error correction capabilities. This kind of interleaver cannot avoid that there are many two bit error, called errors of order 1, in the code. Errors of order n are defined as the number of two by two errors in a block (e.g. Error of order 2: 2 pairs of errors).

Applying the following rules to the design of the interleaver decreases the probability of errors of order one and two:

Decreasing errors of order 1

Maximize

$$\min\left(|i-j|+|G(i)-G(j)|\right), \forall i,j, \text{where}$$
$$i \neq j, i \bmod n_{cycle} = j \bmod n_{cycle}, G(i) \bmod n_{cycle} = G(j) \bmod n_{cycle} \quad (4.1)$$

i,j are the positions of the switching bits and G is the transfer function of the interleaver.

Decreasing errors of order 2

Maximize

$$\min\left(|i_1-i_2|+|j_1-j_2|+|G(i_1)-G(j_1)|+|G(i_2)-G(j_2)|-Ov(i,j)-Ov(G(i),G(j))\right),$$
$$\forall i_1,i_2,j_1,j_2, \text{ where } i_1 \neq i_2 \neq j_1 \neq j_2, i_1 \bmod n_{cycle} = i_2 \bmod n_{cycle},$$
$$j_1 \bmod n_{cycle} = j_2 \bmod n_{cycle}, G(i_1) \bmod n_{cycle} = G(j_1) \bmod n_{cycle}, \quad (4.2)$$
$$G(i_2) \bmod n_{cycle} = G(j_2) \bmod n_{cycle}$$

The Overlap Ov(i,j) is defined like showed in Figure 4.7. The example showed in Figure 4.8 has no Overlap.

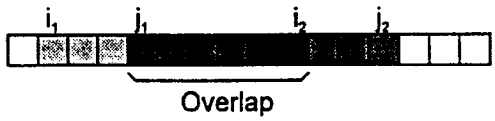



Figure 4.7: Overlap in Error of order 2

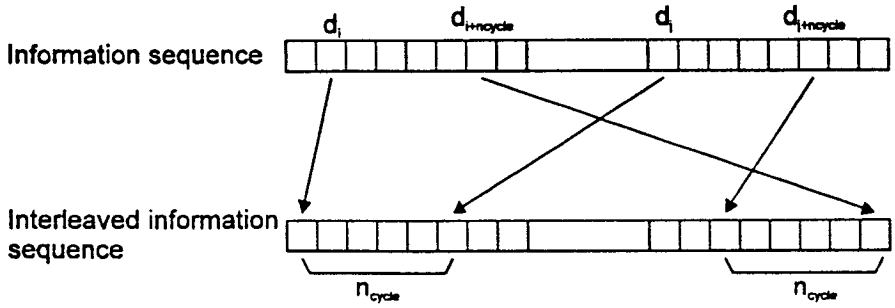



Figure 4.8: Bad interleaver for 2nd order errors

Applying these rules will encrease the performace. Unfortunately, it is only possible to decrease errors of order up to two with a reasonable effort on computational power. Figure 4.9 shows an interleaver where the minimum of (4.1) and (4.2) is 30.

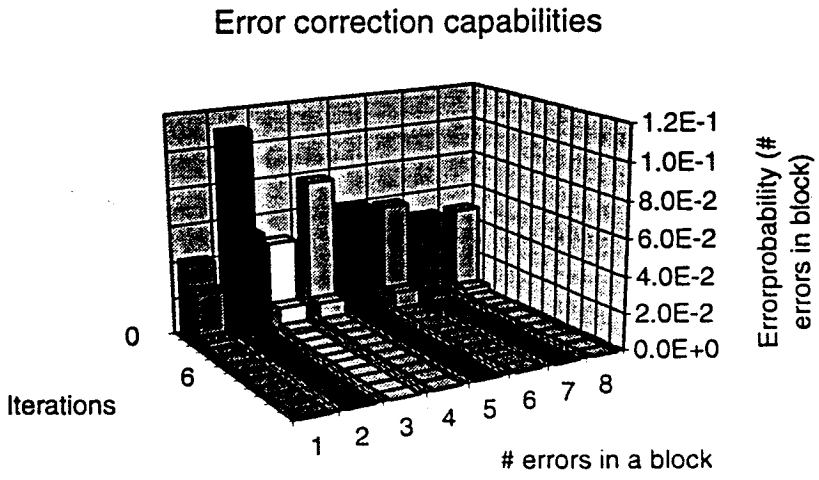



Figure 4.9: Error correction capabilities of improved interleavers (Blocksize 400, $E_b/N_o$ 1.9 dB)

The influence of the interleaver improvement on the two first iterations is small. But with an improved interleaver, there are less errors which can 'fall trough' and this improves the performance.

The resulting free distance of the turbo code is a function of the trellis as well as the interleaver. Simulations have shown, that optimized 400 bit Interleavers are better than interleavers of size 1000 for a signal to noise ratio above 2 dB.

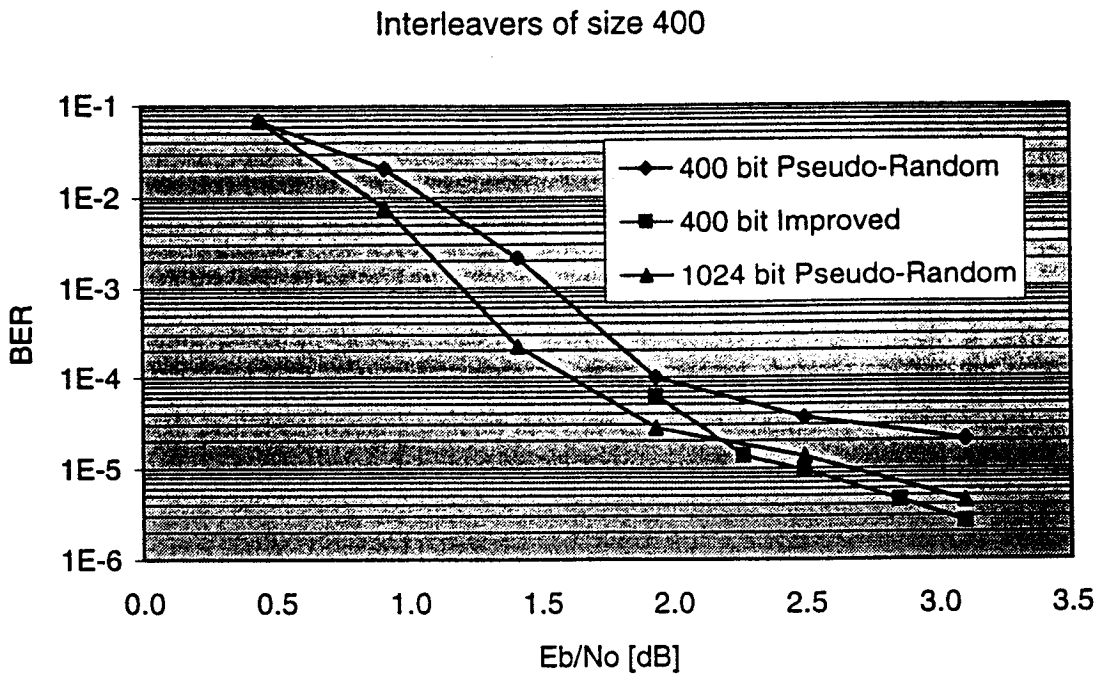



Figure 4.10: BER of improved interleavers

## 4.2 Interleaver with maximum cycle length encoders

The improvement rules explain why it is important to use a maximum cycle length encoder. According to (4.1) a bad transformation is given if $i \bmod n_{cycle} = j \bmod n_{cycle}$ and $G(i) \bmod n_{cycle} = G(j) \bmod n_{cycle}$. Therefore the probability of choosing a bad interleaver transformation for the element j is

$$P_{bad} = \frac{1}{n_{cycle}} \qquad (4.3)$$

In order to minimize bad interleaver transformations one has to maximize the cycle length.

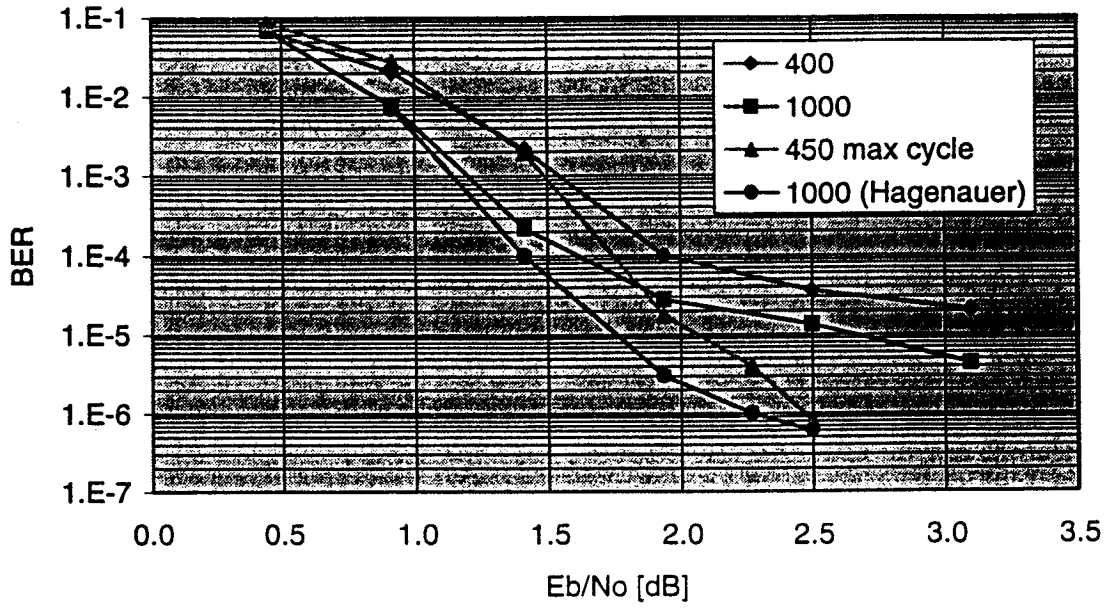



Figure 4.11: Influence of maximum cycle length code to the performance

Another way to improve the average weight of a crucial codeword is to increase the size of the interleaver. If a bad interleaver transformation is given, then the average weight of the parity bit sequence is

$$\bar{d} = w \cdot \frac{\sum_{i=1}^{N'} (N'-i)\cdot i}{\sum_{i}^{N'} (N'-i)} = w \frac{\frac{1}{6}\cdot\left(N'^3 - N'\right)}{\frac{1}{2}\cdot\left(N'^2 - N'\right)} = \frac{w}{3}(N'+1)$$

where w is the number of ones in a cycle $w \cong \frac{n_{cycle}}{2}$ and $N' = \frac{N}{n_{cycle}}$, where N is the interleaver size. So the average weight of the parity bit sequence is

$$\bar{d} \cong \frac{1}{6}\cdot n_{cycle}\cdot\left(\frac{N}{n_{cycle}}+1\right) = \frac{1}{6}\cdot\left(N + n_{cycle}\right) \qquad (4.4)$$

The observation that a bigger interleaver size will lead to better performance has been made several times in the literature[7].

## 4.3 Distance spectrum

The distance spectrum of the code is partly a function of the interleaver. Calculations of distance spectrums for small interleaver have shown, that the average distance remains the same, even the interleaver is improved. With an improved interleaver the variance of the distance spectrum decreases and the minimum distance is increased. The higher free distance lowers the error floor. This explains why a performance increase can only be shown for a signal to noise ratio above 2 dB in Figure 4.10. For lower SNR's the whole distance spectrum is responsible for the performance and, because the average distance does not change, an improvement cannot be expected there.

## 4.4 Terminating interleavers

One of the problems with the Turbo Code scheme is, that it is not possible to terminate both encoders with a general interleaver. The original MAP algorithm cannot be used for the second encoder. This will lead to slightly lower performance especially for small interleavers.

There exist $(n_{cycle}) \cdot \left(\frac{N}{n_{cycle}}\right)!$ different interleavers that terminate both encoders for every interleaver size N, where N is a multiple of $n_{cycle}$. For these block sizes, a permutation of elements such that

$$i \bmod n_{cycle} = G(i) \bmod n_{cycle} \quad \forall i \tag{4.3}$$

where G(i) is the interleaver function, has no influence on the content of the memory cells at the end of the block. Therefore both trellis will terminate.

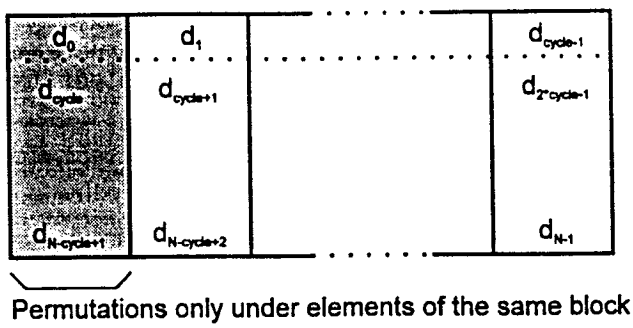



Figure 4.12 - Permutations for terminating interleavers

The following proof is valid only for encoders where all feedback taps are equal to one (See Figure 3.3). The terminology is the same like in (3.3)

If N mod (m+1)=0 the contents of the memory cells can written as follow:

$$S_j(N) = \sum_{k=1}^{\frac{N}{m+1}} d_{k\cdot(m+1)-j} + \sum_{k=1}^{\frac{N}{m+1}} d_{k\cdot(m+1)-j-1} \quad \forall 0 \le j < m \tag{4.4}$$

**Proof:**

$$\begin{aligned}
S_j(N) &= \sum_{k=1}^{\frac{N}{m+1}} d_{k\cdot(m+1)-j} + d_{k\cdot(m+1)-j-1} \\
&= \sum_{k=1}^{N} \big((-k-j) \bmod (m+1) = 0\big) d_k \oplus \big((-k-j-1) \bmod (m+1) = 0\big) d_k \\
&= \sum_{k=1}^{N} \big((N-k-j) \bmod (m+1) = 0\big) d_k \oplus \big((N-k-j-1) \bmod (m+1) = 0\big) d_k \\
\Rightarrow S_0(N) &= \sum_{k=1}^{N} \big((N-k) \bmod (m+1) = 0\big) d_k \oplus \big((N-k-1) \bmod (m+1) = 0\big) d_k \\
\Rightarrow S_j(N) &= \sum_{k=1}^{N-j} \big((N-k-j) \bmod (m+1) = 0\big) d_k \oplus \big((N-k-j-1) \bmod (m+1) = 0\big) d_k \\
&\quad + \underbrace{\sum_{k=N-j+1}^{N} \big((N-k-j) \bmod (m+1) = 0\big) d_k \oplus \big((N-k-j-1) \bmod (m+1) = 0\big) d_k}_{=0,\text{because of Initialization}} \\
&= S_0(N-j)
\end{aligned}$$

Addition is a commutative operation, so a change in the order has no influence on the result and both encoders will be in the same (zero) state at the end of the block. The second encoder terminates if and only if $i \bmod n_{cycle} = G(i) \bmod n_{cycle} \; \forall i$, where G is the interleaver function.

It is possible to build optimized interleavers under these conditions. However the performance of terminating interleavers will be worse in general because of the reduction in the freedom in placing the elements. Simulations have shown, that it is only worth to make self terminating interleavers, if their size is less than 1000.

A special class of interleavers, where both encoders are terminated, are helical interleavers, as discussed in earlier works [8][9]. No interleaver optimization will be made for this class

of interleavers, so that the axis ratio of the helical interleaver is crucial to the performance. The performance of the Turbo Code with a good and a bad helical interleaver is show in the following two examples.

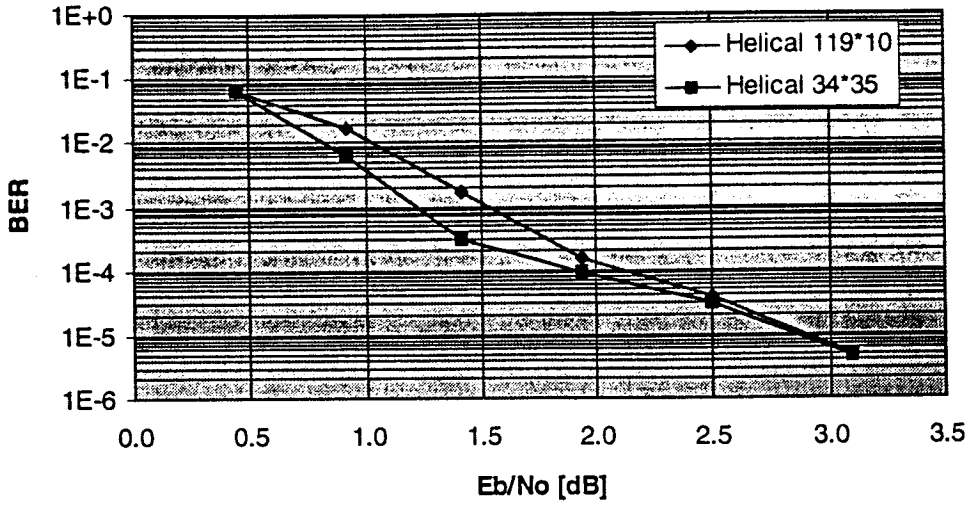



Figure 4.13: Helical interleaver of the size 1190 with different axis ratios

The helical interleaver with the axis ratio of about 1 clearly gives the better results. A look at the distribution of the errors within a block shows that the errors are concentrated on a band by the 119*10 helical interleaver

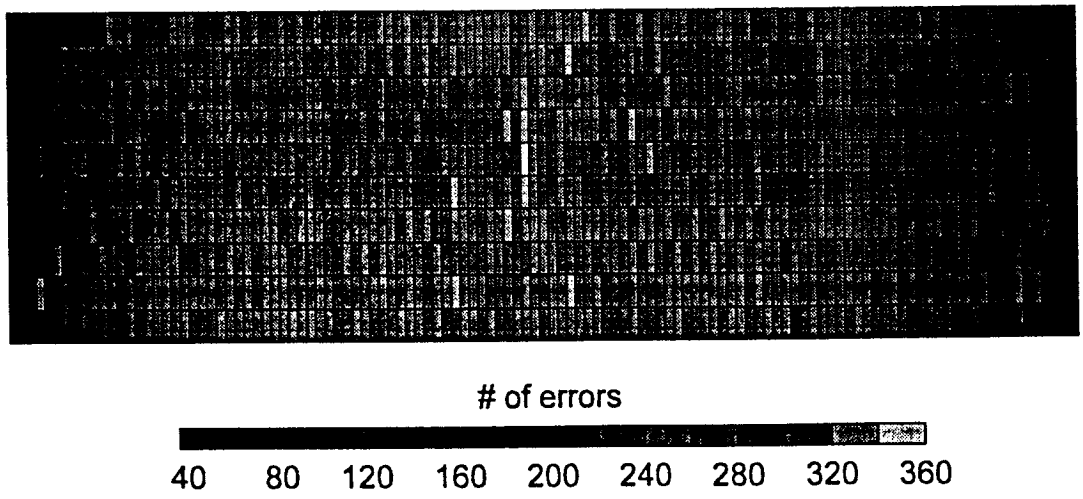



Figure 4.14: Errors in different block positions (119*10 Helical interleaver, $E_b/N_0$: 1 dB)

The error band corresponds with the symetry of the helical interleaver. Calculating the distance given by (4.1) and (4.2) will show similar bands.

# 5 Quantization

Loss of error correcting performance due to quantization is well known. This loss can be made negligible by using an appropriate number of quantization levels. An eight level quantization scheme has turned out to be sufficient in practice.

However, all of the simulation results in the Turbo Code literature were obtained without quantization (to the best of the authors. Therefore, the question arises whether quantization has an unexpected influence on Turbo Codes. That is on the performance improvement with increasing number of iterations.

## 5.1 Optimized Quantization

For many applications, the results achieved by equal spaced quantization is sufficient. To get better results and to say something about the quality of equal spaced quantization, it is possible to apply the optimized quantization presented by Massey [13]. He showed that if the likelihood ratio is defined as

$$\lambda(\Re_j) = \frac{\int_{\Re_j} p_Y(y|0)}{\int_{\Re_j} p_Y(y|1)}, \tag{6.1}$$

then the optimum values for the likelihood thresholds are

$$T_j = \sqrt{\lambda(\Re_{j-1}) \cdot \lambda(\Re_j)} \; \forall j. \tag{6.2}$$

For an AWGN channel the probability density function of the received signal is given by

$$p_Y(y|0) = \frac{1}{\sqrt{\pi \cdot N_0}} \exp\left( -\frac{\left(y - \sqrt{E_s}\right)^2}{N_0} \right), \; p_Y(y|1) = \frac{1}{\sqrt{\pi \cdot N_0}} \exp\left( -\frac{\left(y + \sqrt{E_s}\right)^2}{N_0} \right) \tag{6.3}$$

Equation (6.3) can be normalized and written as

$$p_S(s|0) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y-a)^2}{2}\right), \; p_S(s|1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y+a)^2}{2}\right)$$

, where $a = \sqrt{2\frac{E_s}{N_0}}$ and $s = \sqrt{\frac{2}{N_0}}y$.

With the following iterative procedure it is possible to find the optimized likelihood thresholds $T_j$ and quantization thresholds $I_j$. The example is made for an n-Level quantizer. The initialization conditions are $I_0 = -\infty$, $I_n = \infty$, $T_0 = 0$ and $T_n = \infty$

1. Choose $I_1$ arbitrarily.
2. Calculate

$$\lambda(\Re_0) = \frac{\int_{-\infty}^{I_1} p_S(s|0)ds}{\int_{-\infty}^{I_1} p_S(s|1)ds}$$

3. Find $I_2$ using (6.1) and (6.2). First calculate

$$\lambda(\Re_1) = \frac{T_1^2}{\lambda(\Re_0)}$$

then find $I_2$ from

$$\lambda(\Re_1) = \frac{\int_{I_1}^{I_2} p_S(s|0)ds}{\int_{I_1}^{I_2} p_S(s|1)ds}$$

4. Find $I_3$ through $I_{n-1}$ in the same manner by repeating 2 and 3
5. Calculate

$$\tilde{\lambda}(\Re_{n-1}) = \frac{T_{n-1}^2}{\lambda(\Re_{n-2})}$$

If $\tilde{\lambda}(\Re_{n-1}) > \lambda(\Re_{n-1})$, choose a smaller value for I1 and return to 2. If $\tilde{\lambda}(\Re_{n-1}) < \lambda(\Re_{n-1})$, choose a bigger value for I1 and return to 2. Stop if $\tilde{\lambda}(\Re_{n-1}) \approx \lambda(\Re_{n-1})$.

The conversion between the likelihood and the quantization thresholds is done in the following way

$$T_j = \frac{p_s(s|0)}{p_s(s|1)} = \frac{\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(I_j - a)^2}{2}\right)}{\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(I_j + a)^2}{2}\right)} = \exp(2 \cdot a \cdot I_j).$$

## 5.2 Simulation Results

The quantization results scheme affects the performance of the code. This depends on the number of quantization levels and the appropriate choice of them.

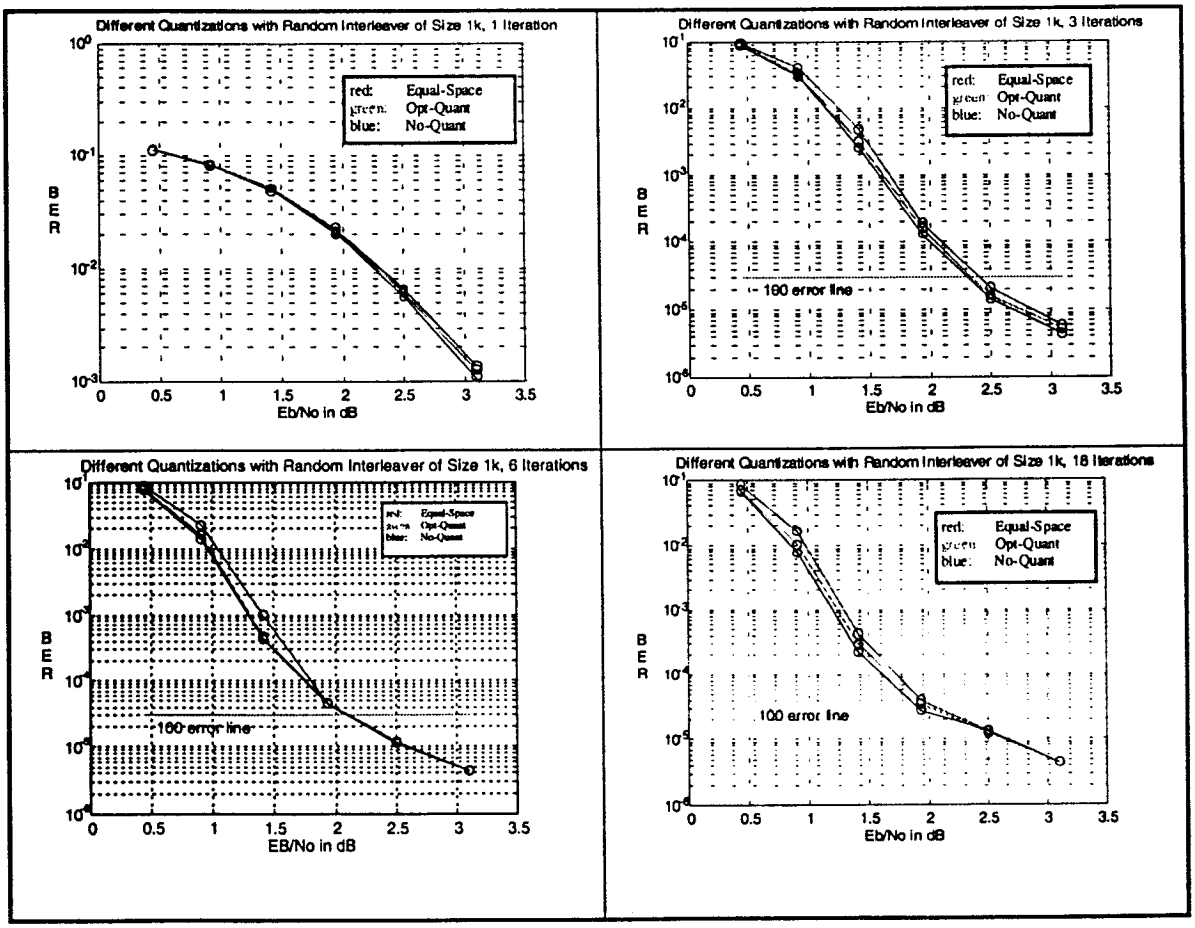



Figure 5.1: Comparision of different quantization methods on different iterations for 32-level quantization

Figure 5.1 shows simulation results where different quantization methods are compared for several iterations. The first method is called the 'Equal-Space' method where the differences between all quantization levels are equal to 1/16 for 32-level quantization; i.e. they are

independent of the Signal-to-Noise-Ratio (SNR). The second method, called 'optimal quantization', is also a 32-level quantization method but the levels are now dependent on the SNR, as proposed in [4]. As a comparison, unquantized decoding was used to obtain the results of the blue graph. The best results are achieved without quantization. Choosing the quantization levels as a function of the SNR minimizes the loss of performance caused by quantization.

These observations can be summarized as follows. The simulations results show that quantization leads to the expected performance degradation. It is interesting that this shifting is fully maintained throughout all iterations.

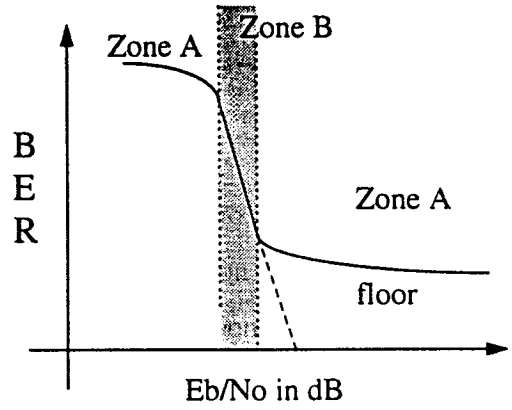



Figure 5.2: BER function of the Turbo-Code with floor (dashed without floor)

In conclusion it can be said that quantization has no unexpected influence on Turbo Codes. However, one detail has to be mentioned. Since the BER-function of the Turbo Code has an error floor, a small dB loss results in much greater BER loss in zone B than in zone A (see Figure 5.2).

# 6 Improvements on the Turbo Coding scheme

In the original turbo-code scheme the output information sequence is coming out of the decoder two. This has the disadvantage that the output is based on a nonterminating trellis, if a general interleaver is used. By changing the decoding order and decoding the open trellis first, it is possible to improve the performance.

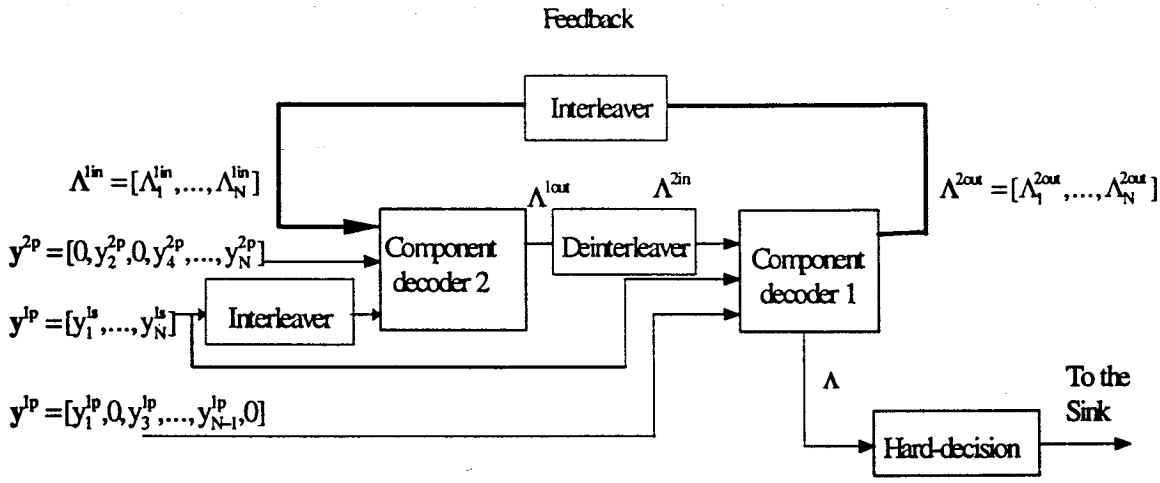



Figure 6.1: Modified Turbo-Code scheme

The change of the decoders has the second advantage of a decreased complexity, because no deinterleaver is necessary for the output.

The following figure compares the performances of the two decoder positions. The same interleaver, trellis is used for both decoding schemes.

# 7 Conclusions

We have explained the structure of the turbo-code scheme. The results claimed in [2] and [7] can be reproduced with the simulation programs except for the optimized 1024 bit interleaver from [7].

It is possible to increase the performance of the turbo-code by using maximum cycle length linear feedback shift registers in the encoder. The use of them decreases the probability of low weight code words.

An interleaver modification rule was presented which allows the construction of better codes. In combination with maximum cycle length linear feedback shift registers better results could be achieved than presented in [7].

Finally, we have given the class of interleavers which allows that both encoders of the turbo-code ends in the same state.

# 8 Acknowledgements

The authors would like to thank Dr. L. Perez, B. Keusch and J. Sayir for their hints and many helpful discussions and comments.

# Appendix A - References


[1] C. Berrou, A. Glavieux and P. Thitimajshima, „Near Shannon limit error-correcting coding and decoding: TURBO codes", Proc. ICC'93, Geneva, Switzerland, 1993, pp. 1064-1070.

[2] C. Berrou, A. Glavieux, „Turbo-Codes: General Principles and Applications", Proc. 6th Tirrenia WS on Dig. Comm., Tirrenia, Italy, 1994, pp. 215-226.

[3] S. Lin, D. Costello, „Error Control Coding", Prentice Hall, NJ 1983.

[4] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, „Optimal decoding of linear codes for minimizing symbol error rate", IEEE Trans. on Inform. Theory, IT-20, pp. 284-287, March 1974.

[5] J. Hagenauer, L. Papke, „Decoding 'TURBO'-codes with the soft output Viterbi algorithm (SOVA) „, Proc. 1994 IEEE Int. Sym. Inform. Theory, Trondheim, Norway, 1994, p. 164.

[6] J. Hagenauer, L. Papke and P. Robertson, „Iterative ('TURBO') Decoding of Systematic Convolutional Codes with the MAP and SOVA Algorithms",Submitted ITG 1994 Conf., Frankfurt, October 1994, pp.21-28.

[7] P. Robertson, „Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (TURBO) Codes", IEEE GLOBCOM, 1994, pp. 1298-1303.

[8] O. Joerssen, H. Meyr, „Terminating the trellis of turbo-codes", Electron. Lett., 1994, 30, pp. 1285-1286.

[9] A. S. Barbulescu, S. S. Pietrobon, „Terminating the trellis of turbo-codes in the same state", Electron. Lett., 1995, 31, pp. 22-23.

[10] J. B. Cain, G. C. Clark Jr. and J. M. Geist, „Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Likelihood Decoding", IEEE Trans. on Infrom. Theory, 1979, 25, pp. 97-100.

[11] P. Jung, M. Nasshan, „Dependence of the error performance of turbo-codes on the interleaver structure in short frame transmission systems", Electron. Lett., 1994, 30, pp. 287-288.

[12] P. Jung, M. Nasshan, „Performance evaluation of turbocodes for short frame transmission systems", Electron. Lett., 1994, 30, pp. 111-113.

[13] J. L. Massey, „Coding and Modulation in Digital Communication", Zurich Seminar, Zurich, 1974.

[14] J. D. Anderson, „The TURBO Coding Scheme", Report IT-146 ISSN 0105-854, 1994, Technical University of Denmark, Lyngby.

[15] A. J. Viterbi and J. K. Omura, „Principles of Digital Communication and Coding", New York, McGraw Hill Book Co., 1979

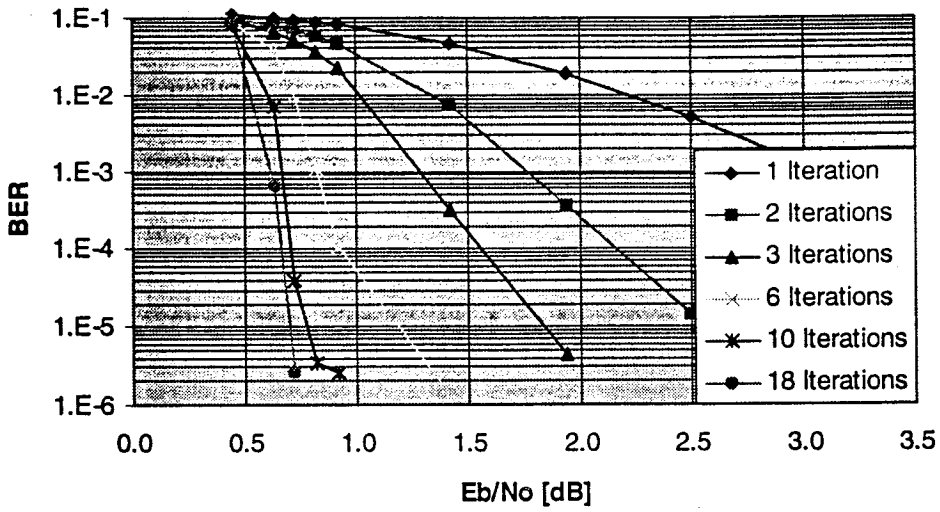
[16] F. Jessie MacWilliams, Neil J. A. Sloane, „Pseudo-Random Sequences and Arrays", Proceedings of the IEEE, Vol 64, No. 12, December 1976

[17] W. Press, S Teukolsky, W Vetterling, B. Flannery, „Numerical Recipes in C", Cambridge Press, 1992

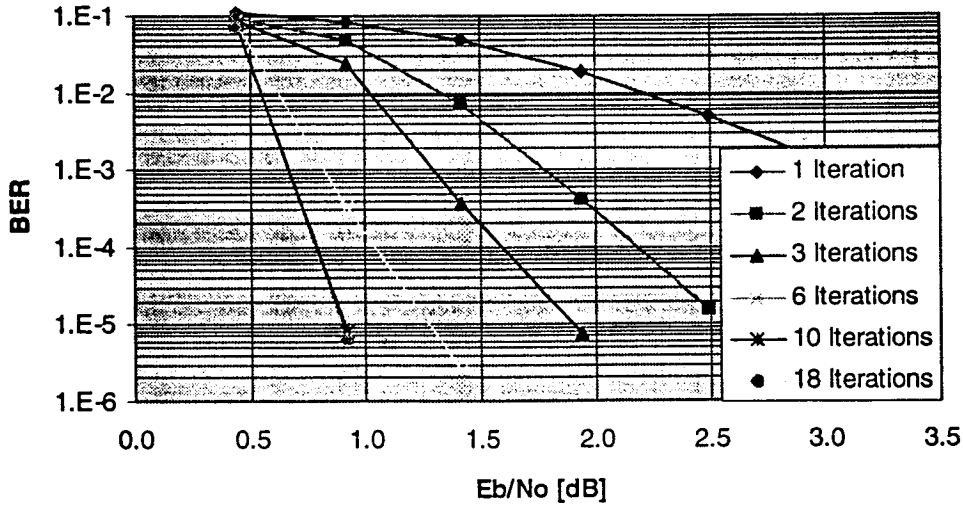
# Appendix B - Simulation results

***Bit error rate of Turbo Codes with different interleaver sizes***

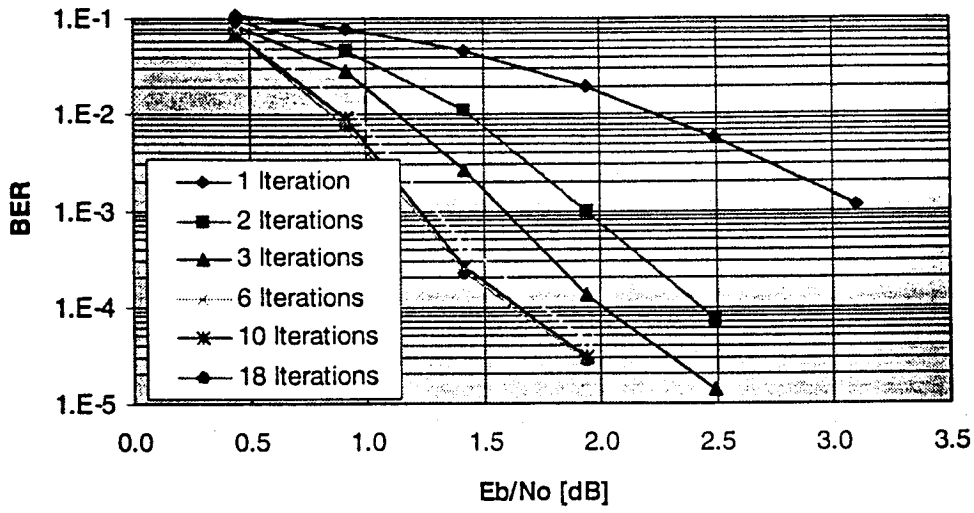
Interleaver size: 65536, Generator polynomials (37,21), Memory order: 4



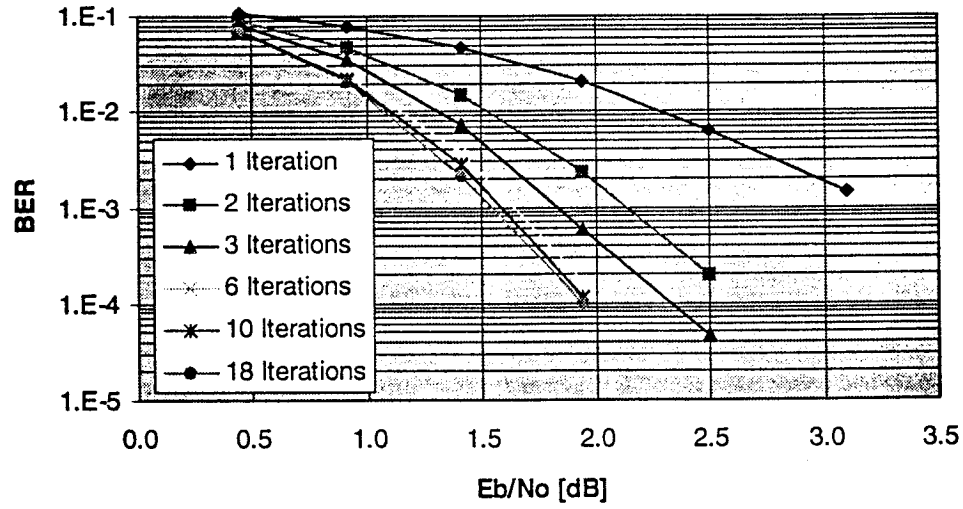

Interleaver size: 16384, Generator polynomials (37,21), Memory order: 4




Interleaver size: 1024, Generator polynomials (37,21), Memory order: 4



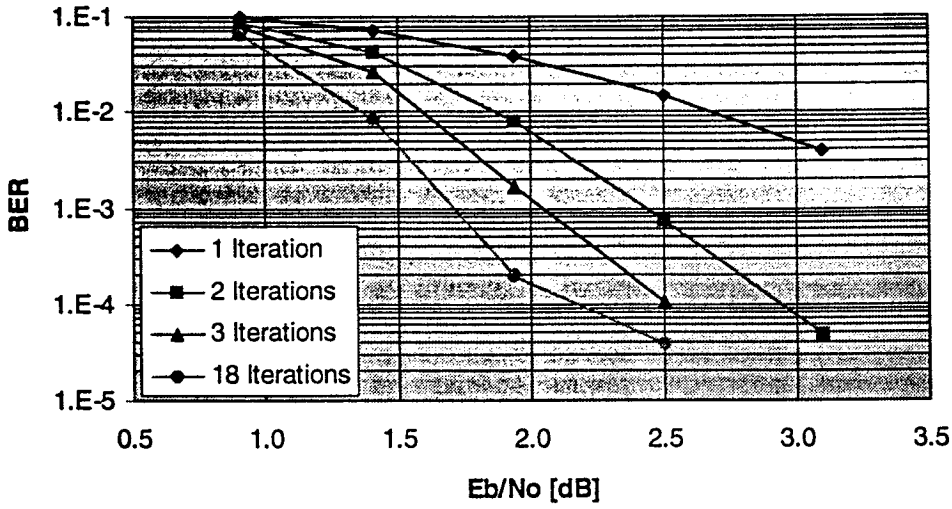

Interleaver size: 400, Generator polynomials (37,21), Memory order: 4



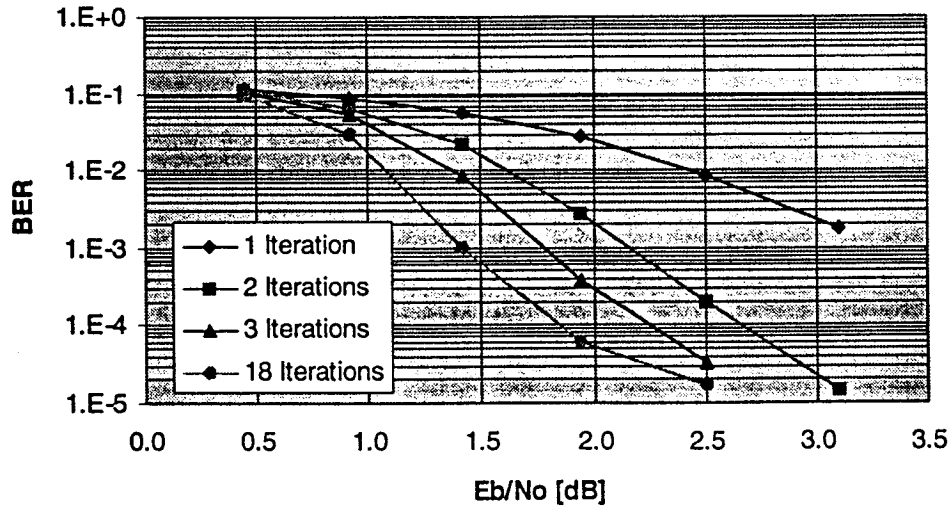
### *Simulations with quantization*

All simulations are done with an interleaver size of 400 and the generator polynomials (37,21).

Quantization levels: 4 (Equal spaced)




Quantization levels: 16 (Equal spaced)



# Appendix B

## On the Free Distance of Turbo Codes and Related Product Codes

ETH Eidgenössische Technische Hochschule Zürich

Ecole polytechnique fédérale de Zurich
Politechnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

*Signal- and Information Processing Laboratory*
*Prof. Dr. James L. Massey*

# On the Free Distance of TURBO Codes and Related Product Codes.

Jan Seghers

Professors:

Prof. Dr. James L. Massey

Prof. Dr. Daniel J. Costello, Jr.

Prof. dr. ir. Marc Moeneclaey

Advisors:

Dr. Lance C. Perez

Beat Keusch

Jossy Sayir

Diploma Project SS 1995

Nr. 6613

# Acknowledgements

The realisation of this project indebted me to many people.
I am especially thankful to Dr. Lance Perez, my first advisor. He has shown great patience introducing me to error control coding. His enthusiasm has stimulated me during this project. I have enjoyed the many discussions we had.
I thank Beat Keusch and Jossy Sayir, my co-advisors. I value their comments on my work very high. They have also introduced me to the 'ISI-world' and solved many practical problems.
I am also indebted to Prof. Dr. Daniel Costello for his interest in my work and his useful comments.
I am grateful that I had colleagues like Guido Meyerhans and Dieter Arnold, who have implemented the Turbo Coding scheme in their semester project. It was through their efforts that I was able to obtain the simulation results published in this report.
Last, but not least, I thank Prof. dr. ir. Marc Moeneclaey of the University of Ghent for evaluating my work.
Thank you very much,

Zürich, 17 August 1995.

ETH Eidgenössische Technische Hochschule Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut für Signal- und Informationsverarbeitung
Prof. Dr. J.L. Massey
Gloriastrasse 35
Durchwahl-Nr. 01/632 5192
Telefonzentrale 01/632 2211
Postadresse: ETH-Zentrum
8092 Zürich

Abt. III.B
Sommersemester 95
Dr. Lance C. Perez
Jossy Sayir
Beat Keusch

## Diploma Project

*for Jan Seghers*

# On the Free Distance of TURBO Codes and Related Product Codes

## 1 Introduction

In 1993, Berrou, Glavieux, and Thitimajshima [1] stunned the coding community with their announcement of a coding technique, which they called "TURBO Codes", that came within 0.7 dB of the Shannon limit for the additive white Gaussian noise channel used with a spectral efficiency of 0.5 bits per dimension. These claims were initially viewed with some skepticism, but subsequent research [2],[3] has verified the performance of this new coding scheme. TURBO codes and similar schemes are now an active research area throughout the world.

The TURBO Code utilizes a parallel concatenation of two identical punctured convolutional codes, whose encoders are realized in systematic feedback form, and a complex interleaving/deinterleaving scheme. The use of punctured systematic codes reduces the complexity of the decoder and allows higher spectral efficiencies to be achieved. The interleaver and deinterleaver serve two purposes. First, they perform the traditional role of redistributing and decorrelating the bursty errors out of the decoder. Second, they insure that both encoders do not simultaneously output low weight code sequences.

This encoding scheme is used in conjunction with an iterative decoding technique based on a modified version of the *maximum a posteriori* (MAP) algorithm introduced by Bahl, et. al. [4]. The MAP algorithm is used because it naturally produces reliability information about its decoding decisions. This "soft" information can then be used in the next iteration of decoding.

It is now widely believed that the primary contribution of Berrou et. al. [1], is the decoding algorithm and not the encoder structure.

Though the performance of the TURBO code has been reproduced, several fundamental questions concerning their performance and structure remain. The goal of this project is to address some of these questions, enumerated below, and to develop a general framework in which the properties and performance of TURBO codes can be easily understood. To assist you in this undertaking, you will have access to a computer simulation program for TURBO codes that is being written in a concurrent semester project. This program is to be used as an experimental tool to test and verify your hypotheses and conclusions, not simply to generate performance curves

# 2 Tasks

1. Familiarize yourself with the literature on TURBO codes and prepare a short, informal presentation on the state of the art.

2. Find the free distance of the TURBO code. This may be done by developing an appropriate computer algorithm or by analytical methods, but the answer must be exact.

3. Develop a relationship between the interleaver structure and the free distance of TURBO codes.

4. Examine and discuss the relationship between TURBO codes and product codes.

5. Explain the performance of the TURBO code in terms of its distance properties. In particular, the effect of iterated decoding and the presence of the error floor [2] should be addressed.

6. Determine a meaningful measure of the decoding complexity of the iterated MAP algorithm. Use this measure to make an intelligent performance versus complexity comparison between TURBO codes and other complex coding schemes.

7. Construct alternative coding schemes to the TURBO code. The program being developed in the semester project may be used to simulate these codes.

# References

[1] C. Berrou, A. Glaieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: TURBO codes," *Proc. ICC'93*, Geneva, Switzerland, 1993, pp. 1064–1070.

[2] J. D. Andersen, "The TURBO coding scheme," Report IT-146, Technical University of Denmark, June 1994.

[3] J. Hagenauer and L. Papke, "Decoding 'TURBO'-codes with the soft output Viterbi algorithm (SOVA)," *Proc. 1994 IEEE Int. Sym. Inform. Theory*, Trondheim, Norway, 1994, p. 164.

[4] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on Inform. Theory*, **IT-20**, pp. 284-287, March 1974.

# 3 Allgemeine Bestimmungen

Ca. 1 Woche vor Abgabedatum müssen sie einen 15-Minutigen Vortrag über Ihre Arbeit halten.

Der Bericht ist in zwei Exemplaren abzugeben. Beide Exemplare müssen unterschrieben sein. Das Original bleibt Eigentum des Instituts.

| | |
|---|---|
| Ausgabe: | Dienstag, 18. April 1995, ab 08.00 im ETZ F88 |
| Abgabe: | Donnerstag, 17. August 1995, bis 12.00 im ETZ F88 |

*Zürich, den 7. April 1995*

Prof. J.L. Massey

**Abstract**

It is the goal of this project to explain the performance of Turbo Codes. An algorithm for determining the free distance of Turbo Codes is proposed and applied to a few examples. The origin of the error floor is explained. The method of random interleaving is corrected and extended to punctured Turbo Codes. The distance spectrum of Turbo Codes is investigated using random interleaving. A theory is developed that explains the performance of Turbo Codes. The decoding complexity of Turbo Codes is reviewed. Constituent encoder optimization and interleaver design are discussed.

Chapter 1: *Introduction to Turbo Codes* introduces the Turbo Code's performance and the Turbo Coding scheme.
Chapter 2: *The Free Distance of Turbo Codes* presents an algorithm to calculate the free distance of a Turbo Code. This algorithm is applied to a few examples. The origin of the error floor is identified. The problem of interleaver design for optimal free distance is formalized.
Chapter 3: *The Distance Spectrum of Turbo Codes* explains how the distance spectrum determines the Turbo Code's performance. In a first analysis, the influence of the interleaver on the distance spectrum is shown. The method of random interleaving is corrected and extended to punctured Turbo Codes. Theoretical results are derived that show how the distance spectrum changes as the interleaver size increases. A theory is proposed that relates the Turbo Code's performance to its distance spectrum.
Chapter 4: *The Decoding Complexity of Turbo Codes* reviews the decoding complexity. The decoding complexity of a Turbo Code is compared to the decoding complexity of a convolutional code with Viterbi decoding.
Chapter 5: *Improvements for Turbo Codes* discusses constituent encoder selection and interleaver design. The good performance of Turbo Codes with maximum cycle length constituent encoders is explained.
Chapter 6: *The Relation Between Turbo Codes and Product Codes*
Chapter 7: *Conclusions*

# Contents

# Chapter 1

# Introduction to TURBO Codes.

## 1.1 The Turbo Code's performance.

In [1], Berrou et al. claim to have build a code that comes within 0.7 dB of the Shannon limit for the additive white Gaussian noise channel used with a spectral efficiency of 0.5 bits per dimension. Since these results were first published, several research groups have been able to verify this performance [5], [6]. Figure 1.1 shows simulation results for a Turbo Code with constituent encoder and puncturing pattern as defined in [1], and a 64K pseudorandom interleaver. These simulations were performed at this institute by Arnold and Meyerhans [18]. Their results confirm the claim made by Berrou et al: Only a signal to noise ratio of 0.7 dB is required to obtain a bit error rate of $10^{-5}$.
Performance curves of Turbo Codes show a typical shape, which is unlike that of any other code. First, their is a rather flat part for moderate signal to noise ratios. In the literature [5], this part of the performance curve has been labeled as the 'error floor'. Below a certain signal to noise ratio, the steepness of the curve suddenly increases. It is the combination of the error floor and the drastic increase in steepness that puzzles the coding community. In order to appreciate the performance of Turbo Codes, the Turbo Code's performance is compared to that of a (2,1,14) convolutional code with Viterbi decoding. This convolutional code was build for use on the Galileo space probe and has a free distance of eighteen, see [19]. At a bit error rate of $10^{-5}$ the Turbo Code offers a coding gain of approximately 2 dB compared to this complex convolutional code.

Figure 1.2 shows how the performance of Turbo Codes evolves as the interleaver size increases. These Turbo Codes consist of the same constituent encoders and puncturing pattern as the code mentioned above. The interleavers are pseudorandom and of sizes: 100, 400, 1024. Two effects call the attention: As the interleaver size increases, the error floor shifts downward and dominates the performance down to lower signal to noise ratios. The combination of these effects causes the performance curve to grow steeper as the interleaver size increases.
It is the goal of this project to explain these effects.

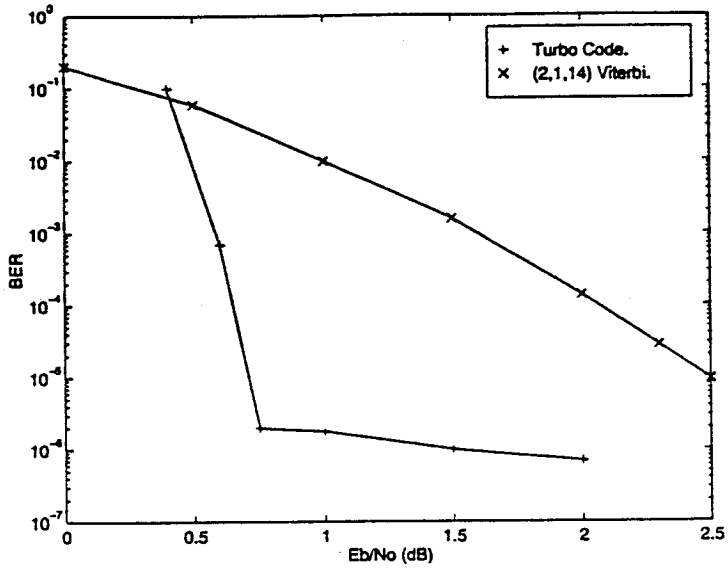



Figure 1.1: The performance of a Turbo Code compared to the performance of a (2,1,14) convolutional code with Viterbi decoding.

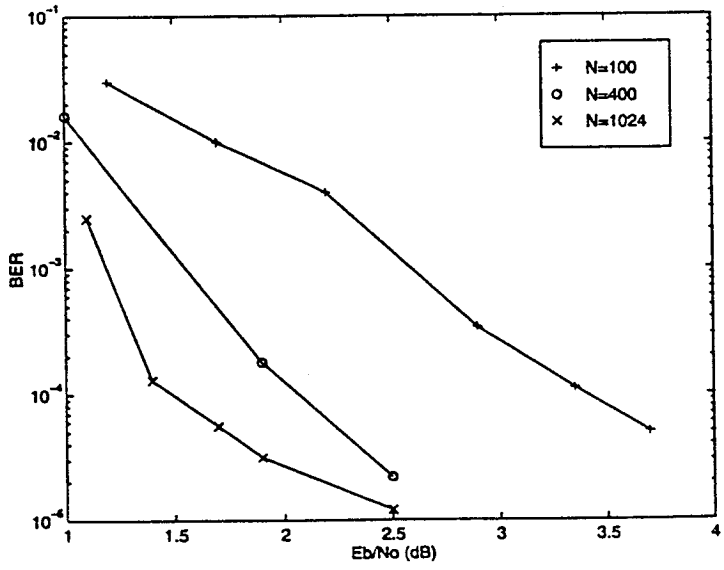



Figure 1.2: The performance of Turbo Codes for increasing interleaver size.

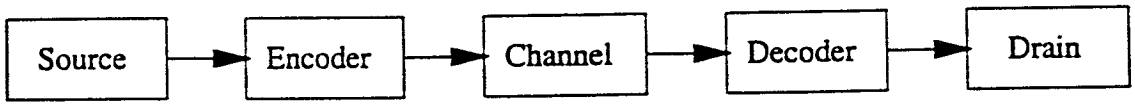



Figure 1.3: The Turbo coding scheme.

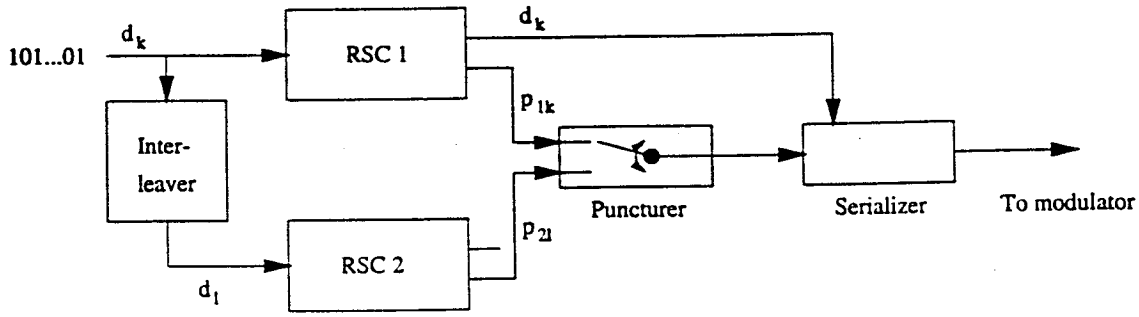



Figure 1.4: The Turbo encoder with puncturing pattern as in [1].

## 1.2 The coding scheme.

Figure 1.3 shows a block diagram of the Turbo coding scheme. The source emits a block of information bits with block length $N$. This information block is encoded and transmitted over a binary input additive white gaussian noise channel. The Turbo decoder produces estimated values for the information bits
The encoder and decoder are now separately discussed.

The Turbo encoder, see Figure 1.4, is a parallel concatenation of two recursive systematic convolutional encoders, called the constituent encoders. The information block, which has a finite length, is encoded by the first constituent encoder, but interleaved before it enters the second constituent encoder. Note that the interleaver operates on an information sequence, and not on a code sequence. The interleaver size equals the information block length. The output of the constituent encoders is punctured to increase the spectral efficiency.
In [1], Berrou et al. propose constituent encoders with generators (37,21). The systematic output of the second encoder is completely punctured. On even time steps the parity bit of the second encoder is punctured, on odd time steps the parity bit of the first encoder. This puncturing pattern leads to a rate of $\frac{1}{2}$ for the Turbo Code.
Berrou et al. have used pseudorandom 64K interleavers for their simulations. A pseudorandom interleaver is randomly generated, but then used unaltered throughout the encoder's operation.
Berrou et al. do not mention how the constituent encoders are terminated. Since the encoders are recursive, it does not suffice to set the last M information bits to zero in order to drive the encoder back to the all zero state. The necessary termination sequence depends on the state of the encoder at the moment that the termination is to begin. Note

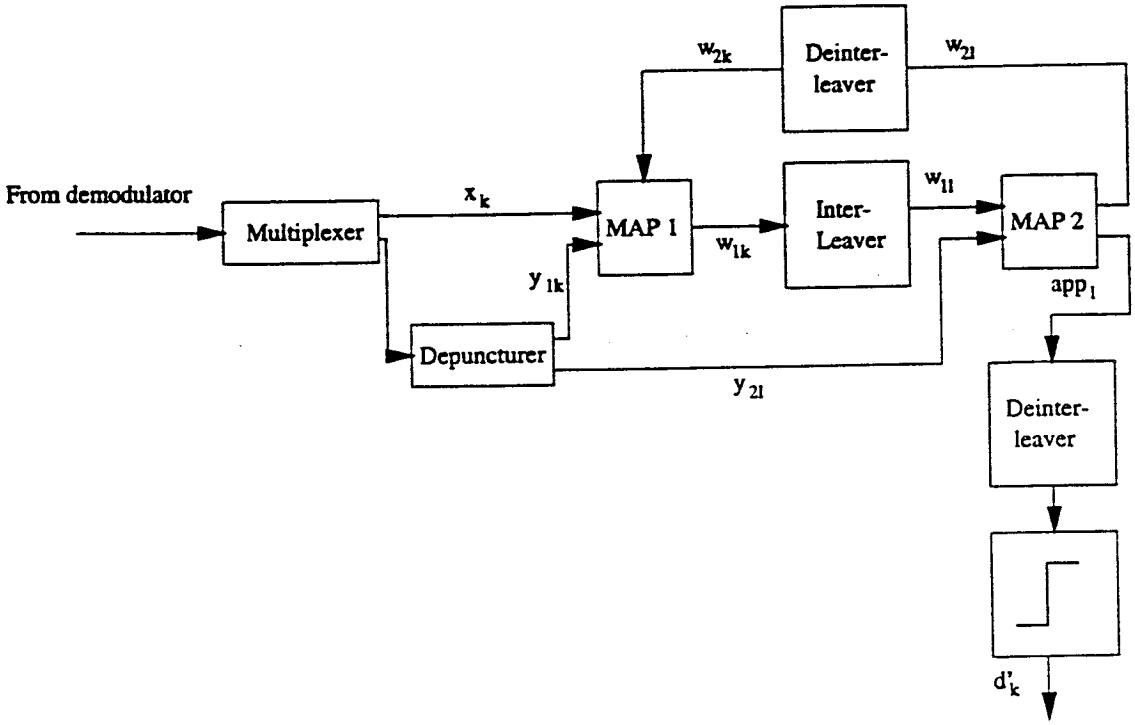



Figure 1.5: The Turbo decoder with puncturing pattern as in [1].

that, in general, driving the first encoder to the all zero state does not terminate the second encoder. This is due to the interleaver. It has become a standard approach in the literature [3] that the first encoder is terminated, and the trellis of the second encoder is simply 'left open'.

The encoder is used in conjunction with a suboptimal iterative decoding scheme based on a modified version of the maximum a posteriori (MAP) algorithm introduced by Bahl in [4], see Figure 1.5. The MAP algorithm is used because it produces reliability information about its decoding decisions.
A MAP decoder has two inputs: the channel output and the a priori probabilities for the information bits. Its output consists of the a posteriori probabilities for the information bits.
For the first iteration, the a priori probabilities $w_{2k}$ entering the first MAP decoder are set to 0.5. After interleaving, the a posteriori probabilities $w_{1k}$ produced by the first MAP decoder are fed into the second MAP decoder as a priori probabilities $w_{1l}$. The second MAP decoder, however, does not output the a posteriori probability, but a slightly modified quantity called the *extrinsic information*. The calculation of the extrinsic information is almost identical to that of the a posteriori probability. When the a posteriori probability is calculated for bit $d_k$ with the a priori probability of $d_k$ set to 0.5, the extrinsic information is obtained. The combination of the interleavers and the extrinsic information

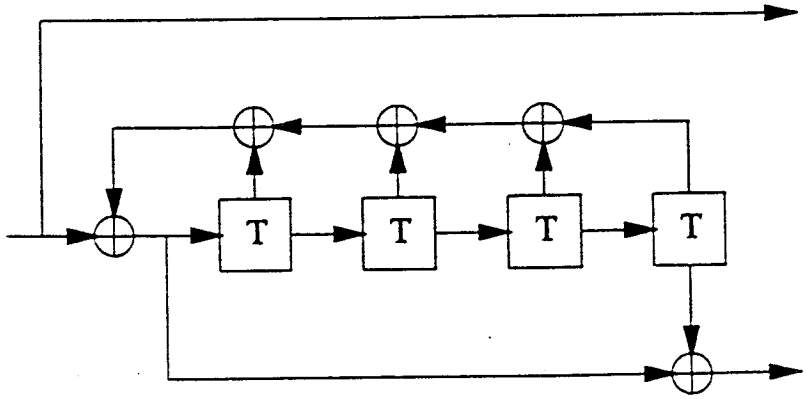



Figure 1.6: The (2,1,4) RSC encoder with generators (37,21), as proposed by Berrou in [1].

decorrelates $w_{2k}$ and $(x_k, y_k)$ so that the iterative decoding is effective. Now the extrinsic information produced by the second MAP decoder enters the first decoder as a priori probability. From the second iteration on, both decoders produce the extrinsic information. In this way, also $w_{1l}$ and $y_{2l}$ are decorrelated. After the last iteration, the second MAP decoder produces the a posteriori probabilities $app_l$ that are used to make a hard decision about the values of the information bits. In practice, the extrinsic information exchanged between the MAP decoders is in the log-likelyhood ratio form, see Section 1.4.
For the simulations in [1], eighteen iterations are used. All the simulation results in this report are obtained with eighteen decoding iterations.

Note that none of the Turbo Code's components are new. The MAP algorithm was presented already in 1972. Iterative decoding was introduced in 1954 by Elias in [17].

## 1.3 Recursive systematic convolutional encoders.

The constituent encoders of Turbo Codes are convolutional encoders in systematic feedback form. Figure 1.6 shows the constituent encoder proposed by Berrou et al. in [1]. The generator sequence is (37,21). The first number, 37, defines the feedback path, the second number, 21, defines the forward path.

## 1.4 The MAP algorithm.

The MAP algorithm, introduced by Bahl in [4], calculates the a posteriori probabilities of the states and transitions of a Markov source observed through a discrete memoryless channel. Bahl applied the algorithm to terminated feedforward convolutional codes. In [1], Berrou modified the algorithm for recursive convolutional codes. The treatment of the MAP algorithm that gives the best insight for use with Turbo Codes is published by Andersen in [5]. Only the results are stated here:

The recursive systematic convolutional encoder has $2^M$ states $S_t$ (M is the memory of the encoder), input $d_t$ and output $X_t$, $t = 0, 1, \ldots, L$. $L$ stands for the information block length. The encoder starts and ends in the zero state. The output $X_t$ is a function of the input $d_t$ and the previous state $S_{t-1}$. The source is described by the transition probabilities $Pr(S_t = m|S_{t-1} = m')$ that depend on the a priori probabilities for the information bits $d_t$. The output $X_t$ is observed through a binary input AWGN channel. The output of the channel is $Y_t$.
The following quantity is defined:

$$\begin{aligned}\gamma_t(m', m) &= Pr(S_t = m; Y_t|S_{t-1} = m') \\ &= Pr(S_t = m|S_{t-1} = m') \cdot Pr(Y_t|S_t = m; S_{t-1} = m') \\ &= Pr(d_t = i; Y_t|S_{t-1} = m') \\ &= \gamma'_t(m', i)\end{aligned}$$

$\gamma'_t(m', i)$ can be calculated using the a priori probabilities of the information bits and the probability density function $p(Y_t|X_t)$. Now the quantity $\alpha_t(m) = Pr(S_t = m; Y_1^t)$ can be calculated by the forward recursion:

$$\alpha_t(m) = \alpha_{t-1}(m'_0) \cdot \gamma'_{t-1}(m'_0, 0) + \alpha_{t-1}(m'_1) \cdot \gamma'_{t-1}(m'_1, 1)$$

using the starting conditions $\alpha_0(0) = 1$, and $\alpha_0(m) = 0$, $m = 1, 2, \ldots, 2^M - 1$. In general, $Y_a^b$ stands for the sequence $Y_a, Y_{a+1}, \ldots, Y_{b-1}, Y_b$. The state leading to $m$ with input $i$ is denoted $m'_i$.
The quantity $\beta_t(m) = Pr(Y_{t+1}^L|S_t = m)$ can be calculated as a backward recursion:

$$\beta_t(m') = \beta_{t+1}(m_0) \cdot \gamma'_{t+1}(m', 0) + \beta_{t+1}(m_1) \cdot \gamma'_{t+1}(m', 1)$$

with the boundary conditions $\beta_L(0) = 1$, and $\beta_L(m) = 0$, $m = 1, 2, \ldots, 2^M - 1$. The state obtained from $m'$ with input $i$ is denoted $m_i$.
Then the quantity $\sigma_t(m', m) = Pr(S_{t-1} = m'; S_t = m; Y_1^L)$ can be calculated as:

$$\begin{aligned}\sigma_t(m', m) &= \alpha_{t-1}(m') \cdot \gamma_t(m', m) \cdot \beta_t(m) \\ &= \alpha_{t-1}(m') \cdot \gamma'_t(m', i) \cdot \beta_t(C(m', i)) \\ &= \sigma'_t(m', i)\end{aligned}$$

where the function $C(m', i)$ gives the new state when the old state is $m'$ and the input $d_t = i$.
The log-likelyhood ratio is then defined as:

$$\begin{aligned}\Lambda(d_t) &= \log \frac{Pr(d_t = 1|observation)}{Pr(d_t = 0|observation)} \\ &= \log \frac{Pr(d_t = 1; observation)}{Pr(d_t = 0; observation)} \\ &= \log \frac{\sum_{m'} \sigma'(m', 1)}{\sum_{m'} \sigma'(m', 0)}\end{aligned}$$

# Chapter 2

# The Free Distance of TURBO Codes.

## 2.1 Introduction.

An important characteristic of every code is its free distance. For large enough signal to noise ratios, the free distance will dominate the code's performance. Therefore, it makes sense to start the analysis of Turbo Codes by determining their free distance.
The project description asks to find the free distance of 'the' Turbo Code, thereby refering to the code that was introduced by Berrou et al. in [1]. However, this paper defines 'the' Turbo Code in an ambiguous manner. The exact form of the interleaver and the way the constituent encoders are terminated are not mentioned. As can easily be shown, both have an effect on the free distance of the code.
The interleaver used to obtain the simulation results in [1] is a pseudorandom interleaver. This means that it is generated randomly and then used unaltered throughout the simulation. An example of how the interleaver influences the free distance will be given later in this chapter.
It is not that obvious that the termination of the constituent encoders influences the free distance. Consider a Turbo Code with a rectangular interleaver and the constituent encoders chosen as in [1]. 'Rectangular' means that the information sequence is written linewise and read columnwise, as is the case for interleavers used to decorrelate decoding errors in serial coding schemes. In the event that none of the encoders is terminated, the free distance cannot be bigger than two. The information sequence consisting of all zeros and a single one in the last bit position is interleaved to itself. Thus, only in the last time step of the encoding will this single one cause a deviation from the all zero state in the trellises of both encoders, which results in a weight two codeword. Now, assume that the last four bits of the information sequence are used to terminate the first encoder. Then, the codeword weight cannot be smaller than the free distance of the first constituent encoder, which equals four after puncturing. Thus, the termination influences the free distance of a Turbo Code.
For the reasons mentioned above, it is not possible to find the free distance of 'the' Turbo Code as mentioned in task two of the project description. Nevertheless, in this chapter, an

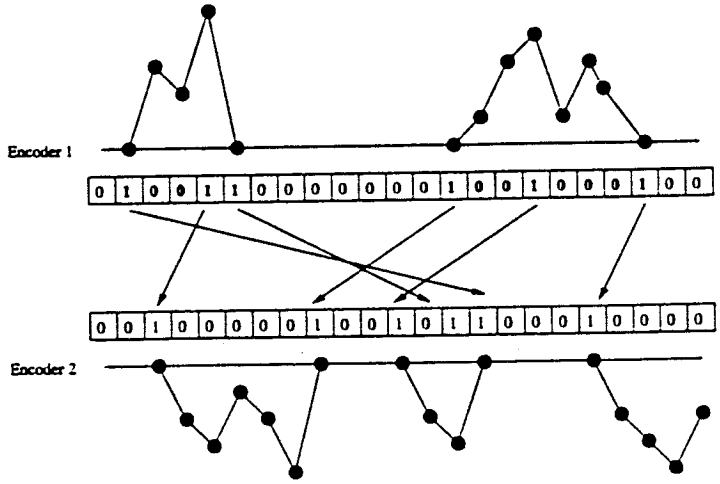



Figure 2.1: The creation of a codeword. The information sequences causing the completed detours in the first encoder are printed bold.

algorithm to find the free distance will be introduced. This algorithm assumes termination of the first encoder as was proposed by Robertson in [3]. The algorithm will be applied to two examples to demonstrate the influence of the interleaver choice on the free distance. The results will also show how the error floor is caused by the free distance.
At the end of this chapter, the problem of interleaver design for maximizing the free distance will be formalized and mathematically defined. The approach will be to develop a parity-check matrix for a Turbo Code. This approach shows the influence of the interleaver on the basis vectors of the code.
Finally, in the conclusion, a comparison will be made between the work on the free distance presented in this report and the results of Robertson in [3].

## 2.2 An algorithm for determining the free distance.

### 2.2.1 Theory.

An important characteristic of the Turbo Code is its block length N. The most straightforward way to find the free distance of this code is to encode all possible information blocks and record the smallest codeword Hamming weight. For any practical values of N, however, the number of information blocks, $2^N$, is just too high to make this calculation. It is the goal of this algorithm to focus only on those information blocks that could possibly cause the free distance. The number of information blocks that have to be encoded to find the free distance will be drastically reduced. The algorithm only requires an upper bound on the free distance and the termination of the first constituent encoder.

First, an example is presented that shows how a codeword of a Turbo Code is formed. This example leads to a better understanding of the algorithm that follows. In Figure 2.1, an information block entering the first constituent encoder is shown. Since the first encoder

must be terminated, this information sequence causes a number of completed detours in the trellis of the first encoder. A 'completed detour' is defined as a path that starts and ends in the zero state, but does not return to the zero state between its start - and endpoint. This information block is interleaved before it enters the second constituent encoder. In general, the interleaved information block will not terminate the second encoder. Thus, in the trellis of the second encoder, it will cause a number of completed detours, but also an incompleted detour at the end of the trellis. An 'incompleted detour' is defined as a path that starts in the zero state, but never returns to the zero state. An incompleted detour is finite because it stops at the end of the trellis. It is therefore possible that the free distance is caused by multiple detours in both constituent encoders, including an incompleted detour in the second encoder.
Every detour in the trellis of an encoder is caused by a unique information sequence that has the same length as the detour. In the example above, there are two completed detours in the trellis of the first encoder. Therefore the information block has to contain the information sequences that cause these detours. Before, between, and after these sequences, the information block can contain only zeros since the encoder stays in the zero state at these locations. One could say that these sequences are 'embedded in zeros'. By 'embedded in zeros' it is meant that the bit positions before the first sequence, between two consecutive sequences, and after the last sequence, all contain a zero bit.

Consider an information block that terminates the first encoder: it causes $n$ completed detours in the trellis of the encoder. This information block has to contain the $n$ sequences that cause these detours. These $n$ sequences are embedded in zeros.
Consider $n$ information sequences that each cause one completed detour in the trellis of the first encoder. An information block that consists out of these $n$ sequences embedded in zeros, causes the corresponding detours in the trellis of the first encoder. Since all these detours are completed, the information block terminates the first encoder. This implies the following result:

**Lemma 1** *Given is a Turbo Code with block length $N$ and a terminated first constituent encoder. Let $S$ be the set of all information sequences that cause a completed detour, with length smaller than or equal to $N$, in the trellis of the first constituent encoder. Every information block terminating the first constituent encoder consists of elements of $S$ embedded in zeros. But also: Every information block of length $N$ that consists of elements of $S$ embedded in zeros, terminates the first constituent encoder.*

This lemma gives a method to construct all information blocks that terminate the first constituent encoder of a Turbo Code. This task is equivalent to determining all information blocks that consist of elements of S embedded in zeros. Notice that a certain element of S may appear more than once in such an information block. The sum of the lengths of the elements of S that such an information block contains, has to be smaller than the blocklength $N$. The combined Hamming weight that such an information block causes in the first encoder, is the sum of the weights of the detours that it causes.
To determine all the information blocks terminating the first encoder, the following steps are taken:

- Determine the set S of all information sequences that cause a single completed detour in the trellis of the first encoder and whose length is smaller than or equal to $N$.

- Determine all different combinations of elements of S such that a combination may contain a certain element more than once, and the sum of the lengths of the sequences that a combination contains, is smaller than or equal to $N$.

- For each combination found in the previous step, determine all information blocks that contain the sequences of the combination by embedding these sequences in zeros.

This lemma is now used to focus only on those information blocks that could possibly cause the free distance. Therefore an upper bound on the free distance is needed. Such an upper bound can be obtained in three ways: a guess, a good guess relying on simulation results, or an analytically derived bound.
Only the information blocks that cause an output Hamming weight in the first encoder that is smaller than or equal to the bound, can cause the free distance of the Turbo Code. Thus, only those information blocks have to be considered that consist of elements of S such that the sum of the weights of the detours that correspond to these elements, is smaller than or equal to the bound. This requirement immediately eliminates all elements of S causing a weight bigger than the bound.
The total output weight caused by an information block is given by the sum of the weights caused in each constituent encoder. When an information block is assembled using the method above, the output weight it causes in the first encoder, is the combined output weight of its detours. The output weight for this information block in the second encoder can be determined by simply encoding it with the second encoder.
These considerations lead to the following algorithm:

1. Determine an upper bound on the free distance.

2. In the trellis of the first constituent encoder search all information sequences such that each sequence generates one completed detour with output weight smaller than or equal to the bound, and length not bigger than the blocklength N.

3. Determine all different combinations of sequences found in step 2. A combination may contain a certain sequence more than once. The sum of the lengths of the sequences that a combination contains, is smaller than or equal to $N$. The sum of the output weights that the sequences of a combination cause, is smaller than or equal to the bound on the free distance.

4. For each combination found in step 3, assemble all possible information blocks of length N that contain the sequences of the combination by embedding these sequences in zeros.

5. Interleave all information blocks from step 4, encode the resulting sequences with the second constituent encoder, and record the additional output weight in the second encoder. If the total output weight is smaller than the bound, the bound takes on this new value and the process continues.

6. The final value of the bound is the free distance of the Turbo Code.

The complexity of the algorithm strongly depends on the upper bound for the free distance. The number of sequences, found in step 2, increases exponentially with this bound and so does the number of information blocks to consider in step 4. The number of possible information blocks further depends on the block size N and the puncturing period, but only in a polynomial fashion.

Note that when the output of a constituent encoder is punctured, a detour can cause different output weights depending on its starting position in the trellis. The starting position of a detour in the trellis determines how its output sequence is punctured. A sequence may therefore appear multiple times in S with a different output weight. Thus when information blocks are assembled also the puncturing has to be taken into account to determine the output weight this block produces in the first encoder.

### 2.2.2 Implementation.

The algorithm is implemented in two ANSI-C programs:
The program 'pathfind' calculates all the sequences causing completed detours in the trellis of the first constituent encoder, with output Hamming weight smaller than or equal to the upper bound on the free distance. Also the puncturing is taken into account. The program requires the following input:

- The generators of the first constituent encoder.
- The puncturing pattern.
- The upper bound on the free distance.
- An upper bound on the input weight, which proved useful in the case of a rectangular interleaver, see Section 2.4.
- Name of the output file.

The following output is produced:

- Status files.
- An output file containing a list of the found sequences, including:
  - Input weight.
  - Output weight.
  - Beginning of the sequence relative to the puncturing period.
  - The location of the ones in the sequence.

The output of 'pathfind' is then used to calculate the free distance in the program 'freedist'. 'freedist' assembles all possible information blocks and encodes them in the second encoder after interleaving. In order to function efficiently, it starts encoding the interleaved block at a depth in the trellis corresponding to the location of the first one. Up to that point, no weight has been accumulated. The encoding lasts until the sum of the weight caused in the first encoder and the accumulated weight in the second encoder exceeds the bound, or

until the end of the trellis is reached. In the second case a codeword with weight smaller than the bound has been found.
'freedist' requires the following input:

- The file produced by 'pathfind' containing the sequences for the first encoder.
- The upper bound on the free distance.
- The generators for the second encoder.
- The puncturing pattern.
- The interleaver.
- The name of the output file.

The output of 'freedist' is a file containing the free distance and the information blocks that lead to free distance codewords.
As mentioned in Section 2.2.1, the complexity of the algorithm primarily depends on the upper bound for the free distance. This is reflected in the calculation time for both programs and the memory requirement to store the sequences found by 'pathfind'. Therefore, if the expected free distance becomes too large, the algorithm becomes unpractical.

## 2.3 The free distance for a pseudorandom interleaver.

### 2.3.1 Problem description.

In this section, the free distance algorithm is applied on a Turbo Code with a 64K pseudorandom interleaver. The component encoders and the puncturing pattern are chosen as in [1]. The first encoder is terminated using the last four bits of the information block.

### 2.3.2 Application of the algorithm.

The free distance of the unpunctured constituent encoder is six and caused by the information sequence 100001. When the output is punctured, the same sequence generates the free distance of the punctured first encoder: four. Therefore, the free distance of the Turbo Code has to be at least four. This weight is also caused in the first encoder by the sequences 10000000001 and 1100011 when these sequences start at odd locations in the trellis. This is illustrated in Table 2.1: The output weight that these input sequences produce depends on the constituent encoder and the parity of the position where they begin in the trellis. Note that the free distance of this Turbo Code could be as small as four: the output weight of the input sequence 1100011 can be completely punctured away in the second encoder.
For these reasons the expected value for the free distance was first set to four and then increased to five and six.

| Input sequence | Output sequence | | Encoder 1 | | | | Encoder 2 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | even | | odd | | odd | | even | |
| 1 | 1 | 1 | 1 | 1 | 1 | ✗ | ✗ | 1 | ✗ | ✗ |
| 0 | 0 | 1 | 0 | ✗ | 0 | 1 | ∅ | ✗ | ∅ | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | ∅ | ∅ | 0 | ∅ | ∅ |
| 0 | 0 | 0 | 0 | ∅ | 0 | 0 | ∅ | ∅ | ∅ | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | ✗ | ∅ | 1 | ∅ | ✗ |
| 1 | 1 | 1 | 1 | ✗ | 1 | 1 | ✗ | ✗ | ✗ | 1 |

| Input sequence | Output sequence | | Encoder 1 | | | | Encoder 2 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | even | | odd | | odd | | even | |
| 1 | 1 | 1 | 1 | 1 | 1 | ✗ | ✗ | 1 | ✗ | ✗ |
| 0 | 0 | 1 | 0 | ✗ | 0 | 1 | ∅ | ✗ | ∅ | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | ∅ | ∅ | 0 | ∅ | ∅ |
| 0 | 0 | 0 | 0 | ∅ | 0 | 0 | ∅ | ∅ | ∅ | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | ✗ | ∅ | 1 | ∅ | ✗ |
| 0 | 0 | 0 | 0 | ∅ | 0 | 0 | ∅ | ∅ | ∅ | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | ✗ | ∅ | 1 | ∅ | ✗ |
| 0 | 0 | 0 | 0 | ∅ | 0 | 0 | ∅ | ∅ | ∅ | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | ∅ | ∅ | 0 | ∅ | ∅ |
| 0 | 0 | 1 | 0 | ✗ | 0 | 1 | ∅ | ✗ | ∅ | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | ✗ | ✗ | 1 | ✗ | ✗ |

| Input sequence | Output sequence | | Encoder 1 | | | | Encoder 2 | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | even | | odd | | odd | | even | |
| 1 | 1 | 1 | 1 | 1 | 1 | ✗ | ✗ | 1 | ✗ | ✗ |
| 1 | 1 | 0 | 1 | ∅ | 1 | 0 | ✗ | ∅ | ✗ | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | ✗ | ∅ | 1 | ∅ | ✗ |
| 0 | 0 | 0 | 0 | ∅ | 0 | 0 | ∅ | ∅ | ∅ | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | ✗ | ∅ | 1 | ∅ | ✗ |
| 1 | 1 | 0 | 1 | ∅ | 1 | 0 | ✗ | ∅ | ✗ | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | ✗ | ✗ | 1 | ✗ | ✗ |

Table 2.1: This table shows the systematic and parity output sequences of a (37,21) RSC encoder for three input sequences. For a Turbo Code with puncturing pattern as defined in [1], the effective output sequence depends on the constituent encoder and the begin position (even or odd) of the input sequence in the information block: the erased bits are punctured.

### 2.3.3 Results.

The free distance was determined to be six, and there are three codewords of this weight. Their information blocks are depicted in Figure 2.2. They all consist of an information sequence of weight two interleaved to a similar sequence.

In Figure 2.3, simulation results for this Turbo Code are shown, together with the free distance asymptote.

This free distance asymptote is the free distance term of the union bound for block codes and a binary input AWGN channel:

$$P_b \leq \sum_{d=d_{free}}^{\infty} \sum_{w=1}^{N} a(w,d) \cdot \frac{w}{N} \cdot \frac{1}{2} \cdot \text{erfc}\left(\sqrt{d \cdot \frac{R \cdot E_b}{N_o}}\right) \tag{2.1}$$

The symbols in this formula stand for:

$P_b$ The bit error rate.

$d$ The codeword weight.

$w$ The information weight.

$a(w,d)$ The number of codewords with information weight $w$ and codeword weight $d$.

$N$ The blocksize or, equivalently, the interleaver size.

$R$ The rate of the code.

$E_b/N_o$ The signal to noise ratio per information bit.

For the free distance asymptote:

$$3 \cdot \frac{2}{65536} \cdot \frac{1}{2} \cdot \text{erfc}\left(\sqrt{6 \cdot \frac{\frac{1}{2} \cdot E_b}{N_o}}\right)$$

Note that, see Figure 2.3, the free distance asymptote is responsible for the error floor of this Turbo Code. The reason for the flatness of the error floor is that it is caused by a small free distance. For higher signal to noise ratios, however, the 'floor' will disappear because the bit error rate decreases exponentially as a function of the signal to noise ratio. It is important for the performance of this Turbo Code that the number of the free distance codewords be small. This leads to a small effective multiplicity:

$$\sum_{w=1}^{N} a(w,d) \cdot \frac{w}{N}$$

of the free distance term of the union bound. The reason why the free distance asymptote determines the performance at low signal to noise ratios will de discussed in Chapter 3.

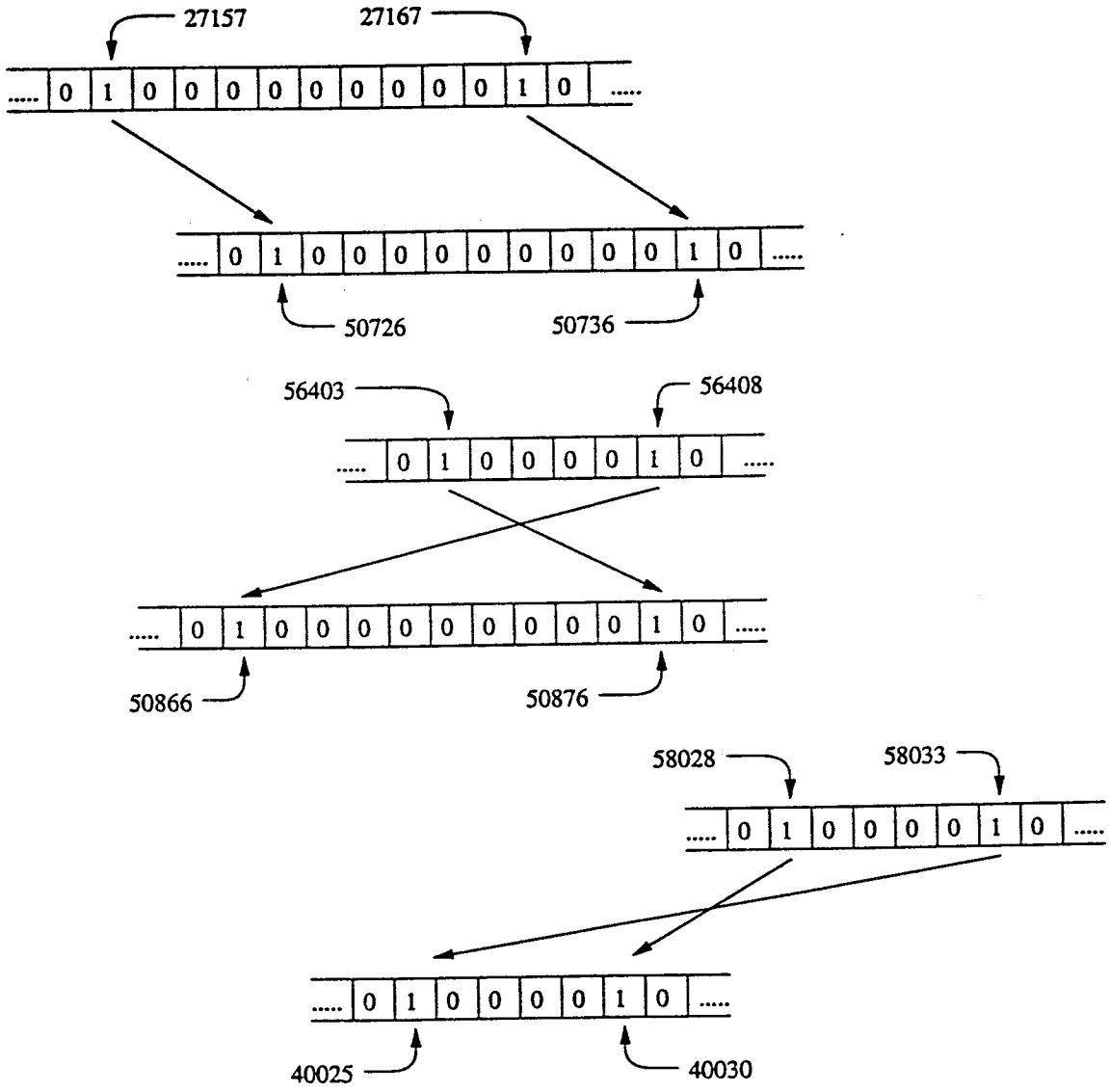



Figure 2.2: The creation of the three free distance codewords. The information blocks entering the constituent encoders are depicted. The arrows between the information blocks symbolize the interleaving. The numbers are the positions in the information block. The codeword Hamming weight that these information sequences produce can be checked with Table 2.1: it equals six for each codeword.

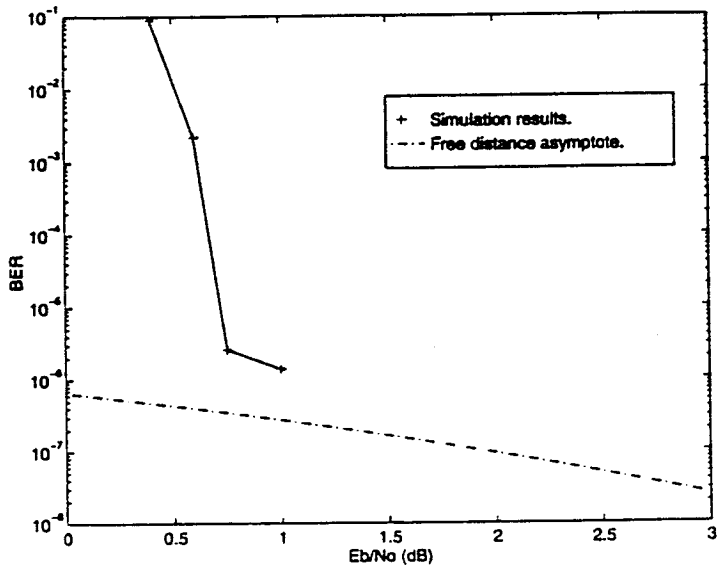



Figure 2.3: The performance of a Turbo Code with 64K pseudorandom interleaver: simulation results and the free distance asymptote.

## 2.4 The free distance for a rectangular interleaver.

### 2.4.1 Problem description.

In this section, the free distance algorithm is applied to a Turbo Code with a $120 \times 120$ rectangular interleaver. The information block is hence 14400 bits long. The information bits are written linewise in the interleaver and read columnwise. The component encoders and the puncturing pattern are chosen as in [1]. The first encoder is terminated using the last four bits of the information block.

### 2.4.2 Application of the algorithm.

Because of the regular structure of this interleaver, it is possible to find an analytical bound on the free distance. The information pattern in Figure 2.4 generates a codeword of weight twelve. This can be verified using Table 2.1. The two horizontal 100001 sequences each cause outputs of weight four in the first encoder, while the two vertical 100001 sequences each cause outputs of weight two in the second encoder.
An upper bound of twelve on the free distance leads to 43819 sequences in the trellis of the first encoder that cause completed detours. However, the number of these detours to consider for the algorithm can still be decreased, based on the following facts:

- Of all the 43819 sequences for the first encoder, the shortest has a length of 5, the longest a length of 53.
- For the second encoder the longest possible detour with weight smaller than or equal twelve, has a length equal to 117. Note that for the second encoder both completed

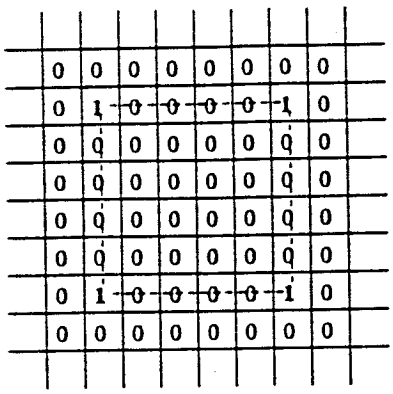



Figure 2.4: This input pattern causes a codeword of weight twelve for this Turbo Code with rectangular interleaver.

and incompleted detours must be considered.

- The interleaver structure and size.
- From the trellis of the encoders, it can be seen that every information sequence causing a completed detour, starts and ends with a one.

In Figure 2.4, the ones of the two vertical sequences for the second encoder overlap with the ones of the two horizontal sequences going into the first encoder. In this way, a codeword is formed with weight twelve.
Now, because the length of the longest sequence to consider is 117 and this is smaller than the interleaver width of 120, it is impossible for the ones of a single sequence for the first encoder to overlap with the ones of a single sequence for the second encoder. The ones of any sequence for the first encoder are at least separated by 5 positions. After interleaving, they are separated by at least $5 \times 120$ positions, and therefore no sequence for the second encoder of length 117 or less can have overlapping ones with the sequence in the first encoder. As a matter of fact, one can prove that one needs at least two sequences for both encoders to form a pattern such that the ones overlap. Therefore, only information blocks that contain two or more sequences for the first encoder can cause free distance codewords.
Because the bound equals twelve and the free distance is four for the punctured first encoder, only completed detours with weight up to eight have to be considered for the free distance algorithm. Considering a detour of weight nine already leads to codewords of weight thirteen, since there are a minimum of two detours, and the second detour causes at least weight four.
This reasoning reduces the number of sequences to 274.

### 2.4.3 Results.

The free distance of this code was found to be twelve. It is caused by the patterns shown in Figure 2.5.

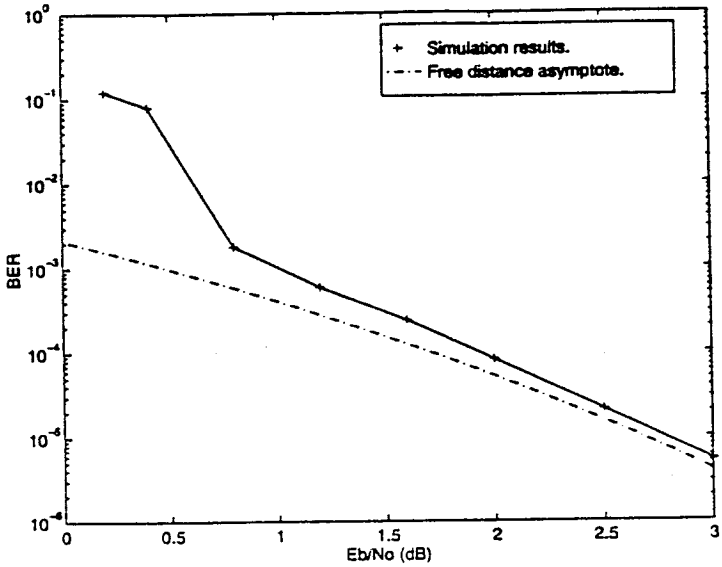



Figure 2.6: The performance of a Turbo Code with 120 × 120 rectangular interleaver: simulation results and the free distance asymptote.

It is remarkable that the number of free distance codewords is very high. It is equal to the amount of times that these patterns fit into the interleaver. The first pattern, for example, fits $(\sqrt{N}-5)\cdot(\sqrt{N}-5)$ times, causing an equal amount of codewords of weight twelve. Due to the puncturing of 10000000001, which needs to be odd to cause weight two in the second encoder, the second pattern fits $(\sqrt{N}-5)\cdot(\frac{\sqrt{N}-10}{2})$ times. After similar considerations for the other patterns,the free distance asymptote is given by:

$$\left((\sqrt{N}-5)^2+2\cdot(\frac{\sqrt{N}-10}{2})\cdot(\sqrt{N}-5)+(\frac{\sqrt{N}-10}{2})^2\right)\cdot\frac{4}{N}\cdot\frac{1}{2}\cdot\text{erfc}\left(\sqrt{12\cdot\frac{\frac{1}{2}\cdot E_b}{N_o}}\right)$$

For $N = 14400$ this becomes:

$$28900\cdot\frac{4}{14400}\cdot\frac{1}{2}\cdot\text{erfc}\left(\sqrt{12\cdot\frac{\frac{1}{2}\cdot E_b}{N_o}}\right)$$

This is the free distance asymptote drawn in Figure 2.6.
The reason why the free distance asymptote was first derived for $N$ in general is because it can now be shown that it remains the free distance asymptote as the interleaver size increases. This follows from the fact that the bound on the free distance does not change for a larger interleaver: the pattern in Figure 2.4 still gives weight twelve. This means that the sequences to consider for the algorithm remain the same. The size of the 120 × 120 interleaver is so large that every pattern formed in a larger interleaver can also be assembled in this interleaver. Therefore no codeword with weight smaller than twelve can be formed

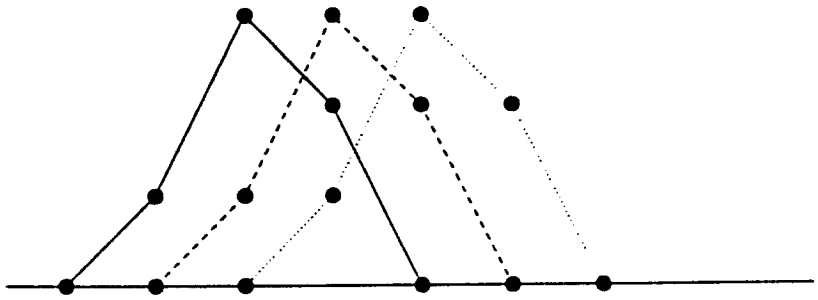


Figure 2.7: At every point of the trellis of a convolutional code, free distance paths start.

in the larger interleaver as it can not be done in the 120 × 120 interleaver. As the size of the interleaver increases, the free distance asymptote rises and converges toward the following curve:

$$4.5 \cdot \text{erfc}\left(\sqrt{12 \cdot \frac{\frac{1}{2} \cdot E_b}{N_o}}\right)$$

Comparing to the previous example of a Turbo Code with a 64K pseudorandom interleaver, the performance for this Turbo Code is worse at moderate signal to noise ratios. At first this might appear strange because the Turbo Code with rectangular interleaver has a free distance that is twice that of the Turbo Code with pseudorandom interleaver. However, the number of the free distance codewords is 28900 with the rectangular interleaver versus only 3 for the pseudorandom case. It is also interesting that the error floor is not as flat as that of the pseudorandom interleaver, as shown in Figure 2.6. The slope of the error floor is determined only by the free distance, not by the number of free distance codewords. The higher the free distance, the steeper the slope.

## 2.5 Turbo Code versus convolutional code.

In the previous example of a Turbo Code with a rectangular interleaver, the number of free distance codewords increases proportionally to the interleaver size $N$. This results in a bad performance. Something similar happens when a convolutional code is terminated and treated like a block code. At every point in the trellis, free distance paths start. This is illustrated in Figure 2.7. Thus, the number of free distance codewords increases proportionally to the block size.
The Turbo Code with the 64K pseudorandom interleaver, however, has only three codewords at the free distance. This effect can not be reached with a convolutional code. The only way to improve the asymptotic performance of a convolutional code is by increasing the free distance itself, or decreasing the number of free distance paths starting at a *given* point in the trellis. This leads to using complexer convolutional codes with more memory.

## 2.6 The parity-check matrix of a Turbo Code.

### 2.6.1 Intentions.

In Section 2.2.1, a method has been developed to determine the free distance of a Turbo Code. The creation of codewords has been explained in terms of detours in the trellises of both constituent encoders. A free distance codeword is formed when an information sequence causing a low weight path in the first encoder is interleaved to a similar sequence for the second encoder. No insight has been given, however, in how to design an interleaver for maximizing the free distance and for obtaining a minimum number of free distance codewords. An analysis will be presented here that defines the problem mathematically, giving better insight into the role of the interleaver, and showing the complexity of interleaver optimization.
The approach consists in developing the parity check matrix of a particular Turbo Code.

### 2.6.2 The parity-check matrix.

In [16], the following method is given to construct a reduced parity-check matrix (the names of the symbols are changed to avoid confusion):

**Theorem 1** *Construction of a Reduced Parity-Check Matrix: A reduced parity-check matrix $H$ for an $(Z, N)$ $q$-ary linear code with $1 \leq N < Z$ having an encoding matrix $G$ (whose rows we denote here by $v_1, v_2, \ldots, v_N$ ) can be constructed as follows:*

1. *Choose $v_i$ as any vector in $F^N \setminus S(v_1, v_2, \ldots, v_{i-1})$ for $i = N+1, N+2, \ldots, Z$.*
2. *Form the $Z \times Z$ matrix $M$ whose rows are $v_1, v_2, \ldots, v_Z$.*
   *Then compute the inverse matrix $M^{-1}$.*
3. *Take $H^T$ as the last $Z - N$ columns of the matrix $M^{-1}$.*

Here $F$ stands for $GF(q)$, and $S(v_1, v_2, \ldots, v_{i-1})$ for the vector space spanned by $v_1, v_2, \ldots, v_{i-1}$.
The free distance is then equal to the smallest positive integer $d$ such that there are $d$ rows of $H^T$ that are linearly dependent.

### 2.6.3 The parity-check matrix of a Turbo Code.

Consider a Turbo Code with constituent encoders with generators $(37, 21)$, as defined in [1]. None of the encoders is terminated. Only the systematic output of the second encoder is punctured, making this a rate 1/3 code. At first no interleaver is used, but the block length equals $N$.
The first step in determining the parity-check matrix is to find an encoding matrix G. Therefore the $N$ linearly independent information blocks $00\ldots1_i\ldots0$ with $i = 0, 1, \ldots, N-1$ are encoded. These sequences contain only a single one at location $i$. The resulting codewords are linearly independent and form a basis for the vector space of the codewords. They are used as rows for the encoding matrix. The dimension of the encoding matrix is therefore $N \times 3N$. After reorganizing the columns of the output of the first encoder,

such that the systematic part comes first, followed by the parity part, the encoding matrix looks as follows:

$$( I_{N\times N} \quad P_{N\times N} \mid P_{N\times N} )$$

Here $P_{N\times N}$ stands for the parity output of the encoders and $I_{N\times N}$ for the systematic output. $I_{N\times N}$ is the identity matrix of dimension $N \times N$, $P_{N\times N}$ is given by:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & \dots \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & \dots \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & \dots \\ \vdots & & & & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Every row of $P_{N\times N}$ consists of the previous row, shifted one position to the right, and completed with zeros at the front. The encoders are time-invariant.
One possible choice for the matrix $M$ is:

$$\begin{pmatrix} I_{N\times N} & P_{N\times N} & P_{N\times N} \\ O_{N\times N} & I_{N\times N} & O_{N\times N} \\ O_{N\times N} & O_{N\times N} & I_{N\times N} \end{pmatrix}$$

$O_{N\times N}$ is the zero matrix of dimension $N \times N$.
It is easy to verify that $M \times M = I_{3N\times 3N}$, or that $M^{-1} = M$. Therefore $H^T$ is given by:

$$\begin{pmatrix} P_{N\times N} & P_{N\times N} \\ I_{N\times N} & O_{N\times N} \\ O_{N\times N} & I_{N\times N} \end{pmatrix}$$

Now consider the case where an interleaver is used. Every information block $00\dots1_i\dots0$ with $i = 0, 1, \dots, N-1$ would be interleaved to $00\dots1_{G(i)}\dots0$ where $G(i)$ is a bijection from $\{0, 1, \dots, N-1\}$ onto itself, and symbolises the function of the interleaver. This results in a permutation of the rows of $P_{N\times N}$ for the second encoder. $H^T(G)$ is then equal to

$$\begin{pmatrix} P_{N\times N} & P_{N\times N}(G) \\ I_{N\times N} & O_{N\times N} \\ O_{N\times N} & I_{N\times N} \end{pmatrix}$$

where $P_{N\times N}(G)$ symbolises the dependence on the interleaver.
This analysis shows that:

- Maximizing the free distance for this Turbo Code with a fixed interleaversize is now equivalent to finding a permutation $G$ such that the smallest number of rows of $H^T(G)$ that are linearly dependent is maximized.
- This analysis shows the function of the interleaver on the lowest level: it relates the basis vectors of the constituent encoders.
- For practical block lengths, the number of possible permutations, $N!$, is too large to find the optimal interleaver. Therefore a systematic way is needed to optimize the interleaver for the free distance.

## 2.7 Conclusion.

In this chapter, the free distance problem of Turbo Codes was addressed. The following is achieved:

- In the introduction it has been shown that 'the' Turbo Code is not uniquely defined: both the interleaver and the termination of the constituent encoders have an influence on the free distance. In [1], Berrou et al. have introduced a class of codes, not one specific code. This means that task two of the project description, 'Find the free distance of the TURBO code', cannot be solved literally.

- An algorithm for calculating the free distance of Turbo Codes was developed and implemented.

- This algorithm was then applied to a Turbo Code with a 64K pseudorandom interleaver. This led to an explanation of the error floor. The error floor is caused by the free distance asymptote. It looks so flat because of the low free distance of the code. However, due to the low multiplicity of the free distance codewords relative to the interleaver size, the effective multiplicity, $\sum_{w=1}^{N} a(w,d) \cdot \frac{w}{N}$ , is very small so that the performance is good.

- The algorithm was also applied to a Turbo Code with a $120 \times 120$ rectangular interleaver. This code has a bigger free distance, but also a very large number of free distance codewords. It was proven that as the interleaver increases, the free distance remains twelve, but the performance of the code does not improve since the number of free distance codewords grows proportional to the interleaver size $N$. For this Turbo code, the error floor is steeper than in the previous case due to the bigger free distance.

- The free distance problem of a Turbo Code has been compared to that of a terminated convolutional code. It was shown that the effect of a low number of free distance codewords cannot be obtained with a terminated convolutional code.

- In the last section a different approach was taken. Using the parity-check matrix for a Turbo code, the problem of optimizing an interleaver for free distance was formalized. This illustrated the function of the interleaver and the complexity of the problem. A systematic method is needed to optimize the interleaver for the free distance.

The free distance problem of Turbo Codes with pseudorandom interleavers has already been investigated by Robertson in [3]. Robertson has found the free distance of these codes and concludes:

> It has been observed that the event leading to the minimum distance of 6 is quite rare, it typically occurs just a few times for any particular interleaver.

Therefore, part of the work described above is a verification of Robertson's work. However, the free distance algorithm, the free distance of a rectangular interleaver, and the parity-check matrix is not mentioned in [3].

# Chapter 3

# The Distance Spectrum of TURBO Codes.

## 3.1 Introduction.

This chapter focuses on the distance spectrum of Turbo Codes. The distance spectrum is defined as $\sum_w w \cdot a(w,d)$, which is a function of $d$. See Section 2.3 for the notation. It is the goal of this chapter to reveal how the distance spectrum changes as the interleaver size increases and to show how this change influences the code's performance. As mentioned in Section 1.1, two major characteristics describe the performance of Turbo Codes: As the interleaver size increases, the error floor is shifted down, while the point at which the error floor stops dominating the code's performance is shifted to the left. Both of these properties can be explained by considering the distance spectrum of Turbo Codes.
A first analysis of the spectrum is made in Section 3.2. The influence of the interleaver on the distance spectrum becomes clear, and the ideal interleaver is defined.
In [7], Benedetto et al. introduce the concept of random interleaving to calculate an upper bound on the bit error probability of Turbo Codes with given constituent encoders and block length. The principles of random interleaving are discussed in Section 3.3. In Section 3.4, an improvement for random interleaving is proposed and random interleaving is extended to punctured Turbo Codes.
The effect of increasing the interleaver size on the distance spectrum is investigated in Section 3.5 using the concept of random interleaving. Some theoretical results are obtained for use later in the chapter.
The results of Section 3.5 are applied in Section 3.6 to explain the behavior of the error floor. Expected values for the free distance and the number of free distance codewords are derived. The results show that there exist complex block codes with a low number of free distance codewords.
Finally, in Section 3.7, theoretical and empirical evidence is presented to support the concept of spectral thinning for Turbo Codes. A theory is proposed that relates the distance spectrum to the code's performance. The theory predicts that the performance of Turbo Codes approaches the Shannon limit as the interleaver size, and thus the codeword length, increases.

## 3.2 A first analysis: an ideal interleaver.

### 3.2.1 Intention.

It is the intention of this section to perform a first analysis for the distance spectrum of a Turbo Code. For a constant block length, the influence of the interleaver on the distance spectrum will be shown, and compared to the case where no interleaver is used.
In Section 2.2.1, the creation of a typical codeword is shown: An information block causes detours in the first encoder which generates the output weight for the first encoder, independent of the interleaver. The information block is then interleaved and encoded by the second encoder. However, the weight generated by the second encoder depends very strongly on the interleaver. It is intuitively clear that in order to obtain good performance, an information block causing a low weight output in the first encoder, should be interleaved so that it produces a large weight in the second encoder.
In this section, a limit will be introduced on how much the interleaver can improve the performance of the code for a constant block length $N$.

### 3.2.2 A necessary proof.

Before continuing, the following result will be derived:

**Lemma 2** *Assume:*

- *An ordered sequence:* $d_1 \leq d_2 \leq \ldots \leq d_i \leq \ldots \leq d_M$ *with* $i = 1, 2, \ldots, M$.
- *Let* $G$ *be a bijection of* $\{1, 2, \ldots, M\}$ *onto itself.*
- *Let* $D_i(G) = d_i + d_{G(i)}$ *with* $i = 1, 2, \ldots, M$.
- *Define* $D_{\text{Avg}}(G) = \frac{\sum_{i=1}^{M} D_i(G)}{M}$, *which is the average value of* $D_i(G)$.
- *Define* $D_{\text{Var}}(G) = \frac{\sum_{i=1}^{M} (D_{\text{Avg}}(G) - D_i(G))^2}{M}$, *which is the variance of* $D_i(G)$.

*Then:*

1. $D_{\text{Avg}}(G)$ *is independent of* $G$.
2. *The bijection* $G(i) = i$, $\forall i$, *maximizes* $D_{\text{Var}}(G)$.
3. *The bijection* $G(i) = M - i + 1, \forall i$, *minimizes* $D_{\text{Var}}(G)$.

The first item is simply proven:

$$D_{\text{Avg}}(G) = \frac{\sum_{i=1}^{M} D_i(G)}{M} = \frac{\sum_{i=1}^{M} (d_i + d_{G(i)})}{M} = 2 \cdot \frac{\sum_{i=1}^{M} d_i}{M} = D_{\text{Avg}}$$

For the second item:

$$D_{\text{Var}}(i) - D_{\text{Var}}(G) = \frac{\sum_{i=1}^{M} (D_{\text{Avg}} - D_i(i))^2}{M} - \frac{\sum_{i=1}^{M} (D_{\text{Avg}} - D_i(G))^2}{M}$$

C-2

$$
\begin{aligned}
&= \frac{2}{M} \cdot \left( \sum_{i=1}^{M} d_i^2 - \sum_{i=1}^{M} d_i \cdot d_{G(i)} \right) \\
&= \frac{1}{M} \sum_{i=1}^{M} (d_i - d_{G(i)})^2 \\
&\geq 0
\end{aligned}
$$

There is equality if and only if $d_i = d_{G(i)}$, $\forall i$, in particular for $G(i) = i, \forall i$.

For the third item:

$$
\begin{aligned}
& D_{\text{Var}}(G) - D_{\text{Var}}(M - i + 1) && (3.1) \\
= \; & \frac{\sum_{i=1}^{M} (D_{\text{Avg}} - D_i(G))^2}{M} - \frac{\sum_{i=1}^{M} (D_{\text{Avg}} - D_i(M - i + 1))^2}{M} && (3.2) \\
= \; & \frac{2}{M} \cdot \left( \sum_{i=1}^{M} d_i \cdot (d_{G(i)} - d_{M-i+1}) \right) && (3.3)
\end{aligned}
$$

The last expression equals zero if $d_{G(i)} = d_{M-i+1}$, or $d_{G(i)}$ is a decreasing sequence in function of $i$. In particular is this valid for $G(i) = M - i + 1$.
Assume a bijection $G'$, such that $d_{G'(i)}$ is not a decreasing sequence in function of $i$. Then it can be shown, that by sorting this sequence pairwise, $D_{\text{Var}}(G') - D_{\text{Var}}(M - i + 1)$ can be reduced, thereby altering $G'$, until at last a decreasing order is reached, and $D_{\text{Var}}(G') - D_{\text{Var}}(M - i + 1) = 0$.

This is now proven:
Assume a bijection $G'$, such that $d_{G'(i)}$ is not a decreasing sequence in function of $i$. Then there exist $j$ and $k$ with $j > k$ such that $d_{G'(j)} > d_{G'(k)}$. When $j > k$ then $d_j \geq d_k$ and $d_{M-j+1} \leq d_{M-k+1}$.
If the values of $G'(j)$ and $G'(k)$ are exchanged, the following two terms in the sum of Equation 3.1:

$$d_j \cdot (d_{G'(j)} - d_{M-j+1}) + d_k \cdot (d_{G'(k)} - d_{M-k+1})$$

become:

$$d_j \cdot (d_{G'(k)} - d_{M-j+1}) + d_k \cdot (d_{G'(j)} - d_{M-k+1})$$

Therefore the sum in Equation 3.1 decreases by an amount equal to the difference of the last two expressions. This difference equals:

$$(d_j - d_k) \cdot (d_{G'(j)} - d_{G'(k)}) \geq 0$$

Thus, sorting the sequence $d_{G'(i)}$ pairwise to a decreasing sequence, and thereby altering $G'$, can only decrease the sum in Equation 3.1 (or keep it unchanged). This fact, and the fact that a sorted decreasing sequence makes this sum zero, means that

$$D_{\text{Var}}(G) - D_{\text{Var}}(M - i + 1) \geq 0$$

This proves the third item.

### 3.2.3 The ideal interleaver.

Now a Turbo Code is considered consisting of two identical non-punctured encoders (not even the systematic part of the second encoder is punctured). None of the encoders are terminated. The encoders have thus an identical distance spectrum. It is the role of the interleaver to assign to every codeword of the first encoder, a codeword of the second encoder with the same information weight.
Lemma 2 can now be used to make some statements about the distance spectrum of this Turbo Code. For a certain information weight $w$, the ordered sequence $d_1(w) \leq d_2(w) \leq \ldots \leq d_i(w) \leq \ldots \leq d_{M(w)}(w)$ with $i = 1, 2, \ldots, M(w)$ represents the distance spectrum of each of the encoders. The bijection $G(w)$ realizes the function of the interleaver. The resulting spectrum of the Turbo Code is given by the sequence $D_i(G(w))$ (for every information weight separately!).
Applying Lemma 2 in this manner leads to the following conclusions about the distance spectrum of these Turbo Codes:

1. The average value of the weights of the codewords is not a function of the interleaver.
2. Using no interleaver maximizes the variance of the weights of the codewords.
3. Using an ideal interleaver that interleaves the ordered sequence $d_1(w) \leq d_2(w) \leq \ldots \leq d_i(w) \leq \ldots \leq d_{M(w)}(w)$ with $G(i) = M(w) - i + 1$ minimizes the variance of the weights of the codewords. This interleaver relates low weight codewords for the first encoder to high weight codewords for the second encoder and vice-versa.
4. The use of any other interleaver will result in a variance of the codeword weight between these two extremes.

Therefore the use of an interleaver pushes the spectrum together, towards the average codeword weight, thereby shifting low weight codewords to higher weight.
Notice that the ideal interleaver, as described above, cannot be realized in general. It is impossible to independently relate the codewords of the first constituent encoder to the codewords of the second constituent encoder, as is assumed here. As described in Section 2.6.3, only the basis vectors can be related independently! Therefore, these Turbo Codes with the ideal interleaver will lead to a very weak lower bound on the performance of the code for a certain block length and component encoders.
In Figure 3.1, three spectra are drawn for a Turbo Code with (37,21) encoders and block length $N = 20$: the spectrum without interleaver, the spectrum for a pseudo-random interleaver, and the spectrum with ideal interleaver. The spectrum for the ideal interleaver has a very unrealistic shape: this is expected because the interleaver cannot be realized. Compared to using no interleaver, the pseudorandom interleaver compresses the spectrum, as expected.

## 3.3 The principles of random interleaving.

Random interleaving for Turbo Codes has been introduced by Benedetto et al. in [7]. In [9] the method has been used to evaluate an upper bound on the bit error probability

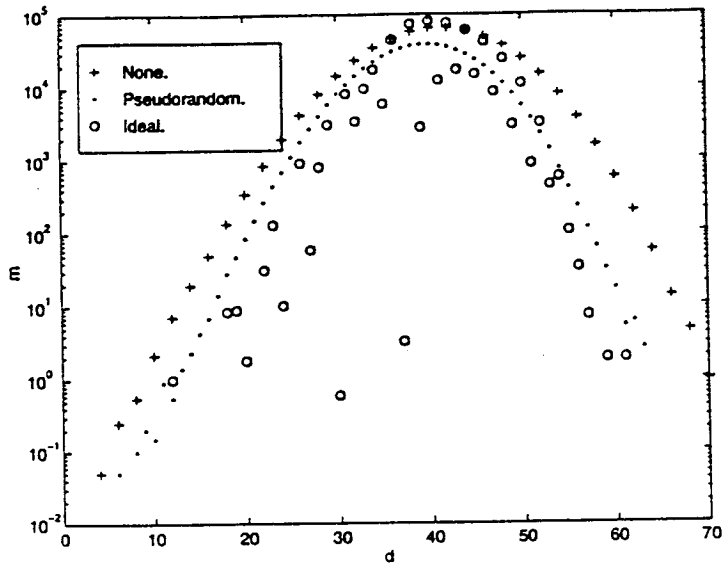



Figure 3.1: The distance spectrum of a Turbo Code: without interleaver, a pseudorandom interleaver, and the ideal interleaver. 'm' stands for $\sum_w w \cdot a(w,d)$, see Equation 2.1. 'd' stands for the Hamming weight of the codewords.

of Turbo Codes with given constituent encoders and block length. The method will be shortly explained here.
Assume two information blocks, with weight $w$, causing detours in both constituent encoders, as in Figure 3.2. The probability that the information block entering the first encoder is interleaved to the information block for the second encoder, is equal to the number of interleavers that interleave the first block to the second block, divided by the total number of interleavers. This, off course, assumes a uniform distribution over the interleavers. The total number of interleavers of length $N$ that interleave correctly is given by $w! \cdot (N-w)!$: There are $w!$ ways of interleaving the ones to the ones, and $(N-w)!$ ways of interleaving the zeros to the zeros. The total number of interleavers of length $N$,

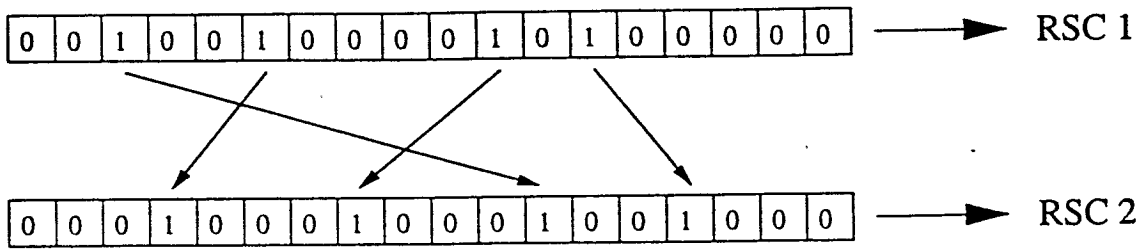



Figure 3.2:

is given by $N!$. Therefore this probability equals $\frac{w!\cdot(N-w)!}{N!}$, or :

$$\frac{1}{\binom{N}{w}}$$

Next, Benedetto et al. introduced the input-redundancy weight enumerating function of a systematic block code as:

$$A^C(W,Z) = \sum_{i,j} A_{i,j} W^i Z^i$$

where $A_{i,j}$ stands for the number of codewords caused by an information block of Hamming weight $i$ whose parity check bits have Hamming weight $j$, so that the total Hamming weight is $i+j$.
To stay consistent with the notation used previously in this report, the following equivalent function is introduced, the input-output weight enumerating function (IOWEF):

$$A^C(W,D) = \sum_{w,d} A^C_{w,d} W^w D^d \tag{3.4}$$

where $A^C_{w,d}$ stands for the number of codewords caused by an information block of Hamming weight $w$ and that have total Hamming weight $d$. The two functions are equivalent because $i+j = w+j = d$.
The Equation 3.4 can be rewritten as:

$$A^C(W,D) = \sum_{w} A^C_w(D) W^w$$

with:

$$A^C_w(D) = \sum_{d} A^C_{w,d} D^d$$

The constituent codes of a Turbo Code can be considered as block codes since the information blocks are finite. Therefore they have IOWEF's $A^{C1}(W,D)$ and $A^{C2}(W,D)$. Benedetto and al. proved that for the IOWEF of the Turbo Code,$A^{TC}(W,D)$, averaged over all possible interleavers:

$$A^{TC}_w(D) = \frac{A^{C1}_w(D) \cdot A^{C2}_w(D)}{\binom{N}{w}} \tag{3.5}$$

The difficulty lies in calculating the terms $A^C_{w,d}$ for each of the constituent encoders. The approach is to calculate how a particular completed detour, or *series of completed detours*, contributes to these terms. A 'series of detours' is defined as an ordered group of detours. This group may contain a particular detour more than once. The 'output weight' of a series of detours is the sum of the output Hamming weights of the detours. The 'length' of a series of detours is defined as the sum of the lenghts of the information sequences that

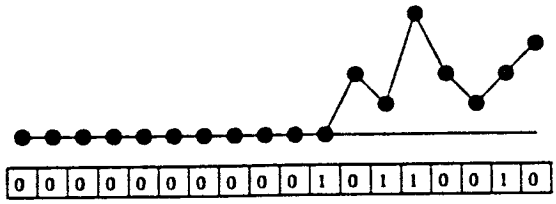



Figure 3.3: There is only one information block that causes this uncompleted detour in the trellis of a constituent encoder.

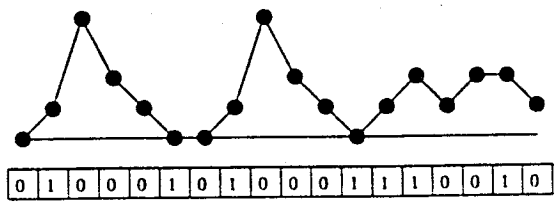



Figure 3.4: This series of three detours contains two completed detours and one incompleted detour.

encoder, and in this specific order, is given by:

$$\begin{pmatrix} N-l+(n-1) \\ (n-1) \end{pmatrix} \tag{3.7}$$

It equals the number of partitions of $N-l$ into $n$ numbers: there are $N-l$ zeros to be divided over $n$ locations before and between the sequences, but not after the last sequence. This series of detours contributes with this amount to $A^C_{w,d}$.
Neglecting information blocks that do not terminate the constituent encoder as in [9] leads to incomplete spectra for Turbo Codes averaged over all interleavers.

### 3.4.2 Extension to punctured codes.

Up to this point, only non-punctured constituent encoders were considered for random interleaving. However, most Turbo coding schemes use puncturing to increase the spectral efficiency.
An example shows the influence of the puncturing on random interleaving: Consider a constituent encoder with generator (37,21) and whose output is punctured as the first constituent encoder in [1]. Figure 3.5 shows two information blocks going into this encoder with $N = 10$. The information blocks cause the same completed detour in the trellis: they both contain the sequence 11111 that causes the completed detour. Nevertheless, they produce a different output Hamming weight due to the puncturing. Instead of contributing with $N-l+1 = 10-5+1 = 6$ to $A_{5,d}$ for some $d$, this detour contributes with 3 to $A_{5,5}$ and 3 to $A_{5,7}$. The best way to think of this, is that there are actually two different sequences causing different output weights, but that there are restrictions on the positions where these sequences may begin in the information block: one sequence has to start on

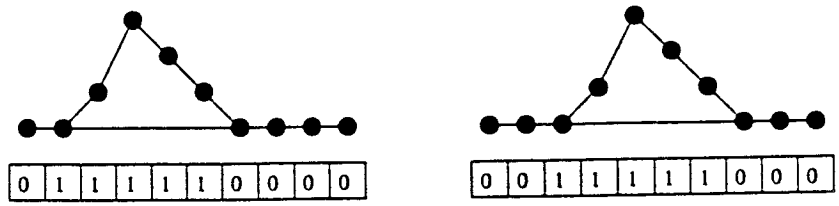



Figure 3.5: These identical sequences 11111 start at different positions in the information block. They produce different output weight due to the puncturing.

an even position, the other on an odd position.
In general, with a puncturing period of $T$, every completed detour for the non-punctured encoder leads to T completed detours for the punctured encoder with different output Hamming weights. Any of the $T$ information sequences of these detours can only start at a limited number of positions in the information block: those positions that have the same value modulo $T$. This means that the contribution of these detours to the distance spectrum of the constituent encoder becomes more complicated to calculate:
Consider a series of $n$ detours. The information sequence causing the $i$-th detour can only start at those positions whose modulo $T$ value equals $p_i$, $i = 1, 2, \ldots, n$. $p_i$ takes on a value in the set $\{0, 1, \ldots, T-1\}$. Now an information block of length $N$ is constructed containing the ordered information sequences of the detours in the series. These sequences are placed so that the number of positions before the first sequence, and the number of positions between two consecutive sequences, is minimized so that this number takes on a value in the set $\{0, 1, \ldots, T-1\}$. The number of positions that remain after the last sequence is called $L$. The spaces before, between, and after the sequences are filled with zeros. This information block causes the series of detours in the trellis of the constituent encoder. Moreover, by taking units of $T$ zeros from the $L$ positions at the end of this information block and inserting them before and between the sequences, more information blocks are formed that cause this series of detours. Note that by inserting $T$ zeros between two sequences, the requirement on the begin positions of the sequences is not violated. The number of partitions of $\lfloor \frac{L}{T} \rfloor$ units of $T$ zeros into $n+1$ locations before, between, and after the sequences, equals:

$$\binom{\lfloor \frac{L}{T} \rfloor + n}{n} \tag{3.8}$$

This is the total number of information blocks that cause this series of detours and it equals the contribution of this series of detours to $A^C_{w,d}$.
It is possible to derive an upper and lower bound for this contribution. $L$ takes on the maximum value $N-l$ when the numbers of positions before the first sequence, and between consecutive sequences, are all zero. $l$ is the sum of the lenghts of the information sequences. The upper bound on Equation 3.8 is therefore:

$$\binom{\lfloor \frac{N-l}{T} \rfloor + n}{n} \tag{3.9}$$

$L$ takes on the minimum value $N - l - n \cdot (T-1)$ when the numbers of positions before the first sequence, and between consecutive sequences, are all $T-1$. The lower bound on Equation 3.8 is therefore:

$$\begin{pmatrix} \lfloor \frac{N-l+n}{T} \rfloor \\ n \end{pmatrix} \tag{3.10}$$

For $N \gg l, n, T$, the two bounds approach one another.

### 3.4.3 Calculating a few terms of the average distance spectrum.

Calculating the whole distance spectrum will only be feasible for Turbo Codes with a small block size $N$. Hereafter, a method will be explained to calculate the distance spectrum of a Turbo Code, averaged over all interleavers, up to a certain weight $d_{\max}$. Only the first encoder is terminated.
For the average Turbo Code, $A_d^{TC}$ is defined as:

$$A_d^{TC} = \sum_{w=1}^{N} w \cdot A_{w,d}^{TC} = \sum_{w=1}^{N} w \cdot a(w,d) \tag{3.11}$$

where $a(w,d) = A_{w,d}^{TC}$. According to Equation 3.5:

$$A_{w,d}^{TC} = \frac{\sum_{x+y=d} A_{w,x}^{C1} \cdot A_{w,y}^{C2}}{\begin{pmatrix} N \\ w \end{pmatrix}} \tag{3.12}$$

The problem is now reduced to determining at most $A_{w,d}^{C1}, A_{w,d}^{C2}$ for $d \leq d_{\max}$ and $w \leq N$. For the first encoder, all completed detours are determined with output Hamming weight $d' \leq d_{\max}$ and length $l' \leq N$. Using these detours, all series of completed detours with output weight $d \leq d_{\max}$ and length $l \leq N$ are assembled and their contributions to $A_{w,d}^{C1}$ are calculated using the techniques explained in the previous sections.
A similar calculation is performed for the second encoder, but incompleted detours are also considered.
The number of completed detours to consider increases exponentially with $d_{\max}$ and so does the complexity of this calculation.

## 3.5 The effect of increasing the interleaver size on one term of the distance spectrum for average Turbo Codes.

### 3.5.1 A property of recursive convolutional codes.

The constituent encoders of Turbo Codes are $(2, 1, m)$ recursive systematic convolutional encoders with memory size $m$. For a recursive convolutional code, the input in the memory at every step is a linear combination of the entering information bit and at least one memory bit.

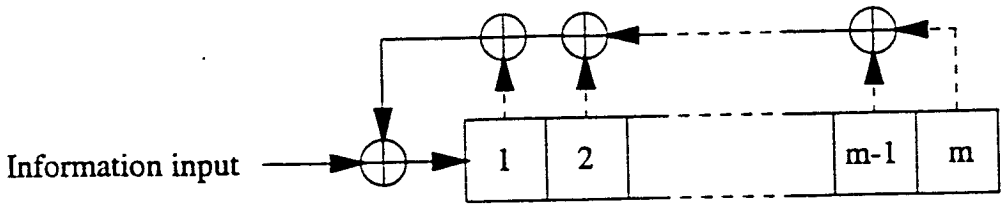



Figure 3.6: The general feedback form of a recursive convolutional encoder.

**Lemma 3** *Given a $(2, 1, m)$ recursive convolutional encoder with memory size $m$, every completed detour in the trellis of this encoder is caused by an information sequence with Hamming weight at least two.*

Proof: Figure 3.6 shows the general feedback form of the encoder.
First, it will be shown that a single information bit with value 1 is needed to take the encoder away from the zero state. Consider an encoder in the all zero state. When an information bit with value 0 enters the encoder, the input into the memory is a linear combination of this 0 and the zero state of the encoder: the encoder must remain in the zero state. However, when an information bit with value 1 enters the encoder, $100\ldots00$ becomes the new state.
If the encoder returns to the zero state during step $t$, then the inputs into the memory at steps $t, t-1, \ldots, t-m+1$ all have to be zero. Before step $t-m+1$, the state of the encoder has a 1 in the first position, otherwise the zero state would have been reached earlier. Starting from this state, $m$ information bits are inserted in the encoder, such that there are $m$ zero inputs entering the encoder's memory. It will now be shown that one of these information bits has to be a 1. After a certain number of steps, the 1 that used to be in the first position of the memory, occupies the last position that is fed back to the input. At this point an information bit with value 1 needs te be inserted so that the input in the memory, which is now the sum of this memory bit with value 1 and the information bit, is zero.
It has been shown that a single 1 is needed to leave the zero state, and at least another 1 to return to the zero state.

### 3.5.2 Increasing the interleaver size for average Turbo Codes.

In Section 3.4.3, a method was explained to determine a few terms of the distance spectrum of the average Turbo Code. The bottom line was to calculate $A_{w,d}^{C1}$ and $A_{w,d}^{C2}$ for a limited number of pairs $(w, d)$ using series of detours for each of the encoders. Then, $A_w^{TC}(D)$ can be determined using Equation 3.12. In this manner, every series of detours for the first encoder is randomly interleaved to every series of detours for the second encoder that has the same information weight $w$. It is possible, however, to skip the calculation of $A_{w,d}^{C1}$ and $A_{w,d}^{C2}$, and calculate the direct contribution of a specific pair of series of detours, one series for each encoder, to $A(W, D)^{TC}$.
Consider for the $i$th encoder, $i = 1, 2$, a series of $n_i$ detours. This series has a length $l_i$, an output weight $d_i$, and an information weight w.

For the first encoder, which is terminated, all the detours of the series are completed. If the detours of the series for the second encoder are also completed, then the contribution of this pair of series to $A^{TC}_{w,d_1+d_2}$ equals:

$$\frac{\binom{N-l_1+n_1}{n_1} \cdot \binom{N-l_2+n_2}{n_2}}{\binom{N}{w}} \tag{3.13}$$

based on Equation 3.5 and Equation 3.6.

If the series of detours for the second encoder contains an incompleted detour, then the contribution of this pair of series to $A^{TC}_{w,d_1+d_2}$ equals:

$$\frac{\binom{N-l_1+n_1}{n_1} \cdot \binom{N-l_2+(n_2-1)}{(n_2-1)}}{\binom{N}{w}} \tag{3.14}$$

based on Equation 3.5, Equation 3.6, and Equation 3.7.
First, the influence of increasing $N$ on Equation 3.13 is investigated. Consider $N \gg l_1, l_2, n_1, n_2$, then Equation 3.13 can be approximated as:

$$\frac{w!}{n_1! \cdot n_2!} \cdot N^{n_1+n_2-w} \tag{3.15}$$

Due to Lemma 3 and the fact that all detours are completed, the minimum information weight per detour is two, and therefore $w \geq 2 \cdot max(n_1, n_2)$. Assume without loss of generality $n_1 \geq n_2$, then $w \geq 2 \cdot n_1$. Three cases are considered:

- $n_1 > n_2$: The exponent of $N$ in Equation 3.15 is always negative. The contribution of this pair of series to $A^{TC}_{w,x+y}$ decreases as $N$ increases.
- $n_1 = n_2$ and $w > 2 \cdot n_1$: The exponent of $N$ in Equation 3.15 is always negative. Same conclusion.
- $n_1 = n_2$ and $w = 2 \cdot n_1$: The exponent of $N$ in Equation 3.15 is zero. The contribution of this pair of series to $A^{TC}_{w,x+y}$ converges to a finite non-zero value as $N$ increases. Note that the series contain an equal number of completed detours, and all these detours have information weight two: they are caused by information sequences with Hamming weight two.

Next, the influence of increasing $N$ on Equation 3.14 is investigated. As above, consider $N \gg l_1, l_2, n_1, n_2$. Equation 3.14 can be approximated as:

$$\frac{w!}{n_1! \cdot (n_2-1)!} \cdot N^{n_1+n_2-w-1} \tag{3.16}$$

For the same reason as mentioned above, $w \geq 2 \cdot n_1$. For the second encoder all the completed detours have at least information Hamming weight two, but the incompleted detour may have information Hamming weight one, and therefore $w \geq 2 \cdot (n_2 - 1) + 1 = 2 \cdot n_2 - 1$. Three cases are considered:

- $n_1 > n_2$: This implies $w \geq 2 \cdot n_1$. The exponent of $N$ in Equation 3.16 is always negative. The contribution of this pair of series to $A^{TC}_{w,x+y}$ decreases as $N$ increases.
- $n_1 = n_2$: Same conclusion.
- $n_1 < n_2$: This implies $w \geq 2 \cdot n_2 - 1$. Same conclusion.

This analysis leads to the following lemma:

**Lemma 4** *A Turbo Code defined by two recursive convolutional constituent encoders is given. The first constituent encoder is terminated, the second encoder not. Two series of detours, one series for each encoder, is given. The series have the same information weight. The contribution of this pair of series to the spectrum of the Turbo Code, averaged over all interleavers with a uniform distribution over the interleavers, converges to a finite non-zero value as the interleaver size increases, if and only if:*

- *the two series contain only completed detours,*
- *the two series contain an equal amount of completed detours,*
- *and these detours have information weight two.*

*In any other case, the contribution of this pair of series converges to zero.*

Note that a puncturing pattern is not mentioned in this lemma: Equation 3.13 and Equation 3.14 are only valid for non-punctured encoders. However, the proof can be repeated with the upper bound, Equation 3.9, and the lower bound, Equation 3.10, that can be extended to incompleted detours. Therefore, the lemma is also valid for punctured encoders. The proof can also be repeated for the case where none of the encoders is terminated. The result is identical.

### 3.5.3 One term of the distance spectrum.

In Section 3.4.3, a method was discussed to calculate some terms of the distance spectrum of an average Turbo Code. First, using series of detours for each of the constituent encoders, $A^{C1}_{w,d}$ and $A^{C2}_{w,d}$ are calculated for some $(w, d)$. Then using Equation 3.12, $A^{TC}_{w,d}$ can be determined.
In the previous section, it has been shown that the influence on the Turbo Code's spectrum of a pair of series of detours, one series for each encoder, can be calculated directly without first determining $A^{C1}_{w,d}$ and $A^{C2}_{w,d}$. Therefore, an equivalent method for calculating $A^{TC}_{w,d}$ is the following: Determine all pairs of series of detours, one series for each encoder, such that all pairs have information Hamming weight $w$ and output Hamming weight $d$. The output Hamming weight of a pair of series is the sum of the output weight of the series. The number of these pairs is finite. Omit those pairs that contain series whose length is greater than the information block size $N$. Then, calculate the influence on $A^{TC}_{w,d}$ for every pair separately.
The results from the previous section are used to predict how $A^{TC}_{w,d}$ evolves for fixed $(w, d)$ as the information block size, equivalent to the interleaver size $N$, increases. Consider

the information block size larger than the length of every series. Then the number of pairs of series that contribute to $A_{w,d}^{TC}$, does not increase as $N$ increases. According to Lemma 4, only those contributions of pairs of series that consist of completed detours with information weight two, converge to a value different from zero. Thus, it can be concluded that for large enough $N$, the contributions of detours with information weight two dominate $A_{w,d}^{TC}$.

**Lemma 5** *A Turbo Code defined by two recursive convolutional constituent encoders and a puncturing pattern is given. The information block size is $N$. For a certain $(w,d)$, the contribution to $A_{w,d}^{TC}$ of pairs of series, one series for each constituent encoder, that contain only completed detours with information weight two, is $A'$.*
*For every $\epsilon$, there exists $N'$ such that $|A_{w,d}^{TC} - A'| < \epsilon$ for every $N > N'$.*

For the distance spectrum of the average Turbo Code, the total multiplicity of the codewords at distance $d$ is $A_d^{TC} = \sum_{w=1}^{N} w \cdot A_{w,d}^{TC}$.

A *non-catastrophic* Turbo Code is defined as a Turbo Code for which $\forall d$, $\exists w'$, independent of the interleaver size, such that $A_{w,d}^{TC} = 0$, $\forall w > w'$. For a given $d$, the number of terms in the sum of Equation 3.11 is then the same for interleaver sizes $N > w'$. Therefore, see Equation 3.12, it is sufficient that one of the constituent encoders is non-catastrophic. This is always the case if one of the constituent encoders is systematic and its systematic output is not punctured. For the remaining part of this chapter only non-catastrophic Turbo Codes are considered.
With the definition of a non-catastrophic Turbo Code, the previous lemma leads to:

**Lemma 6** *A Turbo Code defined by two recursive convolutional constituent encoders and a puncturing pattern is given. The Turbo Code is non-catastrophic. The information block size is $N$. For a certain $d$, the contribution to $A_d^{TC}$ of all pairs of series, one series for each encoder, that contain only completed detours with information weight two, is $A'$.*
*For every $\epsilon$, there exists $N'$ such that $|A_d^{TC} - A'| < \epsilon$ for every $N > N'$.*

Every term of the distance spectrum of an average Turbo Code converges to a finite value as the interleaver size increases.

## 3.6 The free distance of an average Turbo Code.

### 3.6.1 Theory.

In this section, it will be shown that free distance codewords of Turbo Codes most likely originate from a single completed detour with information Hamming weight two in the trellis of each constituent encoder.

Consider the spectrum of the average Turbo Code up to Hamming weight $d_{\max}$. $d_{\max}$ is the smallest codeword weight for which a pair of series of detours contributes to $A_d^{TC}$, where both series only consist of completed detours with information weight two. This weight $d_{\max}$ can easily be found, as well as all such pairs of series:
For the $i$-th encoder, $i = 1, 2$, consider the set $S_i$ of completed detours with information

weight two that cause the minimum output weight $d_i$ compared to other completed detours with information weight two for this encoder. Every element of $S_i$ corresponds to a series of detours containing only this detour. Taking one of these series for each encoder, a pair of series is obtained that contributes to $A^{TC}_{d_1+d_2}$ and for which the series consist of completed detours with information weight two. By construction, no other pair of series for which the series consist of completed detours with information weight two contributes to $A^{TC}_{d}$ for $d < d_1 + d_2$.
According to Lemma 6, as the interleaver size $N$ increases, the multiplicities $A^{TC}_{d}$ for $d < d_1 + d_2$ converge to zero. However, the multiplicity $A^{TC}_{d_1+d_2}$ converges to a finite non-zero value that only depends on the contribution of the pairs constructed above. $A^{TC}_{d_1+d_2}$ converges to:

$$\lim_{N\to\infty} A^{TC}_{d_1+d_2} = 2 \cdot A^{TC}_{2,d_1+d_2} = 2 \cdot \frac{A^{C1}_{2,d_1} \cdot A^{C2}_{2,d_2}}{\binom{N}{2}} \tag{3.17}$$

The following results: *Free distance codewords of Turbo Codes are likely to originate from a single completed detour with information Hamming weight two in the trellis of each constituent encoder.* The construction described above makes it possible to calculate the expected value of the free distance and the expected number of free distance codewords. The interleaver size has to be 'large enough'.
Note that this result can be used to obtain an upper bound on the error floor that can be reached with a particular Turbo Code and interleaver size. There have to be Turbo Codes with a lower error floor than the average Turbo Code. The error floor of the average Turbo Code is given by:

$$\frac{A^{TC}_{d_1+d_2}}{2 \cdot N} \cdot \mathrm{erfc}\left(\sqrt{R \cdot (d_1 + d_2)\frac{E_b}{N_o}}\right)$$

for $N$ sufficiently large.
Note that as $N$ increases, $A^{TC}_{d_1+d_2}$ converges to a finite non-zero value. Therefore: *As the interleaver size of an average Turbo Code increases, its error floor shifts down proportionally to* $\frac{1}{N}$. As the interleaver size increases, it is possible to find interleavers that lower the error floor. This theoretical result confirms the simulation results in [3]. An example in the following section demonstrates the theory.

### 3.6.2 An example.

Consider a Turbo Code with constituent encoders and puncturing pattern as in [1]. The information block size, or interleaver size, is $N$. The construction of Section 3.6.1 is applied to calculate the expected value of the free distance and the expected number of free distance codewords. The following results are obtained:

- The set $S_1$ contains three detours that cause $d_1 = 4$.
- The set $S_2$ contains three detours that cause $d_2 = 2$.
- $A^{C1}_{2,4} = A^{C2}_{2,2} = \lfloor \frac{N-6}{2} \rfloor + \lfloor \frac{N-7}{2} \rfloor + \lfloor \frac{N-12}{2} \rfloor \approx \frac{3}{2} \cdot N$

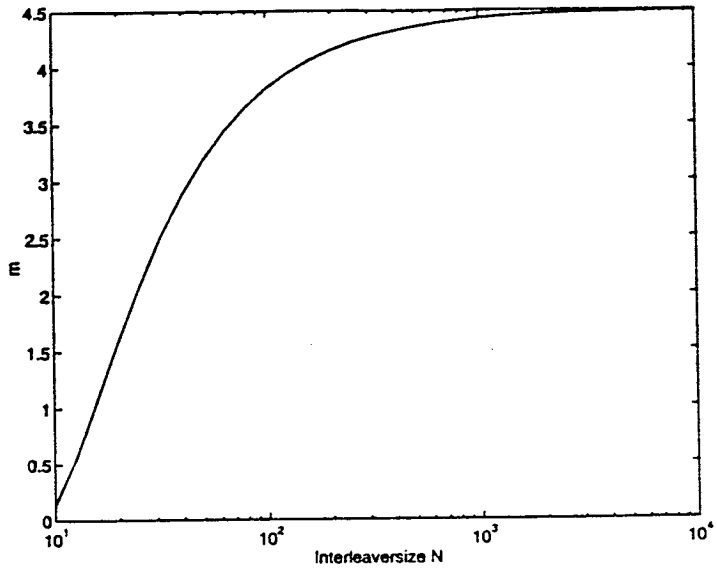



Figure 3.7: This plot shows the convergence of $m = A_{2,6}^{TC}$ to 4.5.

- $A_{d_1+d_2}^{TC} = \sum_{w=1}^{N} w \cdot A_{w,d_1+d_2}^{TC} = A_6^{TC} \approx 2 \cdot A_{2,6}^{TC}$

The third result is calculated using the elements of $S_1, S_2$ and Equation 3.8. Now, using Equation 3.17:

$$A_6^{TC} \approx 2 \cdot \frac{A_{2,4}^{C1} \cdot A_{2,2}^{C2}}{\binom{N}{2}} = 2 \cdot 4.5$$

The expected value of the free distance is six, and the expected number of the free distance codewords, $A_{2,6}^{TC}$, is approximately 4.5 for 'large' interleavers. Figure 3.7 shows the convergence of $A_{2,6}^{TC}$ to 4.5. For $N = 1000$, $A_{2,6}^{TC}$ equals 4.42. For a practical interleaver size of 64K, 4.5 will be a very good approximation. For obtaining $A_6^{TC}$, $A_{2,6}^{TC}$ needs to be multiplied by the information Hamming weight $w = 2$.
These conclusions are confirmed by the results of Section 2.3. There the free distance codewords are determined for a particular Turbo Code with the same constituent encoders, puncturing pattern, and a 64K pseudorandom interleaver. This code has a free distance equal to six and three free distance codewords. Moreover, the form of the free distance codewords is exactly as described above: The free distance codewords cause a single completed detour of information weight two in the trellis of each constituent encoder.
Using the union bound, Equation 2.1, the free distance asymptote of the average Turbo Code is given by:

$$\frac{A_6^{TC}}{2 \cdot N} \cdot \operatorname{erfc}\left(\sqrt{\frac{1}{2} \cdot 6 \cdot \frac{E_b}{N_o}}\right) \approx \frac{2 \cdot 4.5}{2 \cdot N} \cdot \operatorname{erfc}\left(\sqrt{\frac{1}{2} \cdot 6 \cdot \frac{E_b}{N_o}}\right)$$

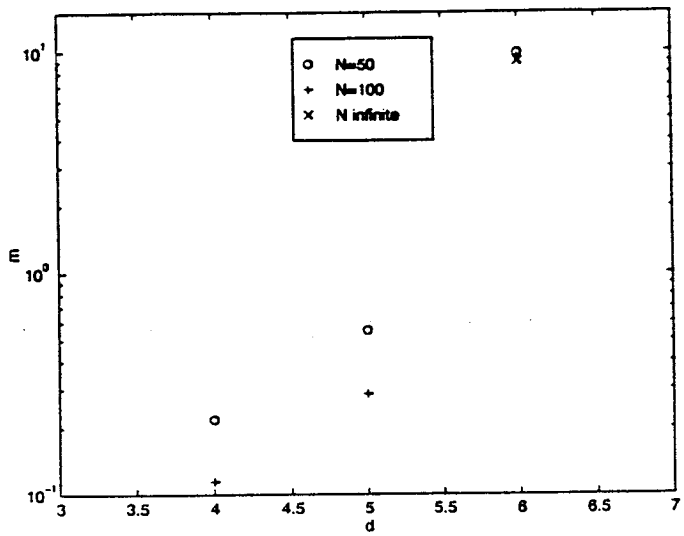



Figure 3.8: This plot shows the convergence of $m = A_d^{TC}$ for $d \leq 6$. As the interleaver size $N$ increases, $A_d^{TC}$ converges to zero for $d < 6$. 'd' stands for the codeword Hamming weight.

This is an upper bound on the obtainable error floor for a particular Turbo Code. As the interleaver size increases, the bound is shifted down proportionally to $\frac{1}{N}$.
Figure 3.8 shows how $A_d^{TC}$ converges to zero for $d < d_1 + d_2 = 6$.

## 3.7 Spectral thinning for increasing interleavers.

### 3.7.1 Intentions.

In the previous section, the behavior of the error floor in function of the interleaver size was explained. As the interleaver size increases, the error floor is shifted down. There is, however, a second characteristic of the performance curve that varies with the interleaver size: As the interleaver size increases, the error floor dominates the performance down to smaller signal to noise ratios, see Figure 1.2 in Section 1.1. In this section an explanation for this behavior is proposed. First, some theoretical evidence is developed to support the idea of spectral thinning. Then, empirical evidence is presented that shows the spectral thinning for a particular Turbo Code. Finally, a theory is proposed that shows how spectral thinning influences the code's performance.

### 3.7.2 Theoretical evidence for spectral thinning.

Section 3.5.3 treats the behavior of one term in the distance spectrum of an average Turbo Code. It was shown that as the interleaver size increases, $A_d^{TC}$ converges to a finite value that depends only on detours with information weight two.

In Section 3.6 this result was used to show how, for large enough interleavers, detours with information weight two determine the free distance of a Turbo Code.
It is now possible to predict how the spectrum changes as the interleaver size increases. The evolution of a certain term $A_{d'}^{TC}$ in function of the interleaver size is discussed.
In Section(1.5.3) it was shown that $A_{d'}^{TC}$ can be calculated as follows:

1. For each encoder determine all detours with output Hamming weight smaller than or equal to $d'$, and length smaller than or equal to the interleaver size $N$. For the first encoder only completed detours are considered, for the second encoder also incompleted detours.

2. For each encoder determine all series of detours such that each series has an output weight equal to $d'$ and a length smaller than or equal to $N$.

3. Determine all pairs of series of detours, one series for each encoder, such that the output weight of each pair equals $d'$.

4. Calculate the contribution of each pair to $A_{d'}^{TC}$ using Equation 3.13, Equation 3.14, or the equivalent of these equations for punctured Turbo Codes.

This algorithm can now be used to predict how $A_{d'}^{TC}$ varies when the interleaver size increases:

- Assume $d' > \frac{N}{R}$, where $R$ stands for the rate of the Turbo Code, then $A_{d'}^{TC} = 0$ because the codeword weight cannot be larger than the codeword length.

- As the interleaver increases, the number of pairs of series that contribute to $A_{d'}^{TC}$ increases because for an increasing number of series with output weight smaller than or equal to $d'$, the length will be smaller than or equal to $N$. Therefore $A_{d'}^{TC}$ increases.

- Above a certain size, increasing the interleaver no longer leads to more pairs that contribute to $A_{d'}^{TC}$. This effect was explained in Section 3.5.3. When the interleaver size keeps increasing, the contributions to $A_{d'}^{TC}$ of those pairs of series, where the series do not only consist of completed detours with information weight two, converge to zero. Therefore, $A_{d'}^{TC}$ decreases as it converges towards its final value, determined only by completed detours with information weight two. This effect is referred to as 'spectral thinning'.

Therefore, the spectrum evolves as shown in Figure 3.9. In the next section, an example illustrates this behavior.

### 3.7.3 Empirical evidence for spectral thinning.

Consider a Turbo Code with constituent encoders and interleaving pattern defined as in [1]. For this code, random interleaving was performed using detours with information weight up to 4 and for codeword weights up to 20. Figure 3.10 shows the results for $N = 50, 100, \infty$. As the interleaver size increases, the spectrum converges to the spectrum of the average Turbo Code with $N = \infty$. This means that the terms converge to zero, or a

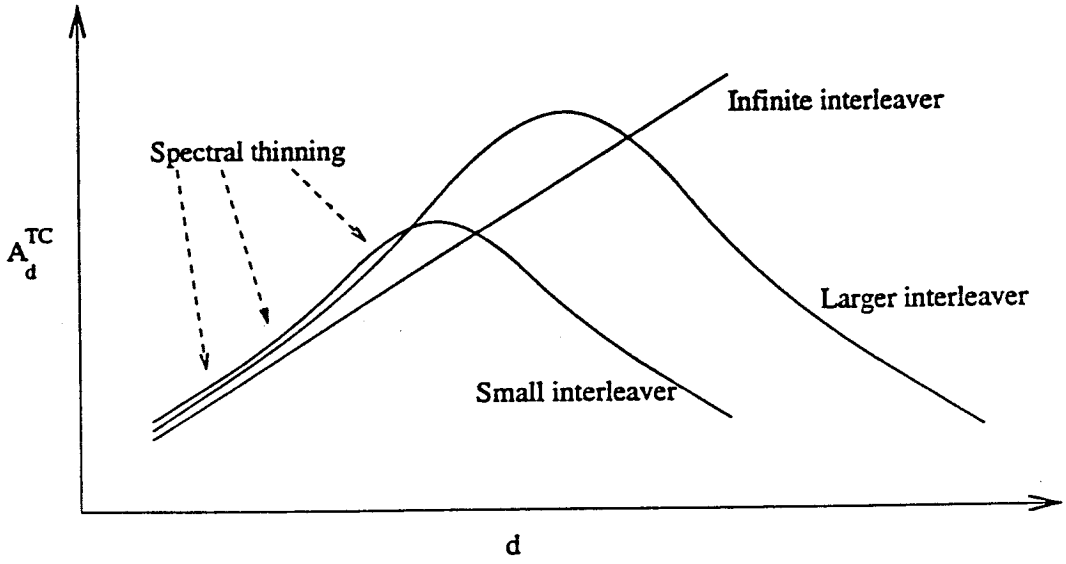



Figure 3.9: This picture shows in principle how the spectrum evolves as the interleaver size increases.

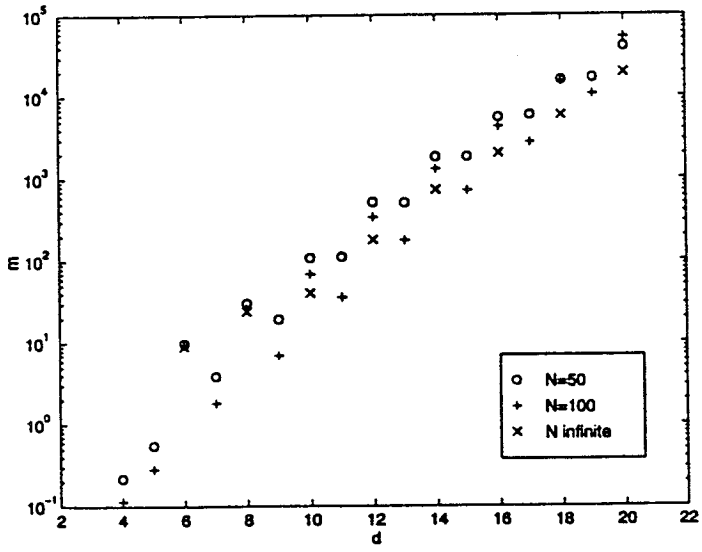



Figure 3.10: This plot shows the effect of spectral thinning for a particular Turbo Code. The spectra were obtained using the algorithm of Section 3.4.3. $m = A_d^{TC}$.

non-zero value. Notice that $A_{20}^{TC}$ is smaller for the $N = 50$ code than for the $N = 100$ code. This is not in contradiction with the results of Section 3.7.2! For this weight, the length of the interleaver limits the number of pairs that contribute to $A_{20}^{TC}$: for many series, the sum of the lengths of the detours exceeds the interleaver size. As the interleaver size increases from 50 to 100, more pairs of series contribute to $A_{20}^{TC}$.
Only detours with information weight up to 4 were used to keep the calculation time feasible. It can be expected that using all relevant detours would make the effect of spectral thinning stronger.

### 3.7.4 A theory of spectral thinning.

In the two previous sections, theoretical and empirical evidence is presented that supports the effect of spectral thinning for Turbo Codes as the interleaver size increases. In this section, a *theory* is proposed that shows the effect of spectral thinning on the performance of Turbo Codes. The theory explains why, for increasing interleaver size, the error floor dominates the performance down to smaller signal to noise ratios, and therefore, the bit error rate curve becomes steeper.
Figure 3.11 shows a part of the spectrum for a Turbo Code with infinite interleaver. The component encoders and the puncturing pattern have been chosen as in [1]. In Figure 3.12, a few terms of the union bound are plotted that correspond to the spectrum of this Turbo Code with infinite interleaver. The free distance term exceeds all other terms: it dominates the performance of this code.
Figure 3.11 also shows parts of the spectra for two other Turbo Codes with finite interleaver. These two spectra are fictional: they have not been calculated as is the case for the Turbo Code with infinite interleaver. However, they illustrate the effect of the spectral thinning described in the previous sections: as the interleaver increases, the spectrum converges to the spectrum of the code with infinite interleaver. Figure 3.12 also shows the terms of the union bound corresponding to these spectra. Note that the free distance term does not dominate the performance at low signal to noise ratios: there is a higher distance term that cuts the free distance term. Thus, the free distance term only dominates the performance for signal to noise ratios above the point where it is cut by a higher order term. This point moves to lower signal to noise ratios for increasing interleavers.
This theory explains the shape of the performance curves of Turbo Codes. As the interleaver size increases, the spectrum thins and the error floor dominates the performance down to smaller signal to noise ratios.
In reality the effects of the error floor shifting down, and the error floor dominating down to smaller SNR, are not independent. Both effects are a consequence of the fact that each term in the spectrum converges to a finite value. As the interleaver size increases, these effects happen simultaneously. There is no reason why this mechanism should stop when a certain interleaver size is reached. Therefore this theory predicts that as the interleaver size increases, the performance of Turbo Codes approaches the Shannon limit.

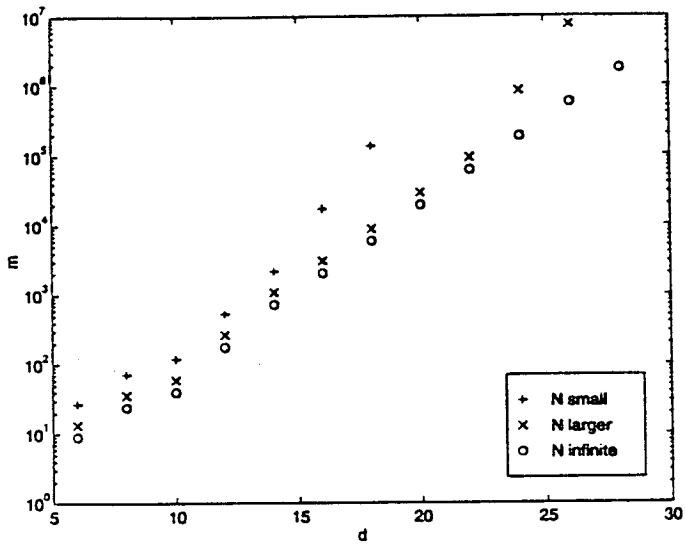



Figure 3.11: This plot shows three spectra of Turbo Codes. The spectrum for the infinite interleaver is exact, the two other spectra are fictional. The spectra show the behavior of spectral thinning. $m = A_d^{TC}$.

## 3.8 Conclusion.

This chapter has focused on the distance spectrum of Turbo Codes to explain their performance. The following conclusions can be drawn:

- Compared to the event that no interleaver is used, the interleaver causes the variance of the spectrum to decrease. It does this by combining low weight codewords for the first constituent encoder with high weight codewords for the second constituent encoder, and vice-versa.
- Every term of the distance spectrum of Turbo Codes with fixed constituent encoders and puncturing pattern, averaged over all interleavers, converges to a finite value as the interleaver size increases.
- For a sufficiently large interleaver size, the free distance codewords most likely cause a single completed detour with information weight two in the trellis of each constituent encoder. It is relatively simple to calculate the expected values for the free distance and the number of free distance codewords.
- The error floor for a Turbo Code, averaged over all interleavers, is an upper bound for the error floor of this Turbo Code with a particular interleaver. This bound shifts down as the interleaver size increases.
- Spectral thinnig occurs as the interleaver size increases. This effect supports a theory that explains the shape of the Turbo Code's performance curves.

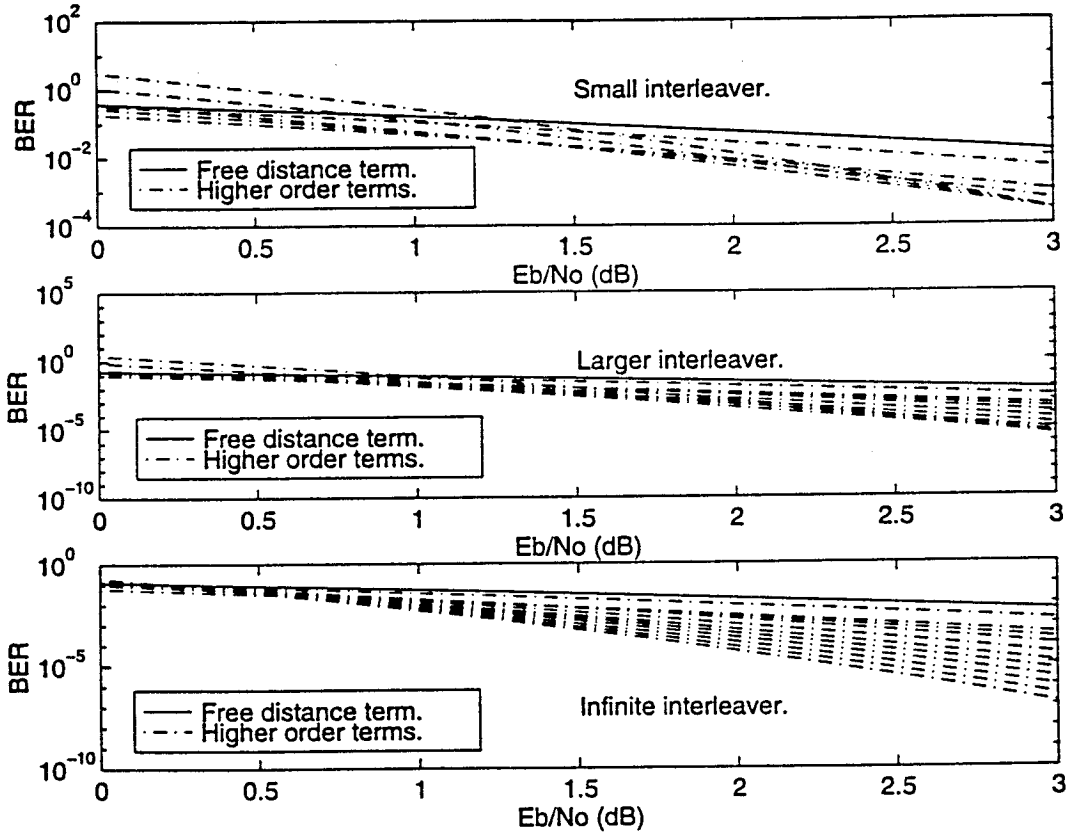



Figure 3.12: This plot shows the influence of spectral thinning on the performance of Turbo Codes. Each of the graphs above depicts a few terms of the union bound corresponding to the spectra in Figure 3.11. Since only the relative position of the terms is important, Equation 2.1 was used with $N = 1$. Note that the free distance term dominates the performance down to smaller signal to noise ratios as the interleaver size increases.

- The combined effects of the error floor and spectral thinning imply that the performance of a Turbo Code approaches the Shannon limit as the interleaver size increases.

The method of random interleaving was improved and extended to punctured codes in order to obtain these results.

# Chapter 4

# The Decoding Complexity of TURBO Codes.

## 4.1 Introduction.

In engineering every decision is a compromise between performance and cost. A careful performance versus complexity comparison is necessary to choose a code for a given application. It would therefore be incomplete to study the performance of Turbo Codes without addressing the decoder's complexity. In this chapter the decoding complexity of Turbo Codes is investigated and compared to the decoding complexity of a Viterbi decoder.

## 4.2 The decoding complexity.

The decoding complexity of Turbo Codes was succesfully investigated by Andersen in [5]. Therefore the results presented in this section differ little from Andersens'. Also the graphical presentation of the operations originates from [5].
*The decoding complexity is measured as the number of operations per information bit that are required to decode.*
First the decoding complexity of the MAP algorithm is considered. For the notations used in this section, see Section 1.4, where the MAP algorithm is discussed. Note that all the operations are floating point. In practice rescaling is needed. Figure 4.1 and Figure 4.2 illustrate the calculation of $\alpha_t(m)$, $\beta_t(m')$, and $\sigma_t'(m', i)$.
The calculation of $\alpha_t(m)$ requires two multiplications and one addition for each state. Assuming an encoder with memory size $M$, this amounts to $2 \cdot 2^M$ multiplications and $2^M$ additions. The same number of operations is needed to calculate $\beta_t(m')$. $\sigma_t'(m', i)$ is calculated along with $\beta_t(m')$ so two more multiplications are needed for every state. For calculating the log-likelyhood ratio $\Lambda(d_t)$, the values of $\sigma_t'(m', i)$ have to be added for every state, and a final division is required. Table 4.1 presents the total number of operations per information bit for the MAP algorithm. Division is assumed to have the same complexity as multiplication.
One decoding iteration requires that the MAP algorithm be applied twice. In [1], constituent encoders with memory size $M = 4$ are used and simulation results are published

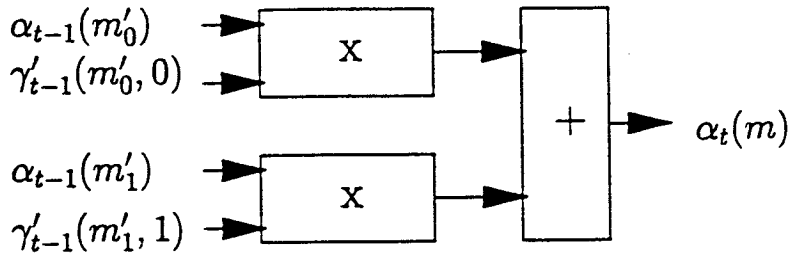



Figure 4.1: This figure illustrates the calculation of $\alpha_t(m)$.

| | Additions | Multiplications |
|---|---|---|
| MAP decoder | $4 \cdot 2^M$ | $6 \cdot 2^M + 1$ |
| Turbo decoder 1 iteration | $8 \cdot 2^M$ | $12 \cdot 2^M + 2$ |
| Turbo decoder M=4 18 iterations | 2304 | 3492 |

Table 4.1: The decoding complexity of a Turbo Code per information bit.

for eighteen iterations. The table also includes the decoding complexity for this Turbo Code.

The channel output and the information exchange between the decoders are assumed to be quantized. When this is the case, $\gamma'_t(m', i)$ can be obtained by $2 \cdot 2^M$ table look-up operations to calculate $\alpha_t(m)$. The values of $\gamma'_t(m', i)$ are then stored to calculate $\beta_t(m')$. The decoder memory requirements can be estimated as $(3 \cdot 2^M + 4) \cdot N$ words. $N$ is the information block size. $\alpha_t(m)$ and $\gamma'_t(m', i)$ need to be stored for the complete block, requiring $3 \cdot 2M \cdot N$ words. Also the complete input block for the encoder must be stored and the two interleavers need $2 \cdot N$ words.

Note that the decoding complexity per information bit only depends on the memory size of the constituent encoder, and not on the information block size $N$. Increasing the information block size improves the performance of Turbo Codes, but does not increase the decoding complexity per information bit.

## 4.3 Turbo decoders versus Viterbi decoders.

For a rate $\frac{1}{2}$ convolutional code with memory size $M$, the Viterbi decoding complexity per information bit is estimated by $3 \cdot 2^M$ integer additions and $2 \cdot 2^M$ table look-up operations: Two branches arrive at every state. For each branch, the metric is found in a look-up table. This branch-metric is then added to the total metric of the path that the branch belongs to. The total metrics of the two paths that arrive in the same state are then compared to determine the surviving path. This comparison is equivalent to a subtraction, or an addition.

For the $(2, 1, 14)$ convolutional code discussed in Section 1.1, the Viterbi decoding complex-

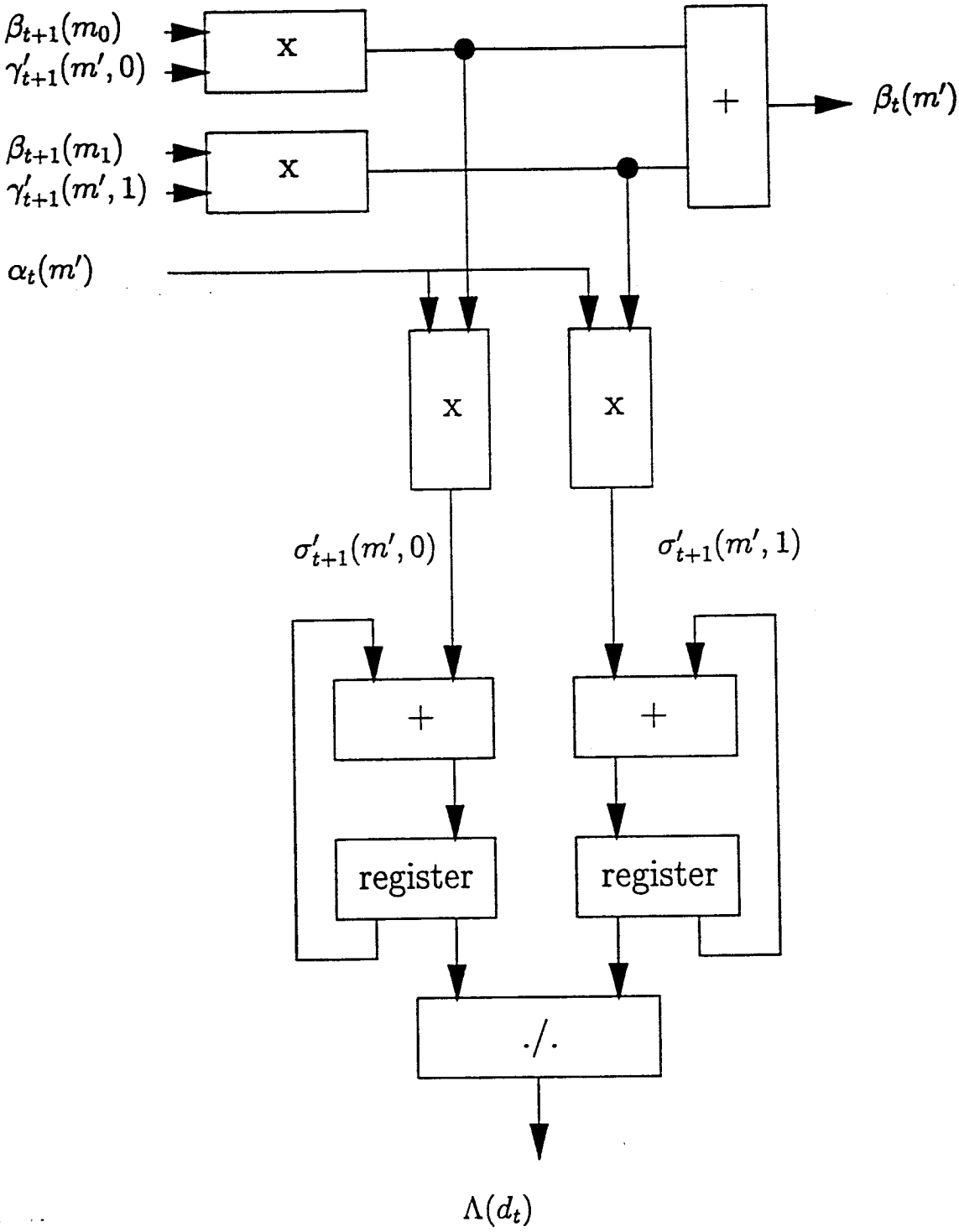



Figure 4.2: This figure illustrates the calculation of $\beta_t(m')$, $\sigma'_t(m', i)$, and $\Lambda(d_t)$.

ity per information bit equals 49152 integer additions and 32768 table look-up operations. It is not trivial to compare the decoding complexity of the Turbo Codes with memory four constituent encoders, eighteen decoder iterations, and a 64K interleaver, to the Viterbi decoding complexity of this $(2, 1, 14)$ convolutional code. The following considerations have to be made:

- The Turbo decoder requires floating point addition and multiplication, while the Viterbi decoder only needs integer addition.
- The Viterbi decoder requires more operations.
- Does the decoder need to be optimized for chip size or decoding time? For practical systems, the possibility of pipelining operations or reusing calculation units needs to be considered.

The comparison has to be made on the hardware level.
Note that in order to improve the performance of a convolutional code, the memory size of the encoder needs to be increased. At the same time the decoding complexity per information bit grows exponentially. This is in contradiction with Turbo Codes where the performance can be improved without increasing the decoding complexity by using a larger information block size. It can be expected that when the performance requirements are high enough, the Turbo Code will outscore convolutional codes in decoding complexity.

## 4.4 Conclusion.

In this chapter, a measure for the decoder complexity of Turbo Codes was discussed. The decoding complexity per information bit only depends on the memory size of the constituent encoders, but not on the information block size. Therefore, the performance of Turbo Codes can be increased without also increasing the decoding complexity.
The decoding complexity of Turbo Codes was compared to the decoding complexity of convolutional codes with Viterbi decoder. The Viterbi decoding complexity grows exponentially with the memory size of the convolutional code. It can be expected that when the performance requirements are sufficiently high, the decoding complexity of Turbo Codes is lower than that of convolutional codes with Viterbi decoding.

# Chapter 5

# Improvements for TURBO Codes.

## 5.1 Introduction.

The performance of Turbo Codes can be improved by increasing the information block size, or equivalently the interleaver size. For a fixed block size, however, the constituent encoders and the interleaver can be used to optimize the performance. Especially interesting are improvements that do not increase the decoding complexity of the code. In the previous chapter, it was concluded that the decoding complexity per information bit is an exponential function of the constituent encoder's memory size. Therefore, the goal is to optimize the performance over all constituent encoders of a given memory size.
In this chapter, constituent encoder selection and interleaver design are addressed.

## 5.2 Constituent encoder selection.

This section addresses the problem of constituent encoder selection for Turbo Codes. As mentioned in the introduction, it is the intention to optimize the encoder for a given memory size.
According to the results of Section 3.6, free distance codewords result from a single completed detour of information weight two in the trellis of each constituent encoder, at least when the interleaver size is sufficiently large. In Section 3.7, the behavior of the distance spectrum is discussed: as the interleaver size increases, every term of the distance spectrum converges to a finite value that is determined only by completed detours with information weight two. This convergence starts at the 'low weight end' of the distance spectrum and expands to higher weights as the interleaver size increases. It is therefore correct to state that the codewords of relatively low Hamming weight for a Turbo Code are caused by completed detours of information weight two. The word 'relatively' is used to stress the dependence of this statement on the interleaver size.
It is therefore trivial to use a constituent encoder which maximizes the output Hamming weight of completed detours of information weight two. The detours that cause the expected value of the free distance now have a larger output weight, so that this expected value increases. Compared to a non-optimal constituent encoder, this encoder shifts the codewords at the 'low weight end' of the spectrum to higher weights.

The search for a constituent encoder that maximizes the output weight for completed detours with information weight two, intuitively leads to maximum cycle length encoders, as will be illustrated in an example. The cycle length of an encoder is the period of the output sequence when the input sequence consists of a single one followed by zeros.
Consider a $(2, 1, 4)$ recursive systematic convolutional code with generators $(37, 21)$. When the sequence $1000\ldots$ is encoded, the output sequences are 1 00000 00000... for the systematic output and 1 10010 10010... for the parity bits. The cycle length 5 can be recognized in the output sequence. When these output sequences are serialised, the result is the first basis vector of the code:

$$11\ 0100000100\ 0100000100\ldots$$

This vector is periodic: it contains the periodic sequence 0100000100. The other basis vectors can be found as described in Section 2.6.3. Because the encoder is time-invariant, they can be obtained by shifting the sequence above.
The output weight produced by the input sequence 100001 can now be found by adding the corresponding basis vectors:

$$\begin{array}{rl} & 11\ 0100000100\ 0100000100\ 0100000100\ldots \\ + & \underline{00\ 0000000011\ 0100000100\ 0100000100\ldots} \\ = & 11\ 0100000111\ 0000000000\ 0000000000\ldots \end{array}$$

The output weight is six. When the sum of two basis vectors for a recursive convolutional encoder produces a finite output weight, the input sequence corresponding to these vectors causes a completed detour of information weight two in the trellis. Note that in order to obtain a finite output weight, the two basis vectors have to be shifted by a multiple of the cycle length. The basis vectors corresponding to the sequence 100001 are shifted by one cycle length. Each additional shift of the basis vectors over one cycle length results in an additional output weight of two.
Now consider a $(2, 1, 4)$ recursive systematic convolutional code with generators $(23, 35)$. The first basis vector is:

$$11\ 0101000000010000010100010001010101000000010000010100010001010\ldots$$

It contains the periodic part 0101000000010000010100010001010. The cycle length is 15, which is the maximum cycle length for memory 4 encoders. The basis vectors corresponding to the input sequence 1000000000000001 are shifted by one cycle length. Therefore this input sequence causes a completed detour in the trellis of the encoder with information weight two. Again, the output weight can be calculated by adding the corresponding basis vectors:

$$\begin{array}{rl} & 11\ 0101000000010000010100010001010101000000010000010100010001010\ldots \\ + & \underline{00\ 0000000000000000000000000000011\ 0101000000010000010100010001010\ldots} \\ = & 11\ 0101000000010000010100010001100000000000000000000000000000000\ldots \end{array}$$

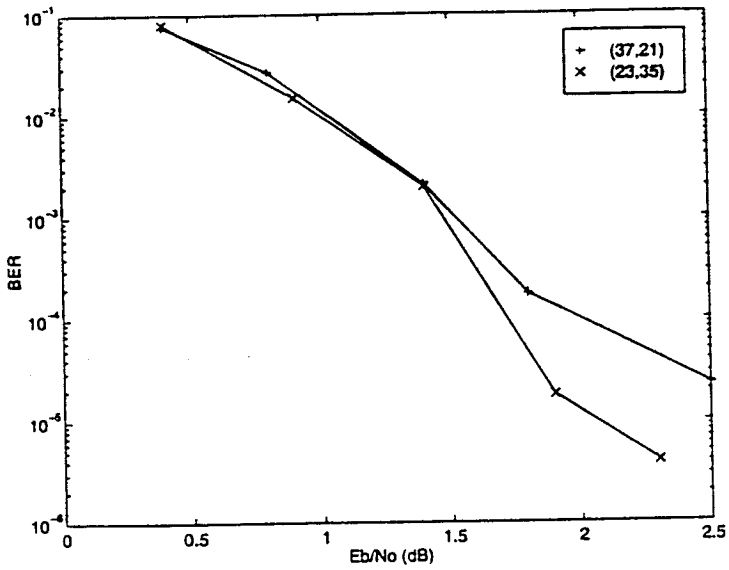



Figure 5.1: The performance of two Turbo Codes with pseudorandom interleaver of size 400 and puncturing pattern as defined in [1]. One Turbo Code has (37,21) constituent encoders, the other (23,35) constituent encoders.

The output weight is ten. Each additional shift of the basis vectors over one cycle length results in an additional output weight of eight.
The maximum cycle length code produces more output weight for completed detours of information weight two. The reason is intuitively clear: For the encoder with larger cycle length, the basis vectors have a larger period so that their repeated part can contain more ones. It is the weight of the repeated part of the basis vectors that determines the output weight of the sum of two basis vectors that are shifted over multiple cycle lengths.
It would be difficult to prove that maximum cycle length encoders are *always* optimal: the interleaver size has to be large enough so that information sequences with weight two are likely to determine the free distance. Note that also the puncturing of the output has to be taken into account. Moreover, the proposed method only optimizes the low weight end of the spectrum. This causes the error floor to shift down, but does not change the performance at low signal to noise ratios. This is illustrated in Figure 5.1.

Finally, it needs to be mentioned that Robertson proposed the use of the $(23, 35)$ generators instead of the $(37, 21)$ generators in [3]. Now theoretical results have been found that explain why the maximum cycle length encoder is better.

## 5.3 Interleaver design.

The second way to improve a Turbo Code's performance is by optimizing the interleaver. A method for interleaver optimization has been proposed by Robertson in [3]. This method optimizes the interleaver for the free distance. The method works as follows:

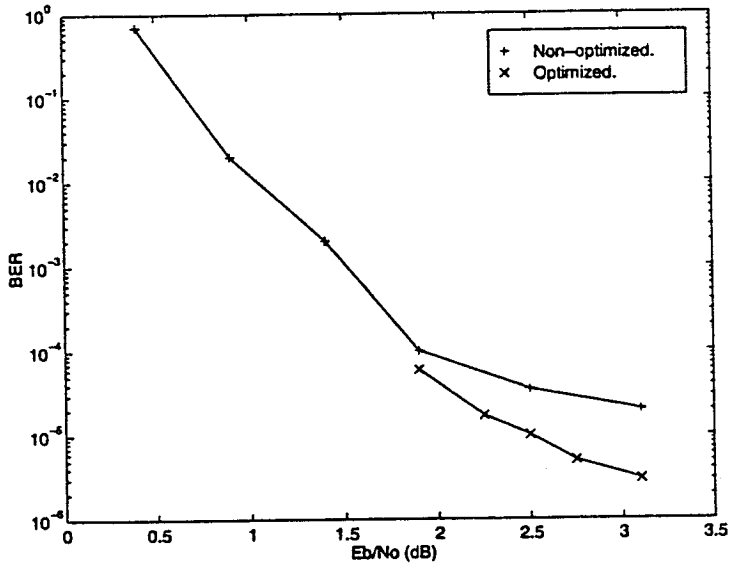



Figure 5.2: The performance of two Turbo Codes with constituent encoders and puncturing pattern as in [1]. The interleaver size is 400. One code has an optimized interleaver, the other code a pseudorandom interleaver.

# Chapter 6

# The Relation Between TURBO Codes and Product Codes.

When Turbo Codes were first introduced, a few publications suggested that Turbo Codes obtain their performance in a similar way as product codes, see [10]. However, there are several facts that contradict this theory.
The encoders for Turbo Codes and product codes contain the same components: two constituent encoders and an interleaver. The difference lies in the interconnection of these components. In the Turbo encoder, there is a parallel concatenation of the constituent encoders as opposed to a serial concatenation for the product encoder. Therefore, an information sequence is interleaved for the Turbo Code, and no code sequence as is the case for product codes: The second constituent encoder of a Turbo encoder does not encode the parity bits produced by the first constituent encoder. However, encoding the parity bits of the first constituent encoder is crucial to the performance of product codes.
A second important difference lies in the aim of the coding schemes. The aim of building a product code is to obtain a code with a high free distance. This code is then used at moderate and high signal to noise ratios (SNR) where the free distance dominates the performance. For Turbo Codes which are used at low SNR, the free distance is just one element that determines the performance. As the SNR decreases, more terms of the distance spectrum influence the performance of the code.
An interesting question remains to be solved. For a product code, increasing the interleaver size does not increase the free distance. Is it possible to keep increasing the free distance by increasing the interleaver size for a Turbo Code with given constituent encoders and puncturing pattern? If not, what is the maximum free distance that can be reached, and what effect prohibits the further increase?
It can be concluded that Turbo Codes and product codes are two fundamentally different coding schemes.

# Chapter 7

# Conclusions.

In Chapter 2, an algorithm was developed to calculate the free distance of a Turbo Code. The algorithm was applied to a Turbo Code with a 64K pseudorandom interleaver and to a Turbo Code with a $120 \times 120$ rectangular interleaver. The results led to the following conclusion:
The error floor is caused by the free distance asymptote of the code. In case the free distance is low, the free distance asymptote has a small slope at low signal to noise ratios: the error floor looks flat. A turbo Code can have a small free distance and still show a good performance when the number of free distance codewords is small relative to the interleaver size.

In Chapter 3, the method of random interleaving was corrected to include incompleted detours, and extended to punctured Turbo Codes. Using random interleaving, it was derived that every term of the distance spectrum of the average Turbo Code (averaged over all interleavers of a given size) converges to a finite value as the interleaver size increases. This has the following consequences:
The number of free distance codewords for the average Turbo Code converges to a finite value as the interleaver size increases, while the free distance stays constant. Therefore, the error floor is shifted down as the interleaver size increases.
The spectrum for the average Turbo Code shows the effect of 'spectral thinning'. 'Spectral thinning' can explain the fact that for Turbo Codes with larger interleavers, the error floor dominates the performance down to smaller signal to noise ratios.
According to the *theory* developed in Section 3.7, the performance of Turbo Codes approaches the Shannon limit as the interleaver size increases.

In Chapter 4, the decoding complexity of Turbo Codes is reviewed and compared to the decoding complexity of convolutional codes with Viterbi decoders. The decoding complexity was defined as the number of operations required to decode one information bit. For the Turbo Code, the decoding complexity is only determined by the memory size of the constituent encoders. As the interleaver size increases and the performance improves, the decoding complexity for the Turbo Code stays constant. However, improving the performance of a convolutional code simultaneously increases the decoding complexity.

In Chapter 5, constituent encoder selection and interleaver design are addressed. The proposed methods only lower the error floor, but have no influence on the performance at low sigal to noise ratios. The performance of Turbo Codes with maximum cycle length constituent encoders is explained.

In Chapter 6, Turbo Codes are compared to product codes. These two coding schemes are fundamentally different.

# Bibliography


[1] Claude Berrou, Alain Glavieux and Punya Thitimajshima, *Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes (1).*, in Proceedings of ICC'93, Geneva, Switzerland, May 1993.

[2] Claude Berrou and Alain Glavieux, *Turbo Codes: general principles and applications.*, in Proceedings of 6th Tirrenia International Workshop on Digital Communications, Tirrenia, Italy, September 1993.

[3] Patrick Robertson, *Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) Codes.*, in Proceedings of GLOBECOM '94, volume 3 , pages 1298-1303, San Francisco, California, November 1994.

[4] 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, pages 284-287, March 1974.

[5] J.D. Andersen, *The TURBO Coding Scheme.*, Report IT-146, Technical University of Denmark, June 1994.

[6] J. Hagenauer and L. Papke, *Decoding 'TURBO'-codes with the soft output Viterbi algorithm (SOVA).*, Proc. 1994 IEEE Int. Sym. Inform. Theory, Trondheim, Norway, page 164, 1994.

[7] S. Benedetto and G. Montorsi, *Performance evaluation of turbo-codes.*, Electronics Letters, Vol. 31, No. 3, page 163, 2nd February 1995.

[8] S. Benedetto and G. Montorsi, *Average performance of parallel concatenated block codes.*, Electronics Letters, Vol. 31, No. 3, page 156, 2nd February 1995.

[9] S. Benedetto and G. Montorsi, *Unveiling turbo codes: some results on parallel concatenated coding schemes.*, Dipartimento di Elettronica, Politecnico di Torino., 27 January 1995.

[10] A.S. Barbulescu and S.S. Pietrobon, *Interleaver design for turbo codes.*, Electronics Letters, Vol. 30, No. 25, page 2107, 8th December 1994.

[11] A.S. Barbulescu and S.S. Pietrobon, *Terminating the trellis of turbo-codes in the same state.*, Electronics Letters, Vol. 31, No. 1, page 22, 5th January 1995.

[12] A.S. Barbulescu and S.S. Pietrobon, *Rate compatible turbo codes.*, Electronics Letters, Vol. 31, No. 7, page 535, 30th March 1995.

[13] P. Jung, *Novel low complexity decoder for turbo-codes.*, Electronics Letters, Vol. 31, No. 2, page 86, 19th January 1995.

[14] P. Jung and M. Nasshan, *Dependence of the error performance of turbo-codes on the interleaver structure in short frame transmission systems.*, Electronics Letters, Vol. 30, No. 4, page 287, 17th February 1994.

[15] O. Joerssen and H. Meyr, *Terminating the trellis of turbo-codes.*, Electronics Letters, Vol. 30, No. 16, page 1285, 4th August 1994.

[16] James L. Massey, *Applied Digital Information Theory II*, Lecture Notes, ETH, 1995.

[17] Peter Elias, *Error-Free Coding.*, IRE Trans. Inform. Theory, vol. PGIT-4,pp. 29-37, September 1954.

[18] Dieter Arnold and Guido Meyerhans, *The Realisation of the Turbo-Coding System.*, ETH, 14th July 1995.

[19] S. Dolinar, *A new code for Galileo.*, TDA Progress Report,pp. 83-96, January-March 1988.