2003

Convolutional Interleaver for Unequal Error Protection of Turbo Codes

Sina Vafi
University of Wollongong, sv39@uow.edu.au

Tadeusz A. Wysocki
University of Nebraska-Lincoln, wysocki@uow.edu.au

Ian Burnett
University of Wollongong, ianb@uow.edu.au

Convolutional interleaver for unequal error protection of turbo codes

Sina Vafi, Tadeusz Wysocki, Ian Burnett

University of Wollongong, NSW 2522, Australia
E-mail:{sv39,wysocki,ian_burnett}@uow.edu.au

Abstract: *This paper describes construction of a convolutional interleaver as a block interleaver and discusses its application to turbo codes with equal and unequal error protection techniques. Based on simulations, different convolutional interleaver structures suitable for turbo codes with unequal error protection capability are suggested. Finally, based on conducted simulations the best method is selected.*

Keywords: Convolutional interleaver, Unequal error protection, turbo codes.

1. INTRODUCTION

Compression has become the norm in mobile communications; currently speech dominates but video and general multimedia compression is fast becoming important. A vital characteristic of a compressed data stream is the unequal perceptual importance of the compressed parameters. In turn, this leads to unequal effects of errors during transmission. Thus Unequal Error Protection (UEP) has become an important part of design of channel encoders for multimedia content.

Amongst the known channel codes, turbo codes have the highest performance in direct bit error rate reduction terms. Turbo codes are produced by the parallel concatenation of two or more convolutional code generators separated by one or more interleavers [1]. The reduction in BER resulting from such codes is directly dependent on the design of a suitable interleaver. In the literature, several techniques for implementing UEP using turbo codes have been proposed. In [2], single interleaver for all levels and, in [3], one interleaver for each level are suggested, while [4] discusses optimization of the single interleaver solution.

The interleavers suggested in [2-4] are from the block interleaver family. In a block interleaver, data bits are written into the memory column-wise and read row-wise (or vice versa). Another type of the block interleaver is a semi random interleaver, where after filling the whole interleaver memories the bits are read in a semi random manner, not row- or column-wise. Both types of interleavers are used with turbo codes and the total delay at the end of interleaving process is twice of the input data length. The non-block interleavers (such as the convolutional interleavers [5]) have better performance in terms of BER reduction during turbo decoding process. Additionally, they introduce a shorter delay, which simplifies the de-interleaving procedure. It is also important to note that convolutional interleaver performance is data dependent and that they allow for a continuous operation, which is a vital consideration for turbo codes.

In this paper, we introduce method that returns the memories of applied convolutional interleaver to the known state at the end of each data block, resulting in the convolutional interleaver performing effectively as a block interleaver. Therefore, the interleaved block-by-block data are independent from each other, which enable to use the conventional iterative decoding methods in the decoder.

The organization of the paper is as follows:
In Section 2, the structure of turbo codes and the position of the interleaver are described. In Section 3, the performance of a new convolutional interleaver with the calculation of free distance parameter of turbo codes is presented. In Section 4, some structures of interleaver suitable for UEP in turbo codes are described. Section 5 concludes the paper.

2. TURBO CODES STRUCTURE

Turbo encoders consist of two or more Recursive Systematic Convolutional (RSC) encoders separated by one or more interleavers. Figure 1 shows a turbo encoder (rate 1/3) structure with two RSC encoders.

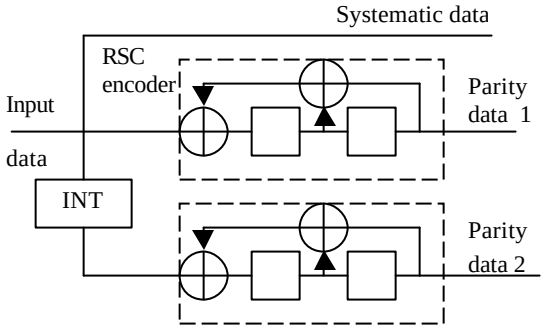

Figure 1: Turbo encoder structure

The input and RSC coded data are referred to as systematic and parity data, respectively (see Figure 1). The interleaver permutes the input data prior to the second RSC encoder to generate differently encoded data compared with the codeword of the first RSC encoder [6]. If a trellis termination or truncation is performed on RSC encoders and block interleaver is used, the turbo code performance can be evaluated as that of one block code (L(n+m), kL) where L, m and k/n represent the interleaver length, the number of RSC encoder memories or encoder states, and the code rate, respectively [6].

In this paper, the trellis termination is performed on the first RSC encoder, which returns its memory contents to zero state, while trellis truncation is performed on the second RSC encoder that leaves its memory states open.

As it has been shown in [7] with an employment of convolutional interleavers, which create less delay than block interleavers and maintain synchronization between the interleaver and deinterleaver, turbo codes performance improves. Moreover, utilization of the continuous behaviour in the stream-oriented turbo codes is suitable for long data length with RSC encoders having high number of states [8-9]. In order to use the convolutional interleaver in turbo codes equivalent to a block code, the interleaver memories should be reset at the end of each block to the specific value to create segmented interleaved data for the second RSC encoder. The new scheme can be used in turbo codes with unequal error protection property and the proposed technique is described in the next sections.

3. CONVOLUTIONAL INTERLEAVER STRUCTURE

A convolutional interleaver consists of T parallel lines, with different delay in each line that represents interleaver period [5]. In general, each successive line has a delay which is M symbols durations higher than the previous line. The structure for M=1 and T=3 is illustrated in Figure 2.

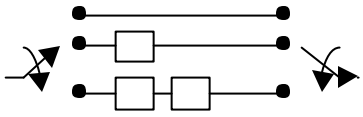

Figure 2: Convolutional interleaver structure with T=3 and M=1.

In a similar manner to the trellis termination technique (which emits stored data from memories), an insertion of a certain number of zeros at the end of a data block can isolate the interleaved data block. This guarantees a data block of a specific length and generates independent blocks of data for the RSC encoder.

For the data length L=6 and the convolutional interleaver with M=1 and T=3, the interleaved data would be X1 0 0 X4 X2 0 0 X5 X3 0 0 X6.

Since the application of convolutional interleaver in turbo codes requires an insertion of stuff bits that reduces channel bandwidth usage, an optimisation should be performed on the interleaver to control the number of those bits, which can be equal to the number of applied memories. For this purpose one block is added after the interleaver to control the data at the interleaver output deleting extra zero stuff bits that are inserted at the end of each block.

In this case, the memory contents at the end of each block have zero value that stays till the beginning of the next block. Figure 3 shows the structure of the optimised convolutional interleaver. Following the previous example, the optimised interleaver output is: X1 0 0 X4 X2 0 X5 X3 X6.

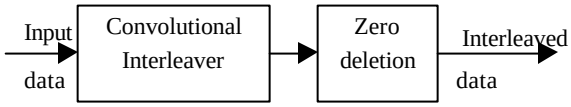

Figure 3- Optimised Convolutional Interleaver.

The same procedure can be repeated to obtain a second set of suitably interleaved data for the second RSC encoder. More optimisation can be done to the zero stuff bits insertion for the systematic and the first parity data after the trellis termination procedure of the first RSC encoder, as it is not necessary to transmit zero stuff bits in the mentioned data parts. However, these bits need to be considered for the second RSC because of their influence on the performance of the interleaver.

To verify performance of the new convolutional interleaver and compare it to conventionally employed interleaver such as row-column interleavers, the free distance parameter (d_{free}) of the turbo encoder of Figure 1 was calculated. Based on this parameter, the Bit Error Rate (BER) for additive white Gaussian noise (AWGN) can be expressed as: [10]

$$BER \approx \frac{N_{free}\,\omega_{free}}{2L}\,\mathrm{erfc}\left(\sqrt{d_{free}\,R\,\frac{E_b}{N_0}}\right) \quad (1)$$

where N_{free} is the number of codeword with weight d_{free}, w_{free} is the average weight of the data sequences with weight d_{free}, L is the interleaver length, $\frac{E_b}{N_0}$ is the signal to noise ratio per bit and R is the code rate. In [11] and [12] two different algorithms for free distance computation of turbo codes with different RSC encoder structures have

been proposed, which are mainly useful for large interleaver size.

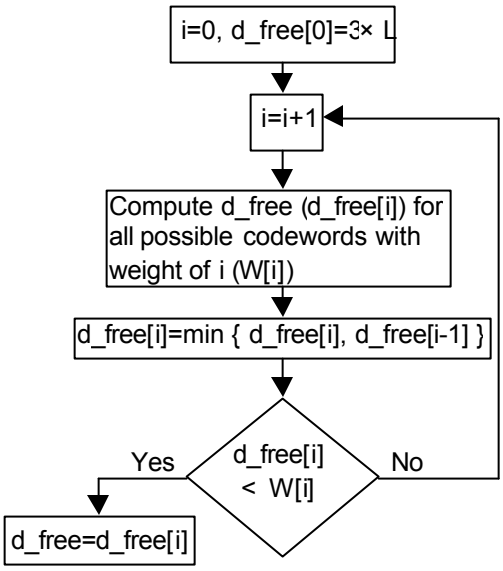

Figure 4: Free distance computation algorithm

Since in UEP turbo codes technique input data is segmented to shorter lengths, we can compute d_{free} from the definition. This involves finding a codeword with minimum weight among 2^L possible codewords of length L. Among all 2^L cases, only codewords that create low weights have been considered. Since input data are transferred directly to the encoder output as systematic bits and puncturing is not performed on them, weight of the input data corresponds to the weight information part of codeword. It should be noted that the tail bits due to trellis termination of the first RSC are considered as a part of the systematic data, which affects the codeword weight.

Algorithm starts d_{free} calculation for all possible data streams of length L and minimum weight, i.e. one. Then, the corresponding codeword weights are calculated. The obtained minimum weight from the first codeword set is considered as free distance at the first step. In the next step, input stream weight is increased by one unit, i.e. stream with weight of 2 at the second step, and their free distance is computed. The minimum d_{free} value among two relevant computed values is considered as d_{free} at the end of second step. Again, d_{free} for the stream with one unit higher weight, i.e. weight 3 is computed and the minimum d_{free} between previous and current calculation is selected.

This procedure is continued till the computed free distance in the related step is less or equal to the weight of the current input stream. The obtained d_{free} at the end of the procedure is considered as d_{free} of the turbo code. Figure 4 shows the applied d_{free} computation algorithm.

Based on the described procedure, d_{free} has been calculated for the turbo codes implemented by the structure presented in Figure 1 with the row-column and convolutional interleavers of the length of 16 and 64 and the rate of 1/3. The plot of the calculated BER versus $\frac{E_b}{N_0}$ for the mentioned interleavers is presented in Figure 5.

For the low signal-to-noise ratio (SNR) the convolutional interleavers have better performance than the block ones. In order to improve the performance for high SNRs, interleaver structure with longer period is suggested and discussed in details in the next section. It should be noted that increasing the period and data separation increases the number of stuff bits which can potentially reduce usage of the channel bandwidth. Data separation into shorter blocks has the same disadvantage, too. Thus, increasing the period and data separation should be done in a reasonable way minimising the overhead but maintaining the advantages.

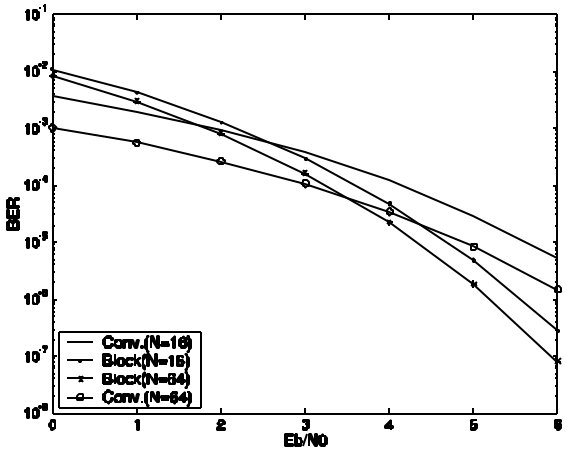

Figure 5: BER for convolutional and block interleaver sizes of 16 and 64 with rate: 1/3.

4. INTERLEAVER FOR UNEQUAL ERROR PROTECTION

For turbo codes with unequal error protection, the fixed interleaver period is considered for all levels, and, depending on the required protection level, the length of the data block at each level is altered. As an example, we consider three protection levels for data with length of 2048 bits. Specifications of these levels have been presented in Table 1. We assume that 2048 bits are split into 5 groups.

At the beginning of each group, 64 bits of data are allocated the protection level 1. Then, there are some blocks of length L=42 with the protection level 2, and the remaining blocks of length L=32 are allocated the protection level 3.

Unequal error protection is obtained by puncturing encoded stream differently at different protection levels.

Part	Lev. 1 (L=64)	Lev.2 (L=42)	Lev.3 (L=32)
1	1	3	6
2	1	3	6
3	1	3	7
4	1	3	7
5	1	4	7

Table 1: Distribution of 2048 bits into three protection levels.

The insertion of stuff bits to clear the interleaver memories affects the effective code rates, of course. Therefore, the effective code rate at each level can be calculated as:

$$R_{\text{eff }i} = \frac{L_i}{(L_i + S_i)} R_i \quad (2)$$

Where L_i, S_i denote lengths of data block and the number of stuff bits, respectively, while $R_i = \frac{k_i}{n_i}$, i=1, 2, 3 is the code rate after puncturing at the relevant protection level. Hence, for the code rates of 1/3,1/2, and 2/3, because of bit stuffing, we get the effective code rates of 0.32,0.47, and 0.61, in the considered example (see Table 1) . In order to verify UEP turbo codes performance, an average rate for the mentioned example is calculated. With employment of puncturing at each level, the average code rate can be calculated as [13]:

$$R_{av} = \frac{\sum_{i=1}^{c} k_i}{\sum_{i=1}^{c} \frac{k_i}{R_i}} \quad (3)$$

Where, c represents the number of levels. Based on the data in Table 2 the effective aggregate rate is determined as: $R_{\text{eff}_{av}} = \frac{2048}{2048 + 162} \times 0.53 \approx .49$

Level	Length (bits)	Code rate	Effective Code rate
1	320	1/3	.32
2	672	1/2	.47
3	1056	2/3	.61
Average	2048	.53	.49

Table2: Specification of three protection levels.

A similar calculation can then be performed to obtain the specification for average block length of the interleaver. Taking into account the number of blocks and block lengths at each level, the average length is given by:

$$L = \frac{\sum_{i=1}^{c} L_i N_i}{\sum_{i=1}^{c} N_i} \quad (4)$$

Where L_i and N_i ; i=1, 2, 3 are the length of data block and the number of blocks at each level, respectively. Hence, the average length of turbo encoder is $L \approx 41$ bits. Since an interleaver with 3 lines and 3 stuff bits has been considered, the length of the average data per block is approximately: $L_{data} \approx 41 - 3 = 38$ bits.

Based on the presented d_{free} in Table 3 for data distribution of Table 1, Fig.6 gives the BER performance of the turbo codes at different levels of protection. At level 1 with the lowest number of data bits, we expect that the protection is better than at other levels. This happens for low SNRs but at high SNRs it is similar to that of level 2. The level 2 protection at low SNRs has weaker performance than the overall performance of the code. This is caused by the multiplicity of free distance in level 2 due to its data length and the puncturing.

	Lev.1	Lev.2	Lev.3	Overall
d_{free}	6	5	2	3
N_{free}	1	18	2	1
w_{free}	3	3	2	3

Table 3- Free distance specifications with insertion of stuff bits after tail bits to the interleaver.

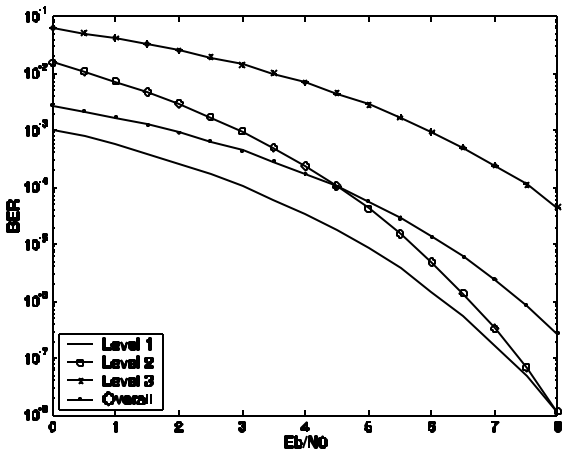

Figure 6: BER for UEP turbo codes with the interleaver of Figure 2 and insertion of stuff bits after the tail bits.

In order to remove degradations in the mentioned parts of level 1 and 2, their free distances

should be increased. One solution is to consider the effect of stuff bits in the systematic and the first coded parity data. In this case, trellis termination is

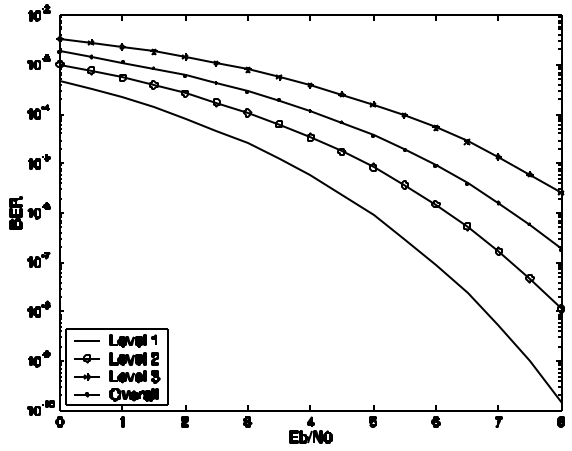

Figure 7: BER for UEP turbo codes with the interleaver of Figure 2 and insertion of stuff bits before tail bits.

done after the entry of stuff bits to the first RSC encoder. In order to consider tail bits effect for the second RSC encoder and to avoid changing of the interleaver memory states, which are in the reset condition, the tail bits are transferred through the first line of interleaver that has no memory.
Figure 7 shows simulation results for the identical data structure of previous example to verify performance of the new interleaver based on relevant computed d_{free} in Table 4.

	Lev.1	Lev.2	Lev.3	Overall
d_{free}	8	4	2	3
N_{free}	1	1	2	1
w_{free}	3	2	2	2

Table 4- Free distance specifications with insertion of stuff bits before tail bits to the interleaver.

Another method (referred here to as method 2) of achieving UEP is to consider a variable period interleaver which alters the number of stuff bits at different levels. Figure 8 shows a convolutional interleaver configured according to this method. In this example, the number of blocks at each level is the same as that of Table 1. The data requiring the highest protection level (level 1) is distributed on 5 lines of the interleaver with 6 stuff bits. Meanwhile, level 2 and 3 data are distributed on 4 and 3 lines of the interleaver, respectively.

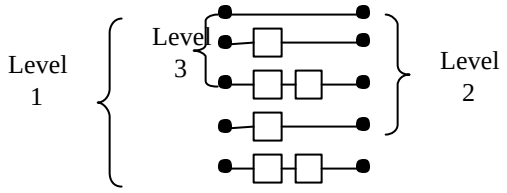

Figure 8: Interleaver structure in method 2.

Again, two alternative schemes using the new structure, i.e. insertion of stuff bits after or before the tail bits are considered and their simulation results compared with the fixed period interleaver for all the levels. Table 5 and Figure 9 show the computed d_{free} and the performance for protection levels when stuff bits are inserted after tail bits, respectively.

Considering the different stuff bits length of different levels, average value of stuff bits for overall level has been calculated, which approximately equals 3.

	Lev.1	Lev.2	Lev.3	Overall
d_{free}	7	5	2	3
N_{free}	1	4	2	1
w_{free}	2	2.25	2	3

Table5- Free distance specifications with insertion of stuff bits after tail bits to the interleaver.

In this scheme d_{free} in level 2 occurs multiple times but its performance at low SNRs is equal to that of the overall performance of the code. Also at level 1, degradation is observed at high SNRs. With insertion of stuff bits before tail bits, an improvement is obtained at relevant protection levels, as illustrated in Figure 10. The resulting free distances have been presented in Table 6.
In comparison with the previous method, level 2 has been improved near to the level 1 graph in Figure 8 while level 1 protection shows better performance than the other levels. Method 2 has improved performance but at the expense of an increase in stuff bits and hence redundancy level. Thus, a compromise needs to be found between optimisation of turbo code performance and the number of utilized stuff bits.
As one optimisation, combination of two mentioned schemes is suggested. For instance, in method 2 and level 1, we apply zero stuff bits insertion before tail bits to get more protection. But in level 2 we insert them after the tail bits that have acceptable behaviour with less redundancy. Also simulation shows that the level 3 protection performance of turbo codes in both sub-methods are identical, which can be due to puncturing effect at short data length. Therefore, for this level, format of stuff bits insertion after tail bits is recommended.

5. CONCLUSIONS

In this paper, we proposed a technique, which initializes convolutional interleaver memories by the insertion of stuff bits at the end data of blocks of specific lengths. This process results in a semi-block

interleaver structure for turbo coding. The performance of this new structure has been compared with that of conventional block interleavers and applied to unequal error protection turbo codes. Two major methods were proposed and tested. In each method, two schemes depending on insertion of stuff bits before or after the tail bits were investigated. The results show that a stuff bit insertion before the tail bits in the interleaver with variable period will produce the best performance. Further work will consider finding a suitable compromise between the turbo-code performance and the stuff-bit redundancy.

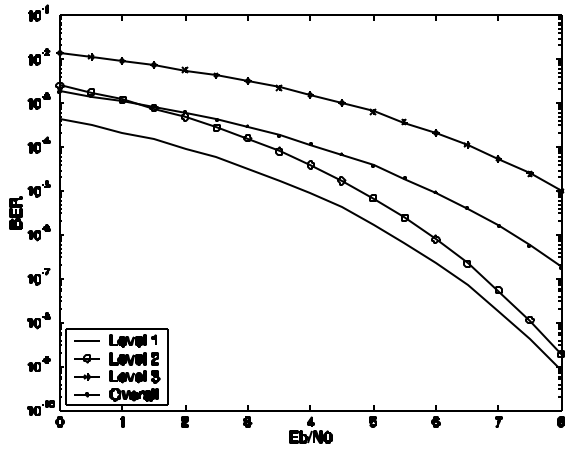

Figure 9: BER for UEP turbo codes with interleaver of Figure 8 and insertion of stuff bits after tail bits.

	Lev.1	Lev.2	Lev.3	Overall
d_{free}	9	5	2	3
N_{free}	1	2	2	1
w_{free}	3	2.5	2	2

Table 6- Free distance specifications with insertion of stuff bits before tail bits to the interleaver.

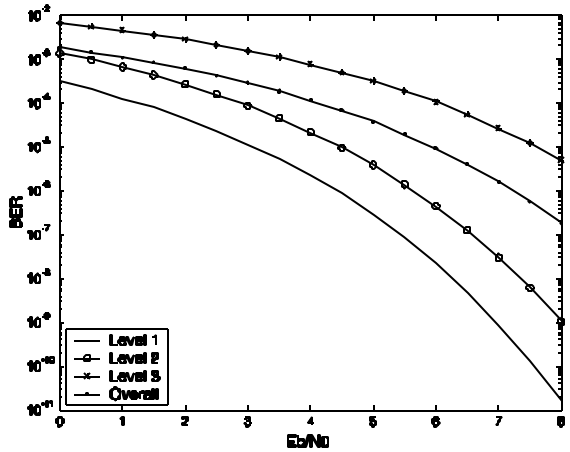

Figure 10: BER for UEP turbo codes with interleaver of Figure 8 and insertion of stuff bits before tail bits.

REFERENCES

[1]- C.Beerrou, A.Glavieux, P.Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo codes", *ICC 1993*, pp.1064-1070, May 1993.

[2]- A.Barbulescu, S.S.Pietrobon," Rate compatible turbo codes", *IEE Electronics letters*, vol.31, no.7, pp. 535-536, 30th March 1995.

[3]- M.Salah,R.A.Raines, A.Temple, T.G.Bailey," A general interleaver for equal and unequal error protections of turbo codes with short frames", *International conference on information technology: Coding and computing 2000*, pp-412-415.

[4]- M.Grangetto, E.Magli, G.Olmo, "Embedding Unequal Error Protection into turbo codes", *35th Asilomar conference on Signals, Systems and Computers*, 2001, vol.1, pp. 300-304.

[5]- G.D.Forney,"Burst-correcting codes for the classic bursty channel," *IEEE Trans. Commun*, vol.COM-19, pp. 772- 781, Oct.1971.

[6]- S.Dolinar, D.Divsalar, "Weight distributions for turbo codes using random and non-random permutations", *TDA progress report* 42-122, pp.56-65, Aug,15,1995.

[7]- E.K.Hall, G.Wilson, "Stream-oriented turbo codes", *IEEE Trans Inform. Theory*, vol.47, no.5, pp.1813-1831, July 2001.

[8]- S.Benedetto, G.Montorsi," Performance of continuous and blockwise decoded turbo codes", *IEEE trans. Letters*, vol.1,no.3, pp.77-79,May 1997.

[9]- M.Breliling,L.Hanzo," The super-trellis structure of turbo codes", *IEEE Trans. On inform.Theory*, vol.46,no.6,pp.2212-2228, Sep 2000.

[10]-L.C.Perez, J.Seghers, D.J.Costello, "A distance spectrum interpretation of turbo codes", *IEEE Trans Inform Theory*,vol.42,pp. 1698 - 1709, Nov.1996.

[11]- J.Seghers, " On the free distance of turbo codes and related product codes" Final Rep. Diploma project ss195,no 6613,Swiss Federal Institute of Technology, Zurich, Switzerland ,Aug. 1995.

[12]-R.Garello, P.Pierleni, S.Benedetto," Computing the free distance of turbo codes and serially concatenated codes with interleavers: Algorithms and applications, IEEE Journal on selected areas in commun., vol.19, no.5,May 2001,pp.800-812.

[13]-G.Caire, E.Biglieri, ”Parallel concatenated codes with Unequal Error Protection”, *IEEE Trans .Commun*, vol.46, no .5, pp.565-567, May 1998.