AIML Journal, Volume (6), Issue (2), June, 2006

Design of an Interleaver for Turbo Codes using Genetic Algorithms

Shobha Rekh[1], Subha Rani[2], Wim Hordijk[1], Princy Gift[1], Shanmugam[3]

[1]*Karunya Deemed University,* [2] *PSG College of Technology,* [3] *Bannari Amman Institute of Technology,*
Coimbatore, India
shobarekh@rediffmail.com

Abstract

Turbo codes are a powerful channel coding technique employed in communication systems. Turbo codes emerged in 1993 and have since become a popular area of communications research. The excellent performance of turbo codes depends strongly on their interleaver pattern. However, finding the best interleaver pattern requires an exhaustive search, and is thus a difficult task. We use a genetic algorithm to search for good interleaver patterns of length N in the space of all possible N! patterns. Compared with random interleavers, the interleaver pattern found by the genetic algorithm is able to achieve a gain of 0.1 db for an interleaver of size N=50 .

Keywords : *Interleaver, Puncturing, Genetic Algorithms***,**
Turbo Codes

1. Introduction

Turbo codes have been widely considered to be the most powerful error control code of practical importance. Turbo codes emerged in 1993 [1-3] and have since become a popular area of communications research. Turbo codes have solved the dilemma of structure and randomness by allowing structure through concatenation and randomness through interleaving. The important characteristics of turbo codes are the small Bit Error Rate (BER) achieved even at a low signal to noise ratio (SNR), and the flattening of the error rate curve, i.e., the error floor at moderate and high values of SNR. The performance of turbo codes is due to the randomness created by the interleaver and the iterative decoding. The motivation behind this work is that the random interleaver is better than other interleavers in terms of the bit error rate. Random interleavers, however, perform worse for certain combinations of inputs, which generate low output weights. Hence it is necessary to find the best interleaver pattern from a possibility of N! combinations. At low SNR's, the interleaver size is very critical for the code performance as it can increase the interleaver gain and enhance the code performance. This performance increases when the frame size is increased. But the disadvantage of going for higher frame size is that the latency also increases, as given in Eqn.1

$$t_d = \frac{K_f}{R_b} N_i \qquad (1)$$

Where R_b is the bit rate, K_f is the frame size and N_i is the number of decoding stages. Hence it is necessary to design good interleavers for small frame sizes in order to reduce the latency.

In this paper, a genetic algorithm is used to search for good interleaver patterns, using as objective function the bit error rate of the bits transmitted. In the past, not much work has been done on the design of interleavers using any search algorithms, as the search space is very large and the objective function is difficult to be defined properly. An ant colony algorithm for finding good interleaving patterns in turbo codes has been proposed in [4]. The performance evaluation was done only on a few test patterns, which may cause low output weights. However, the results are better than the interleaver design proposed in [5], although the reliability is less. A genetic algorithm has been applied to find suitable interleaver patterns in [6], where the objective function is maximization of the free distance. Finding the free distance is complex, however, and suitable algorithms have to be adopted to find the free distance in less time. In this paper, the objective function used is the bit error rate, which proves to be more reliable.

The paper is organized as follows. The basic construction of turbo codes is reviewed in section 2. In section 3, the genetic algorithm as used for searching for interleavers is explained. Simulation results of the best interleaver pattern found by the genetic algorithm, and a comparison with random interleavers is presented in section 4. Finally, conclusions and future work are discussed in section 5.

2. Turbo Coding

2.1 A Turbo Encoder

A turbo encoder is formed by parallel concatenation of two recursive systematic convolutional (RSC) encoders separated by an interleaver. A block diagram of a rate 1/3 turbo encoder is shown in Figure 1.

The generator matrix of a rate 1/2 component RSC code can be represented as in Eqn. 2 where $g_0(D)$ and $g_1(D)$ are a feed back and feed forward polynomial with degree ν, respectively. In the encoder, the same information sequence is encoded twice but in a different order.

$$G(D)=\left[1,\frac{g_1(D)}{g_0(D)}\right] \qquad (2)$$

The first encoder operates directly on the input sequence, denoted by c, of length N. The first component encoder has two outputs. The first output, denoted by v_0, is equal to the input sequence since the encoder is systematic. The other output is the parity check sequence, denoted by v_1. The interleaved information sequence at the input of the second encoder is denoted by *c'*. Only the parity check sequence of the second encoder, denoted by v_2, is transmitted. The information sequence v_0 and the parity check sequences of the two encoders v_1 and v_2, are multiplexed to generate the turbo code sequence. The overall code rate is thus 1/3.

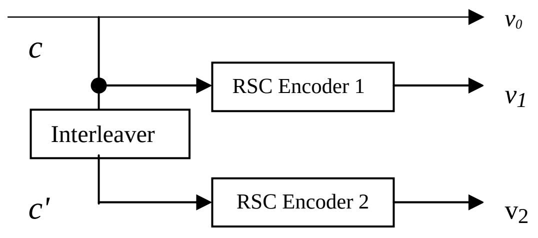

Figure 1. Turbo encoder

2.2 Turbo interleaver

The interleaver in turbo coding is a pseudo-random block scrambler defined by a permutation of *N* elements with no repetitions. The first role of the interleaver is to generate a long block code from small memory convolutional codes. Secondly, it decorrelates the inputs to the two decoders so that an iterative sub-optimum decoding algorithm based on information exchange between the two component decoders can be applied. If the input sequences to the two component decoders are decor related, there is a probability that after correction of some errors in one decoder some of the remaining errors should become correctable in the second decoder. The interleaver length is critical for the code performance; particularly at low SNR's as the code performance is dominated by the interleaver gain. For significant spectral lines, which dominate the turbo code performance, the error coefficients decrease with increasing interleaver size. From Figure. 3, we can conclude that for increasing interleaver sizes, the bit error rate comes down.

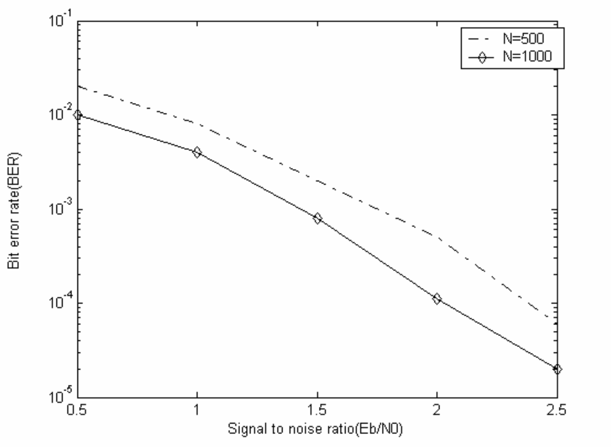

Figure 3. Performance analysis for varying frame sizes.

The interleaver structure [3] is important for the performance at high SNR's, as it affects the distance properties of the overall turbo code. If an input sequence generates a low weight parity sequence from the component encoder, it is desirable that the interleaver is able to break this type of input sequence. If the interleaver is designed to break the low weight input sequences so that the resulting turbo code has a large minimum free distance, the error performance at high SNR's can be improved. Hence an optimal interleaver, which is able to work for small frame sizes at low SNR's, is an interesting and important problem. Here, we aim to find an optimal (or close to optimal) pattern from the set of N! possibilities.

2.3 Turbo Decoding

The iterative decoding consists of two component decoders serially concatenated via an interleaver. There are two main algorithms to decode the data: (i) Soft Output Viterbi Algorithm (SOVA) and (ii) Maximum A Posteriori Algorithm (MAP). The first algorithm finds the most probable output data sequence. The trellis diagram is drawn and the path with the least Hamming distance is found. The MAP algorithm finds the marginal probability that the received bit was a 1 or a 0. Since the bit could occur in many different code words, the sum of all the probabilities is considered. Using the likelihood ratio of the marginal distributions from 1 and 0 makes the decision. The MAP algorithm is optimal for estimating an input information bit in terms of the

BER. Extensive work has been reported on the MAP algorithm [3], [7], [8]. In this paper we have applied the MAP algorithm since it gives better results.

3. Genetic algorithms

3.1 Introduction

The idea of applying the biological principle of evolution to artificial systems, introduced more than three decades ago, has seen impressive growth in the past few years. Genetic algorithms (GAs), originally developed by John Holland [10], are search and optimization algorithms based on the mechanisms of natural selection and genetics. They combine the idea of "survival of the fittest" among simple character strings with a structured yet randomized information exchange to form a search algorithm that "evolves" solutions to a given problem. The algorithm maintains a set of candidate solutions called a population. Repeatedly, solutions from one population are used to form a new population, i.e., the next generation. Solutions that are used to form new solutions are selected according to their fitness, which is a measure of how good of a solution they are to the problem at hand. The more fit they are, the more chances they have to be selected and form new solutions through "genetic" operators like crossover and mutation.

As it turns out, GAs are very successful at finding good solutions to difficult problems in a reasonable amount of time, often outperforming other known search algorithms. Thus, it seems promising to use GAs to try to find good interleaver patterns. For a complete and detailed introduction to GAs, see [9], [11]. Here, we only give a brief overview of the GA operators and parameter settings as we used them in our application.

3.2 Search space and genetic encoding

The space of all solutions to a problem, among which the optimal solution resides, is called the search space. Each point in the search space represents one possible solution. In case of the interleaver design problem, the search space consists of all N! permutations of size N. Thus, our GA will search through this space of permutations where we used N=50. Each possible permutation is simply encoded as a string of the numbers 1 to N, in the order corresponding to the encoded permutation.

3.3 Fitness function

Next, a fitness function has to be defined. Since the bit error rate is an exact measure of the performance of turbo codes, it can be used as the fitness function for the GA. The turbo code bit error probability upper bound based on the code distance spectrum is given by

$$P_b(e) \le \sum_{d=d_{free}} B_d Q\left(\sqrt{2d_{free} R \frac{E_b}{N_0}}\right) \qquad (3)$$

where the set of all pairs of (d,B_d) is the code distance spectrum, R is the code rate, d_{free} is the free distance, E_b is the signal energy per bit, N_0 is the single sided power spectral density of Gaussian noise, and $Q(x)$ is the complimentary error function. At low SNR's [3], the bit error probability is dominated by a number of medium weight spectral line error coefficients B_d. At high SNR's, the free distance d_{free} determines the error performance However, computation of the bit error probability using the above equation is a tedious and time consuming process. Hence, we simulate the transmission and reception of data by adding Additive White Gaussian Noise (AWGN), as represented in Figure 2. The data received at the decoder is evaluated for the bit error probability.

For each transmitted bit c, the MAP algorithm generates its hard estimate and soft output in the form of the a posteriori probability on the basis of the received sequence r. It computes the log-likelihood ratio given as

$$\Lambda(c_t) = \log \frac{P_r\{c_t = 1|r\}}{P_r\{c_t = 0|r\}} \qquad (4)$$

for $1 \le t \le n$, where n is the received sequence length, and compares this value to a zero threshold to determine the hard estimate c_t as

$$c_t = \begin{cases} 1 & \text{if } \Lambda(c_t) \ge 0 \\ 0 & \text{otherwise} \end{cases} \qquad (5)$$

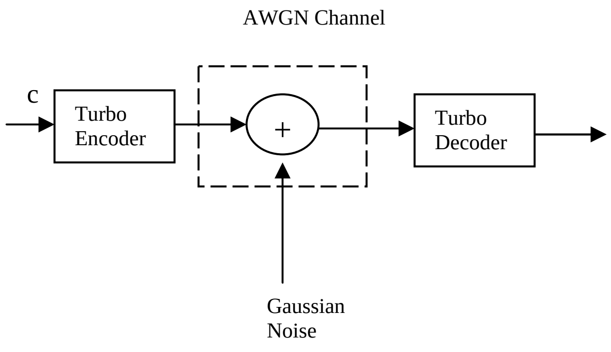

Figure 2. AWGN channel

The data is sent over the Gaussian channel through the turbo encoder. The transmitted signal gets corrupted due to the noise in the channel. The MAP decoder decodes the received data and the received bits are estimated to get the original data bits transmitted. The bit error rate is calculated by finding the ratio of the number of bits in error to the total number of bits transmitted over the channel at any instant of time. The bit error rate thus calculated is taken as the fitness value. The objective is to minimize the bit error rate.

3.3.1 Test space

For turbo codes, the error paths with minimum information weight $w_{min}=2$ dominate the bit error probability performance. The effective free distance [12] of turbo code is defined as

$$d_{free,eff}=2+2z_{min} \tag{6}$$

where z_{min} is the lowest weight of the parity check sequence in error paths of the turbo encoders generated by an information sequence with weight 2. The effective free distance plays a major role in the error performance of turbo codes.

Hence the set of all weight two codes are taken as the test space. Different sets of 200 random samples were chosen from the set of all weight 2 codes for every generation.

3.4 Genetic algorithm operators

3.4.1 Selection

We have used the standard roulette wheel selection method, without using any fitness scaling. So, individuals are selected directly in proportion to their fitness values relative to the population total.

3.4.2 Crossover

The crossover method we used is Partially Matched Crossover (PMX) [9]. Consider two individuals (permutations) as parents,

9 8 4 | 5 6 7 | 1 3 2 0

8 7 1 | 2 3 0 | 9 5 4 6

After (randomly) choosing the crossover points at positions 3 and 6 (indicated by the vertical bars), each string maps to constituents of the matching section of its mate. The repeated symbols are exchanged on a point-to-point basis yielding two offspring individuals as follows.

9 8 4 | 2 3 0 | 1 6 5 7

8 0 1 | 5 6 7 | 9 2 4 3

We used crossover rates of $P_c=0.8$ and $P_c=1.0$, respectively.

3.4.3 Mutation

As mutation operator, we used inversion: the order of a randomly chosen substring of a given permutation is inverted, e.g.,

9 8 | 4 5 6 7 | 1 3 2 0

becomes

9 8 | 7 6 5 4 | 1 3 2 0

In each generation, each newly generated individual has a probability P_m of being mutated, where we used $P_m=0.05$.

3.4.4 Other Parameters

The population size in our experiments is 50 and the number of generations is also taken as 50. We performed several runs, and took the best result out of all of these different runs. The performance of the best interleaver found by the GA is analysed in the next section.

4. Results

Since random and block interleavers are most commonly used, first we compared these two types of interleavers for 500 frames (see Figure 4). The graph clearly shows that the random interleaver performs better than the block interleaver, so the comparison of the best interleaver found by our GA is done only with random interleavers (two randomly chosen ones). For this comparison, the interleaver performances are calculated (off-line) over 1000 random frames from the set of all possible frames (this time not only weight two codes, as is done during the GA run). In fact, when running the GA using a test space of 1000 random frames (out of all possible ones, not only weight two codes), it not only becomes more time consuming, but there is no improvement over the performance of the two (randomly chosen) random interleavers (see Figure 5). However, using a test space with only weight two codes during the GA run, shows a significant improvement compared to the random interleavers. The results for different crossover probabilities (1.0 and 0.8) are shown in Fig's 6 and 7, respectively. The improvement is evident only in the region of low SNR (within 1 db) as the free distance dominates the performance in the region of high SNR. Since the bit error rate is taken as the objective function in this paper, the results apply only for low SNR.

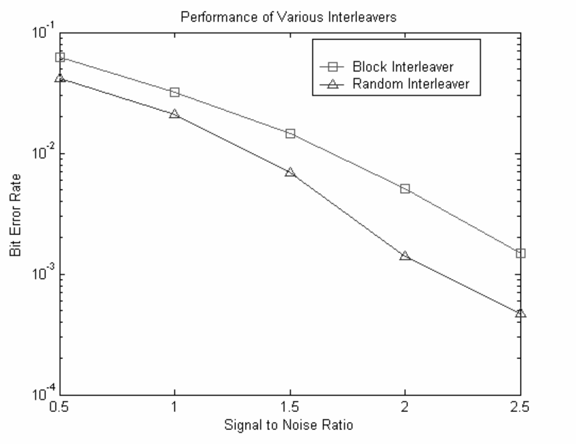

Figure 4. Performance analysis of Block and Random interleavers.

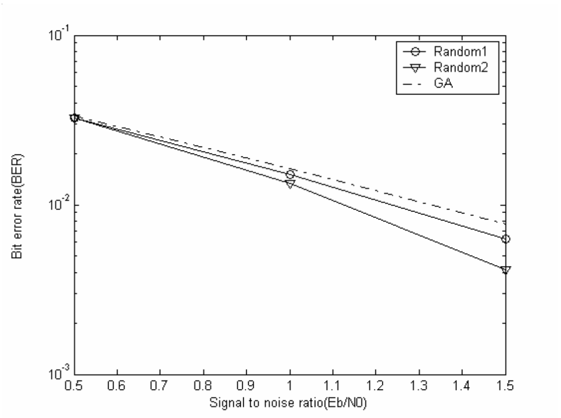

Figure 5. Performance analysis of the best GA interleaver found over a test space of 1000 random sample frames. This gives no improvement over the random interleavers.

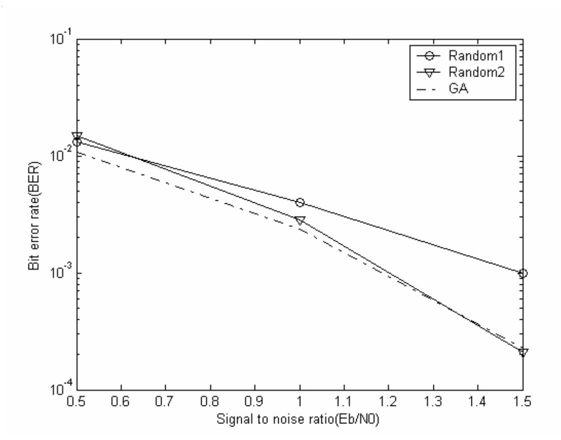

Figure 6. Performance analysis of the best GA interleaver found over a test space of 200 random weight 2 codes (Pc=1.0).

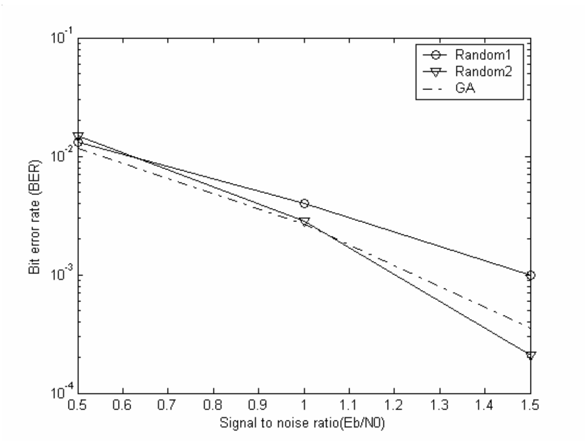

Figure 7. Performance analysis of the best GA interleaver found over a test space of 200 random weight 2 codes (Pc=0.8).

5. Conclusions and future directions

A genetic algorithm has been applied to find good interleavers to be used in turbo codes. The results show that significant improvements can be achieved, up to 0.1db, for an interleaver of size 50. Since the search performance of a GA partly depends on the chosen operators and parameters, possibly even larger improvements can be found by fine-tuning the GA parameters (such as crossover and mutation rates, populations size, etc.).

Future work also includes applying the genetic algorithm for larger sizes of interleavers. This could be achieved by finding a better fitness function which can be evaluated in a small test space. The algorithm could also be made to work for high SNR's by using the free distance as the objective function. So, there is still much additional work to be done, but our initial results are very promising, and warrant further investigation.

6. References

[1] C. Berrou, A. Glavieux, and P. Thitimajshaima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in Proc. IEEE Int. Conf. Commun., May 1993, pp. 1064–1070.

[2] C. Berrou, A. Glavieux , "Near optimum error correcting coding and decoding: Turbo codes," IEEE Trans. Communications, vol. 44,No 10 pp. 1261–1271,Oct. 1996.

[3] Branka Vucetic and Jinhong Yuan, "Turbo Codes Principles and Applications," Buton: Kluwer Academic Publishers, 2000

[4] R.Hoshyar, S.H.Jamali and C.Locus,"Ant colony algorithm for finding good interleaving pattern in turbo codes," IEE Proc,Commun, Vol. 147, No.5 October 2000

[5] Daneshgaran and Mondin. M,"Design of interleavers for turbo codes: iterative interleaver growth algorithms of polynomial complexity", IEEE Trans. Inf. Theory, 1999,pp 1315-1323

[6] Nicolas Durand,Jean-Marc Alliot and Boris Bartolome,"Turbo codes optimization using genetic algorithms," in Proc , IEEE Congress on Evolutionary Computation ,vol.. 2 pp 822-829

[7] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," European Trans. on Telecommun.,vol.8, pp.119-125, Mar./Apr. 1997.

[8] P. Robertson, "Improving decoder and code structure of parallel concatenated recursive systematic (turbo) codes," in Proc., IEEE Int. Conf. on Universal Personal Communications (1994), pp. 183-187.

[9]David E.Goldberg "Genetic algorithms in search, optimization and machine learning", Pearson education, 2003.

[10] John H. Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975.

[11] Melanie Mitchell, "An Introduction to Genetic Algorithms", MIT Press, 1996.

[12] D.Divsalar and R. J. McEliece, "The effective free distance of turbo codes ," Electronic, lett., vol. 32, no. 5, Feb. 1996, pp.445-446.