Understanding Turbo Codes

Ali H. Mugaibel and Maan A. Kousa

King Fahd University of Petroleum and Minerals
PO Box 1721, Dhahran 31261, Saudi Arabia

Abstract

The astonishing performance of turbo codes attracted many researchers and this has resulted in an explosive amount of literature since their introduction few years ago. As turbo codes did not actually arise from applying a pre-existing theory, most of their outstanding features remain to be explained. This paper gives some explanation of the code performance and sheds some light on the relative importance of the code ingredients with particular emphasis on interleaving and puncturing.

Introduction

In channel coding, redundancy is introduced in the information sequence in order to increase its reliability. The channel coding theorem states that even at relatively low E_b/N_0 reliable communication can still be maintained. However, the theorem tells us nothing about how to design the code that achieves such performance. All what it says is that the code should appear random. Unfortunately random codes are very difficult to decode. We need to have some structure in the code to make the decoding feasible. Researchers have been trying to resolve these two seemingly conflicting needs: structure and randomness.

Lately in [4], the proposal of **P**arallel-**C**oncatenated **C**onvolutional **C**odes (PCCC), called *turbo codes*, has solved the dilemma of structure and randomness by allowing structure through concatenation and randomness through interleaving. The introduction of turbo codes has increased the interest in the coding area since these codes give most of the gain promised by the channel-coding theorem.

Turbo codes have an astonishing performance of bit error rate (BER) at relatively low E_b/N_0. To give an idea of how powerful turbo codes are, for a frame size of $256\times256=65536$ bits we can achieve a $BER=10^{-5}$ over AWGN channel at only $E_b/N_0=0.7$dB, which is very close to Shannon limit [4]. For a Rayleigh fading channel a $BER=10^{-5}$ can be achieved at $E_b/N_0=4.3$dB which represents a gain of 2.3dB as compared to classical convolutional codes with similar complexity [14].

In the following sections the structure of both the encoder and the decoder are explained. The effects of interleaving and puncturing on the code performance are discussed. The paper concludes with a summary and suggestions for further research.

Turbo encoder

In a simplified turbo code, there are two convolutional encoders in parallel. The information bits are scrambled before entering the second encoder. The codeword in a turbo code consists of the input bits - i.e. the code is systematic - followed by the parity check bits from the first encoder and then the parity bits from the second encoder, as depicted in Figure 1.

The simplified turbo code block diagram in Figure1 shows only two branches. In general, one can have multiple turbo encoders with more than two branches. The convolutional code at every branch is called the **c**onstituent **c**ode (CC). The CCs can have similar or different generator functions. We will concentrate on the usual configuration with two branches having the same CC. A PAD is shown in the figure to append the proper sequence of bits to terminate all the encoders to the all-zero state. This is because a convolutional code may be used to generate a block code if we use beginning and tail bits. If we have one encoder then the required tail is

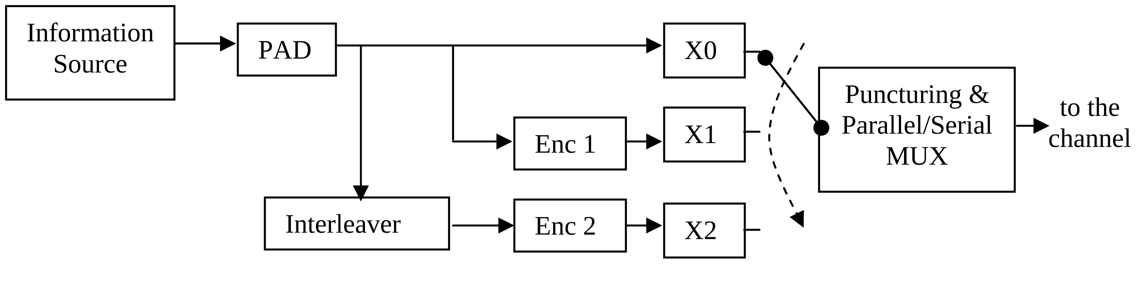


Figure 1: Simplified Turbo Encoder

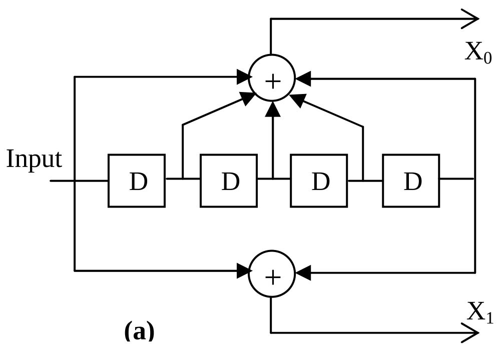
a sequence of zeros with length equal to the memory order *m*. The problem of terminating both encoders simultaneously seems to be difficult because of the interleaver. However, it is still possible to do with *m* tail bits only [6].

In general we can have another interleaver before the first encoder but usually it is replaced with a delay line to account for the interleaver delay and keep the branches working simultaneously.

Puncturing can be introduced to increase the rate of the convolutional code beyond that resulting from the basic structure of the encoder. Separate sections are devoted for interleaving and puncturing.

Some codes are called *recursive* since the state of the internal shift register depends on the past outputs. Figure 2 illustrates a non-recursive non-systematic convolutional code with its corresponding recursive systematic code. X_0 and X_1 are the check bits. Note that for the systematic convolutional encoder, one of the outputs, X_0, is exactly the input sequence.

In turbo codes **R**ecursive **S**ystematic **C**onvolutional (RSC) codes are proved to perform better than the non-recursive ones [4][3]. An RSC encoder can be



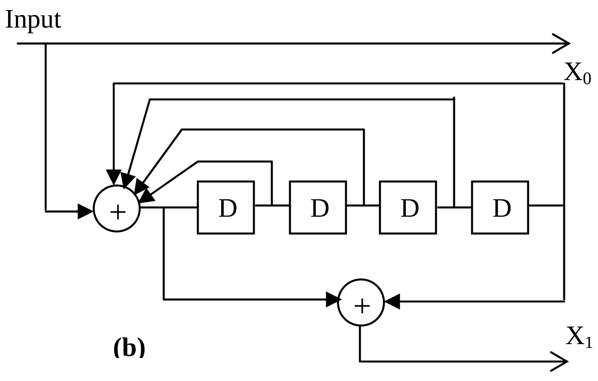


Figure 2: (a) Classical Non-Recursive Non-Systematic Code

(b) Recursive Systematic Code

obtained from an non-systematic non-recursive encoder by setting one of the outputs equal to the input (if we have one input) and using a feed back. The trellis and the free distance ($d_{free)}$) also will be the same for both codes [4]. Of course, the output sequence does not correspond to the same input in the two codes because the two generator functions are not the same.

Turbo decoder

The decoder works in an iterative way. Figure 3 shows a block diagram of a turbo decoder. The iteration stage is shown with doted lines to differentiate it from the initialization stage. Only one loop is performed at a time. In practice the number of iterations does not exceed 18, and in many cases 6 iterations can provide satisfactory performance [4]. Actually, the term turbo codes is given for this iterative decoder scheme with reference to the turbo engine principle. The first decoder will decode the sequence and then pass the hard decision together with a reliability estimate of this decision to the next decoder. Now, the second decoder will have extra information for the decoding; a priori value together with the sequence. The interleaver in-between is responsible for making the two decisions uncorrelated and the channel between the two decoders will seem to be memoryless due to interleaving.

The exact procedure in what information to pass to the next decoder or next iteration stage is a subject of research. In the next section, we describe a widely accepted decoding algorithm, which is the modified soft output Viterbi algorithm.

Soft Output Viterbi Algorithm (SOVA)

Algorithms used in decoding convolutional codes can be modified to be used in decoding turbo codes. In the original paper on turbo codes [4], a modified BAHL *et al* algorithm was proposed for the decoding stage. This algorithm is based on **M**aximum **A**-posteriori **P**robability (MAP). The problem with this algorithm is the inherent complexity and time delay. MAP algorithm was originally developed to minimize the bit-error probability instead of the sequence error probability. The algorithm, although optimal, seems less attractive due to the increased complexity.

Viterbi algorithm is an optimal decoding method that minimizes the probability of sequence error for convolutional codes. A modified version of Viterbi algorithm, called SOVA (**S**oft **O**utput **V**iterbi **A**lgorithm), which uses soft outputs is introduced in [13][14]. SOVA has only twice the complexity of Viterbi algorithm. The new algorithm will make it possible to integrate both the encoder and the decoder in a single silicon chip with unmatched performance at the present time [4].

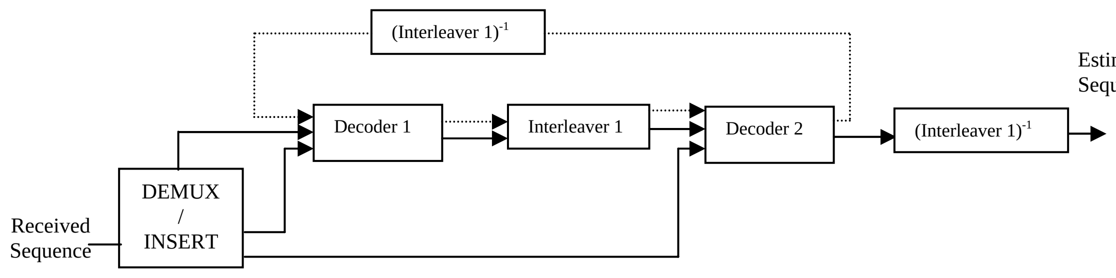


Figure 3: Block Diagram of a Turbo Decoder

The key point in decoding is that every decoder will pass a reliability estimate together with the hard decision. Specifically, the decoder shall deliver for each symbol an estimate of the probability, p`, that this symbol has been incorrectly detected, that is,

p`= Prob{estimated symbol ≠sent symbol | received symbol }

The practical performance of turbo codes resides in the availability of a simple sub-optimal iterative decoding. Iterative SOVA output approaches maximum likelihood (ML) decoding performance bound by increasing the number of iterations [2]. However, the question of convergence is still waiting for a concrete answer.

The a posteriori probability of each bit or the reliability factor will produce the soft output required. If in the trellis we would like to decide on the survivor, we can take the path metric as an indication of the possibility of making an error. Both paths will be stored and information will be passed to the second decoder which is going to perform separate decoding and decide upon the survivor. The final decision will depend on the two decisions and will be passed for further iterations if needed. The exact algorithm together with an estimate of the storage and computation requirements are given in [13].

In [17], it was shown that SOVA as introduced above suffers from two degradations. First, the reliability information of the decoder output is too optimistic especially for bad channels. Secondly, the assumption that the decoders' decisions are uncorrelated is not completely true. An improved decoding with SOVA, called Normalized SOVA, that provides remedy to these two problems was introduced by [17].

Decoding details of turbo codes are out of the scope of this paper. An interested reader is one is referred to [13] [14] and [17] for further details.

Performance evaluation

Lots of research were carried out to find bounds on the performance on turbo codes [2][3][10], but the inherent complexity of the code prevented general and tight results. Simulation is the usual mean for judging the performance of turbo codes. However, simulation requires extensive computation especially at high values of signal to noise ratios since the probability of error will be very low.

Most of the theoretical bounds on turbo codes are based on the union bound, which can be applied to any linear code. To analyze the turbo encoder the effect of the decoder is de-coupled by assuming the availability of an optimum or near optimum decoder. The decision on the encoder is based on the free distance and the weight distribution of the code. The free distance (d_{free}) is defined to be the minimum Hamming weight of all possible codewords. The distribution of the codewords based on the Hamming weight will determine the code performance. The turbo encoder is usually represented by its transfer function or by a table or tree describing the distance spectrum. Both the transfer function and the distance spectrum are very helpful in developing a performance bound for a given code.

Interleaving

An *interleaver* is a device that rearranges the ordering of sequence of symbols in a deterministic manner. Associated with the interleaver is a *deinterleaver* that applies the inverse permutation to restore the original sequence [9].

According to [1] the most critical part in the design of a turbo code is the interleaver. The two main issues in the interleaver design are the interleaver size and the interleaver map. The size of the interleaver plays an important rule in the trade off between performance and time (delay) since both of them are directly proportional to the size. On the other hand, the map of the interleaver plays an important role in setting the code performance.

Conventionally, interleaving is used to spread out the errors occurring in burst. For turbo codes, the interleaver has more functions. Interleaving is used to feed the encoders with permutations so that the generated redundancy sequences can be assumed independent. The validity of the assumption that the generated redundancy sequences are independent is a function of the particular interleaver used. This will exclude a number of interleavers, which generate regular sequences such as cyclic shifts [1].

Another key role of the interleaver is to shape the weight distribution of the code, which ultimately controls its performance. This is so because the interleaver will decide which word of the second encoder will be concatenated with the current word of the first encoder, and hence what weight the complete codeword will have [1]. So the aim of the designer is to produce (by manipulating the weights of the second redundancy part through interleaver mapping) whole codewords with the overall weights as large as possible [18]. Turbo codes, unlike convolutional codes, make the distribution of the weight more important than the minimum distance [2].

Another issue that is worth considering in the design of the interleaver is the termination of the trellis of both convolutional encoders. By properly designing the map of the interleaver, it is possible to force the two encoders to the all-zero state with only *m* bits (where *m* is the memory length of the convolutional encoder assuming the same convolutional code is used in both encoders). To achieve that a condition on the interleaver is proposed by [6] and demonstrated by [20].

There have been many attempts to characterize the effects of the interleaver on the performance of turbo codes. To overcome the difficulty of representing the interleaver map or the difficulty of enumerating all the permutations the authors in [2] introduced an abstract interleaver called *uniform interleaver*, defined as follows:

A uniform interleaver of length k is a probabilistic device, which maps a given input word of weight w into all distinct C_w^k permutations of it with equal probability $1/C_w^k$.

The uniform interleaver can not be used in practice since one is confronted with deterministic interleavers. However, it has been shown that for each value of signal to noise ratio, the performance obtained with the uniform interleaver is achievable by at least one deterministic interleaver [2].

The concept of uniform interleaver was further used in the design and evaluation of turbo codes. In [Per96] an asymptotic bound on the performance was given as a function of the interleaver length and some other code parameters.

In [2] it was shown through a bound that random interleavers offer performance close to the average ones, independent, to a large extent, of the particular interleaver used. It was also shown that the beneficial effect of increasing the interleaver length tends to decrease at high k (interleaver length). Actually the effect of interleaver length should be considered in conjunction with the memory span of the CCs.

Dolinar and Divsalar [11] compared the difference in performance between random and nonrandom interleavers. The authors discussed a partial separation of the problem of picking good permutations and that of picking good component codes.

Researchers are still working to develop design guidelines and to relate the interleaver parameters to the code performance.

Puncturing

Puncturing is the process of deleting some bits from the codeword according to a puncturing matrix. The puncturing matrix (P) consists of zeros and ones where the zero represents an *omitted* bit and the one represents an *emitted* bit. It is usually used to increase the rate of a given code. Puncturing can be applied to both block and convolutional codes. An example of the puncturing matrix to go from rate 1/2 to rate 2/3 is given by $P = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$. This matrix implies that the first bit is always transmitted while every other second bit is omitted.

For turbo codes, the same decoder may serve for various coding rates by means of puncturing, allowing the same silicon product to be used in different applications [5]. When the redundant information of a given encoder is not transmitted, the corresponding decoder input is set to zero. Of course, the decoder needs to know the current puncturing table. This function is performed by the DEMUX/INSERTION block in the turbo decoder (See Figure 2). The DEMUX will demultiplex the stream between the decoders and the INSERTION will insert an analog zero if the corresponding bit is omitted. When the code is punctured, the branch metric corresponding to the punctured bits need not be computed.

Determining the best puncturing pattern for turbo codes is still an open problem. [5] suggests that systematic bits should not be punctured (Berrous puncturing). The paper demonstrated, by simulation, that it is always better to avoid puncturing systematic bits. However, in [15] a new puncturing is suggested, named UKL puncturing. It concludes that nonsystematic version of the designed punctured code is advantageous at high values of E_b/N_0 which are related to low BERs. This result was tested over AWGN and fully interleaved flat Rayleigh fading channels.

One disadvantage of punctured codes is that error events at high rates and higher distances can be quite large. The decision depth of punctured codes is generally longer. This problem is not severe since in most applications data frames are transmitted with sync words and proper termination of short frames. [12]

Puncturing is a trade off between rate and performance, but, fortunately, punctured codes come with 0.1 or 0.2 dB of the optimum code (as reported for the convolutional codes)[19]. Convolutional codes and their punctured alternatives together with their difference in performance are tabulated [9].

It is suggested that using unequal puncturing (puncturing one encoder outputs different than the other) improves the performance [16]. In [8] Caire shows that the performance is not achieved by considering puncturing alone but the interleaver should be designed jointly. In [7], Caire suggested to define the puncturing pattern on the interleaver map. Moreover, adaptive puncturing can be used depending on the channel noise.

Conclusion

The excellent performance of turbo codes requires careful understanding of the code ingredients. The effects of the code parameters on the performance need further investigation. Analytical method for accurately predicting the performance of turbo codes in terms of BER is a topic for further research. Proper understanding of the code will help in making the code applicable for real time applications.

Reference

[1] Battail, Gérard, "A conceptual framework for understanding turbo codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, No. 2, February 1998, pp. 245-254.

[2] Benedetto, Sergio and Montorsi., Guido “Unveiling turbo-codes: some results on parallel concatenated coding schemes,” *IEEE Transactions on Information Theory,* vol. 42, No. 2, March 1996, pp. 409-428.

[3] Benedetto, Sergio and Montorsi., Guido “Design of parallel concatenated convolutional codes,” *IEEE Transactions on Communications*, vol. 44, No. 5, May 1996, pp. 591-600.

[4] Berrou, C., Glavieux, A. and Thitimajhima, P., “Near Shannon limit error correcting coding and decoding : turbo-codes,” *Proc. Of ICC ‘93*, Geneva, May 1993, pp. 1064-1070.

[5] Berrou, C. and Glavieux, A.: "Near optimum error correcting coding and decoding: Turbo-Codes", *IEEE Transactions on Communications*, 1996, vol. 44, pp. 1261-1271.

[6] Blackert, W. J., Hall, E. K., and Wilson, S. G., "Turbo code termination and interleaver conditions," *Electronic Letters,* vol.31, No.24, November 1995, pp. 2082-2084.

[7] Caire, G. and Lencher, G. ,"Turbo codes with unequal error protection" , *Electronic Letters,* Mar. 1996, vol. 32, No. 7, pp. 629-631.

[8] G. Caire and E Biglieri, "Parallel concatenated codes with unequal error protection," *IEEE Transactions on Communications* , vol. 46, No. 5, May 1998, pp. 565-567.

[9] Clark, G. and Cain, J., “Error-Correction Coding for Digital Communications” New York: *Plenum Publishing Corporation* 1988 .

[10] Divsalar, D. Dolinar, S. and Pollara, F., "Transfer function bounds on the performance of turbo codes" , *TDA Progress Report 42-122*, Aug. 1995, pp. 44-55.

[11] Dolinar, S. and Divsalar, D. "Weight distribution for turbo codes using random and nonrandom permutations," *TDA Progress Report 42-122* , Aug. 1995, pp. 56-65.

[12] Hagenauer, J. , "Rate compatible punctured convolutional codes (RCPC-codes) and their application," *IEEE Transactions on Communications,* Apr. 1988, vol. 36, pp. 389-400.

[13] Hagenauer, J., Hoeher, P., “A Viterbi algorithm with soft-decision outputs and its applications,” *Proceedings of GLOBECOM ‘89,* Dallas, Texas, Nov. 1989, pp. 47.11-47.17.

[14] Hagenauer, J. and Papke, L., “Decoding turbo codes with the soft-output Viterbi algorithm (SOVA),” in *Proceedings of International Symposium On Information Theory* (Trondheim, Norway, June 1994), p. 164.

[15] Jung, P. and Plechinger,J. "Performance of rate compatible punctured Turbo-codes for mobile radio applications," *Electronics Lettes*, 1997, vol. 33, No.25, pp. 2102-2103.

[16] Mohammadi, A. H. and Khandani, A. K., "Unequal error protection on the turbo-encoder output bits", *Proceedings of ICC'97 - International Conference on Communications*, vol. 2 , 1997, pp. 730-734.

[17] Papke, L. and Robertson, P., “Improved decoding with the SOVA in a parallel concatenated (turbo code) scheme,” *IEEE Int. Conference on Communications ICC ’96,* 1996, pp. 102-106.

[18] Svirid, Yuri V., "Weight distribution of turbo-codes," *Proceedings of 1995 International Symposium on Iinformation Theory*, 17-22 Sep. 1995 p.38.

[19] Sweeney, P. ,“ Error Control Coding : an Introduction,” UK: *Prentice Hall International Ltd.* 1991.

[20] Valenti, Matthew C., “An introduction to turbo codes,” *Not published*.