UNIVERSITY
OF WOLLONGONG
AUSTRALIA

University of Wollongong
Research Online

Faculty of Engineering - Papers (Archive)

Faculty of Engineering and Information Sciences

2007

Weight distribution of turbo codes with convolutional interleavers

S. Vafi
University of Wollongong, sina@uow.edu.au

T. Wysocki
University of Wollongong

http://ro.uow.edu.au/engpapers/436

Publication Details
This article was originally published as Vafi, S, and Wysocki, T, Weight distribution of turbo codes with convolutional interleavers, IET Communications, 1(1), 2007, 71-78. Copyright 2007 IEEE.

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Weight distribution of turbo codes with convolutional interleavers

S. Vafi and T. Wysocki

Abstract: A simple algorithm for the weight calculation of turbo codes with convolutional interleavers is presented. For codes with short interleaver lengths, the weight distributions are computed using conventionally proposed methods and then utilised together with the interleaver properties to determine the weight specifications for the code with a longer desired length. Based on the calculated weights, a new upper bound for the code is computed. It agrees with simulation results of the code performance in the error-floor region.

1 Introduction

Determining weight distribution is well-known as an effective method to verify turbo code performance. The main issue of its calculation is related to the interleaver structure, which permutes the input bitstreams so as to prohibit generation of low weights for the second recursive systematic convolutional (RSC) codes. Fig. 1 shows a simple structure of turbo codes.

Different algorithms have been proposed to estimate the weight distribution of turbo codes based on different applied interleaver types. In the work of Perez et al. [1], an algorithm considers those input bitstreams that may possibly generate low-weight codewords. It basically computes a few terms of weight distribution and is useful for a code that has free distance with high multiplicities determining the code performance in the error-floor region. The complexity of this method increases with increasing interleaver lengths. A general algorithm was proposed in Pierleoni *et al.* [2] to compute low-weight terms of the code with a moderate length. This algorithm defines the constrained subcode based on the trellis structure of the code and computes the free distance of the code on the basis of an obtained minimum distance of a subcode relevant to the interleaved data. Some improvements to this algorithm, which reduce computation complexity are suggested by Rosnes and Yterhus [3], while Yeh *et al.* [4] and Huebner and Costello [5] present other algorithms for the given interleavers.

The best performance of turbo code is achieved by using random interleavers, which randomly permute input bitstreams to different memories of the interleaver. Due to the existence of randomly interleaved data, determining an adequate analysis for this implementation is a major obstacle. In addition, to achieve synchronisation between random interleavers and deinterleavers, which perform the reverse function to interleavers, it is necessary to store interleaved data in the memory. This is not desirable in some applications when the length of input bitstream is large. Considering these issues, finding good deterministic interleavers to create similar performance to the random interleavers has been widely investigated in previous research.

In contrast to block interleavers, nonblock interleavers are designed with a lower number of memories and often with a self-synchronisation property for the deinterleavers to reduce the design complexity. This can be considered as one of their advantages. Convolutional interleavers are introduced as the most well-known nonblock interleavers. These interleavers have more flexibility to adjust their specifications according to the variable lengths of data block. This property of the convolutional interleaver simplifies the design of turbo codes with UEP application, which protect different parts of data block at different rates [6]. However, considering the application of these interleavers in turbo codes, the performance of the code is accomplished by using the continuous form at the expense of analysis complexity at the decoder.

The advantage of convolutional interleavers in turbo codes can be utilised when they emulate block operation. This is easily achieved when some stuff bits are inserted at the end of each data block forcing the interleaver memories into the known state, which is usually considered as the zero state [7]. This operation simplifies the code analysis and decoding procedure. The disadvantage of this procedure is a reduction in the available channel bandwidth due to insertion of stuff bits. Some optimisations can be performed on the interleaver structure to decrease the number of stuff bits at the encoder output.

For turbo codes with optimised convolutional interleavers, a simple algorithm is presented here to effectively compute the free distance of the code. The algorithm is implemented based on input bitstreams that return the first RSC encoder to the zero state. These patterns are known as self-terminating patterns. The algorithm only considers those self-terminating patterns whose weights, that is, bits having value 1, are fully located in the end part of the interleaved data [8]. However, to precisely verify the code behaviour, it is necessary to find other input bitstreams with weight outside the mentioned range, which produce code weights with high multiplicities that affect the code performance.

Another similar algorithm is presented based on convolutional interleaver properties. It starts by determining the weight of the code with short interleaver lengths and then extrapolates those results for the longer length. This applied property dramatically simplifies the code analysis used to calculate weight of the code for high interleaver length, where processing of all input bitstreams is long or

© The Institution of Engineering and Technology 2007

doi:10.1049/iet-com:20060007

Paper first received 30th September 2005 and in revised form 4th July 2006

The authors are with University of Wollongong, Northfields Ave., NSW 2522, Australia

E-mail: sv39@uow.edu.au

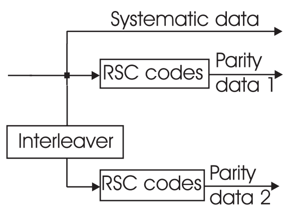

Fig. 1 *Structure of turbo codes*

even in some cases impossible. The calculated upper bounds of the probability of error for turbo codes with different interleavers confirm that the new algorithm gives a good approximation of the code performance in the error-floor region. Both algorithms are utilised for turbo codes without the puncturing of parity bits.

2 Convolutional interleavers structure

Convolutional interleavers are introduced as nonblock deterministic interleavers that were investigated in some communication systems due to applying less memories in their structures compared with the block interleavers [9, 10]. These interleavers are constructed by T parallel lines, which define their period. Conventionally each interleaver line has a different number of memories from other lines. The difference in the numbers of memories between two adjacent interleaver lines is generally considered as a constant referred to as a space parameter of the interleaver. Fig. 2*a* illustrates the general structure of convolutional interleavers with period $T = 4$ and space value $M = 1$. Based on the arithmetic sequence, the overall number of memories for the interleaver (T, M) is given by [11]

$$S = \sum_{i=1}^{T} s_i = M + 2M + \cdots + (T-1)M = \frac{T(T-1)M}{2} \tag{1}$$

A convolutional interleaver can be forced to operate as a block interleaver by inserting number of zero stuff bits to its memories, providing an interleaved data block which is isolated from the other blocks. Initially, interleaver memories are set to the zero value. Based on the number of applied memories in each line, bits distributed to the relevant line appear at different times. The interleaved data block of an input bitstream with the length $L = 24$ from the interleaver $(T = 4, M = 1)$ is shown in Fig. 2*b*.

Depending on the input sequence length and the interleaver period, distributed data is terminated at one of the interleaver lines, which is determined by the Rem(L, T) value, where Rem(L, T) gives the remainder of the L/T operation. Considering an interleaver $(T = 3, M = 1)$, an input bitstream with length L, can be interleaved as one of the following schemes:

Rem$(L, T) = 0$
$\{x_0, 0, 0, x_3, x_1, 0, x_6, \ldots, x_{L-7}, 0, x_{L-2}, x_{L-4}, 0, 0, x_{L-1}\}$
Rem$(L, T) = 1$
$\{x_0, 0, 0, x_3, x_1, 0, x_6, x_4, \ldots, x_{L-3}, x_{L-5}, 0, 0, x_{L-2}\}$
Rem$(L, T) = 2$
$\{x_0, 0, 0, x_3, x_1, 0, x_6, \ldots, x_{L-4}, x_{L-6}, 0, x_{L-1}, x_{L-3}\}$

When the convolutional interleaver is applied as a constituent of turbo codes, inserted stuff bits to the interleaver memories reduce channel bandwidth usage. Therefore an optimisation can 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 controlling the data in the interleaver output delete extra zero stuff bits that appear in the end part of the interleaved data. In this case the memory contents in the end of each block have zero value that stay until the beginning of the next block. Following the previous example, the optimised interleaver outputs for different Rem(L, T) values are given by

Rem$(L, T) = 0$
$\{x_0, 0, 0, x_3, x_1, 0, x_6, \ldots, x_{L-2}, x_{L-4}, x_{L-1}\}$
Rem$(L, T) = 1$
$\{x_0, 0, 0, x_3, x_1, 0, x_6, x_4, \ldots, x_{L-3}, x_{L-5}, x_{L-2}\}$
Rem$(L, T) = 2$
$\{x_0, 0, 0, x_3, x_1, 0, x_6, \ldots, x_{L-4}, x_{L-6}, x_{L-1}, x_{L-3}\}$

Fig. 2*c* shows the optimised interleaved data constructed from the nonoptimised interleaver $(T = 4, M = 1)$.

For every interleaver, input bitstreams can be categorised to T different groups such that each group includes bitstreams with different lengths having the same Rem(L, T) value. In this case some parts of an interleaved data with the shorter length appear in the interleaved data with the higher lengths. Fig. 3 shows these mentioned parts for two bitstreams with the length $L = 20$ and $L = 24$ considered as one group for the interleaver $(T = 4, M = 1)$ which gives the Rem$(L, T) = 0$ value.

3 Weight distribution algorithm for turbo codes with convolutional interleavers

The analysis of turbo codes based on weight-2 distribution confirms that if interleavers increase the distance between two adjacent bits of an input bitstream, higher weights

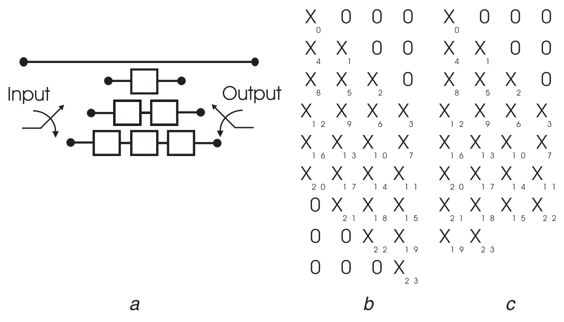

Fig. 2 *Convolutional interleaver structure*

a Interleaver with period $T = 4$ and space value $M = 1$
b Block interleaved data with length $L = 24$
c Optimised interleaved data with length $L = 24$

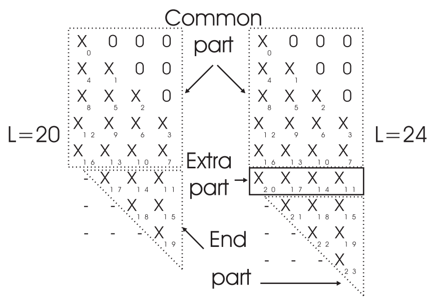

Fig. 3 *Comparison of different parts of interleaved data in output of interleaver with different lengths and similar period and Rem(L, T) values, that is, T = 4, Rem(20, 4) = 0, Rem(24, 4) = 0*

for the code are achieved. This consequently improve the code performance [12]. In optimised convolutional interleavers the distance between two arbitrary adjacent bits of the input bitstream located in the end part of interleaved data is shorter than elsewhere due to the deletion of the zero stuff bits in the end part of the interleaver. For example, for the interleaved data illustrated in Fig. 2*b* and *c*, the distance between x_{18} and x_{19} bits before and after optimisation is equal to 4 and 2, respectively. Therefore in a turbo code with an optimised interleaver, it is expected that low weights of the second-parity codewords are generated from input bitstreams whose weights are located in the end part of the interleaved data.

Considering this structure for the optimised convolutional interleaver, an algorithm can be presented that computes the free distance and some low weights of the code. It is assumed that only the first RSC encoder is terminated to the zero state. Hence the effect of tail bits due to this termination weight is considered in the algorithm. The algorithm is implemented as follows [8].

First, among all input data block streams with the minimum weight (i.e. 1) those self-terminating patterns of the first RSC encoder are selected. Then their bit-1 positions are compared with the bit positions that were located in the end part of the interleaved data. If any pattern returns the first RSC encoder to the zero state and positions of its bits are in the end part of the interleaved data, the overall weight of the corresponding codeword is computed and stored as d_{free} value. A similar procedure is followed for higher input data weights until the computed d_{free} is lower than or equal to the weight of the input bitstream. The final d_{free} is assumed to be the d_{free} of the turbo code.

Since bit-1 positions should be located in the end part of the interleaved data, the pattern length consisting of all the ones inside the bitstream should not exceed the number of bits in the end part of the interleaver. In this case low-weight patterns with a length equal to the number of bits in the end part of the interleaver are encoded returning the RSC encoder to the zero state.

For example, for the interleaver ($T = 4$, $M = 1$) illustrated in Fig. 3, the number of bits located in the end part of interleaved data is six. Therefore the algorithm contributes self-terminating patterns with length six, whose weights are located in the mentioned part. Table 2 represents self-terminating patterns of with weights two and three for turbo codes (1, 5/7). The algorithm covers all patterns shifted cyclically that satisfy the condition.

Table 1: Self-terminating patterns with weight-2 and 3 for 4-state turbo code (1, 5/7)

Input weight	Pattern	Output weight
2	$1\underbrace{0\ldots0}_{d=3k+2}1$ $k=0,1,2,\ldots,\lfloor(L-4)/3\rfloor$	$2d+8/3$
3	$11\underbrace{0\ldots0}_{d=3k}1$ $k=0,1,2,\ldots,\lfloor(L-3)/3\rfloor$	$2(d/3)+1$
3	$1\underbrace{0\ldots0}_{d=3k}11$ $k=0,1,2,\ldots,\lfloor(L-3)/3\rfloor$	$2(d/3)+1$
3	$1\underbrace{0\ldots0}_{d=3k+1}1\underbrace{0\ldots0}_{d'=3k'+1}1$ $k=0,1,\ldots,\lfloor(L\text{-}4\text{-}d')/3\rfloor\ k'=0,1,\ldots,\lfloor(L\text{-}4\text{-}d)/3\rfloor$ $d+d'\leq L-3$	$2(d'+d3)+8/3$
3	111	2

Table 2: Free-distance specifications for turbo codes (1, 5/7) and (1, 35/23) with interleavers ($T = 10$, $M = 1$, $L = 512$) and ($T = 20$, $M = 1$, $L = 1024$)

Turbo code	T	L	d_{free}	N_{free}	$\tilde{w}_{free}$
(1, 5/7)	10	512	10	3	3
(1, 5/7)	20	1024	10	3	3
(1, 35/23)	10	512	11	1	3
(1, 35/23)	20	1024	16	1	3

The computed d_{free} values of turbo codes (1, 5/7) and (1, 35/23) for convolutional interleavers with different lengths were presented in Table 2, where N_{free} and $\tilde{w}_{free}$ represent the total number of multiplicities of the codewords with weight d_{free} and the average input data weight related to d_{free}, respectively. (The results were achieved by input self-terminating patterns with weight no greater than four.) The results show that for the four-state turbo code, increasing the period and length does not affect the free distance specifications, while for the 16-state code the free distance has been increased by five units and the multiplicity has been preserved.

In comparison with most block interleavers the convolutional interleaver generates d_{free} with fewer multiplicities, which can be considered as an advantage. This result expresses that it is necessary to determine other codewords with low weights or codewords having relatively high multiplicities, which affect the code performance.

Depending on the period of the optimised interleaver, the number of bits which are located in the end part of the interleaved data changes. To calculate low weights of the code for different interleaver periods the area of the end part of the interleaver can be increased or decreased based on the interleaver period. By increasing the area, more bits are involved in the calculation. Thus the input bitstreams with lower weights must be considered. Instead, shortening the area makes it possible to involve input bitstreams with

Table 3: Weight distribution for turbo codes (1, 5/7) and (1, 35/23) in end part of interleaver with ($T = 20$, $M = 1$, $L = 1024$) and Rem(L, T) = 4

Weight	Turbo code (1, 5/7)		Turbo code (1, 35/23)	
d	N_d	$\tilde{w}_d$	N_d	$\tilde{w}_d$
10	3	3.0	0	0
11	0	0	0	0
12	0	0	0	0
13	1	3	0	0
14	4	2.75	0	0
15	0	0	0	0
16	4	2.5	1	3
17	1	3.0	0	0
18	7	2.58	0	0
19	3	2.67	2	3
20	78	3.82	3	3
21	11	2.9	1	3
22	33	3.18	1	3
23	20	3	1	3
24	91	3.49	1	3
25	38	3.2	3	2.67
26	63	3.3	6	3

the higher weights [8]. Table 3 gives some low weights of the 4-state (1, 5/7) and 16-state (1, 35/23) turbo code, which have been calculated for ($T = 20$, $M = 1$, $L = 1024$) on the basis of 170 bits located in the end part of the interleaved data.

3.1 Extrapolated weight distribution computation algorithm

Apart from the end part of the interleaved data it is possible to find other input bitstreams having weights outside the mentioned area producing low weights that affect the code performance.

Without considering the end part of the interleaved data, the distance between distributed bits in adjacent interleaver lines is always fixed by the product of T and M. Due to the deterministic behaviour of the convolutional interleaver, it is possible to obtain self-terminating patterns for the second RSC encoder that have been interleaved from other or similar input self-terminating patterns. As a result, both RSC encoders simultaneously return to the zero state and hence low weights for the code are expected.

In turbo codes with different interleaver lengths, when all bits 1 of a self-terminating pattern are positioned in the common part of two interleaved data categorised as one group, increasing the length of patterns will not affect the weight increment of the second RSC code. Therefore both interleavers produce a weight with similar multiplicity. Otherwise, i.e. when some or all weights of this self-terminating pattern are positioned in the extra part of an interleaver with a higher length, multiplicity of the weight is progressively increased proportional to the length difference of two interleavers.

For example, in the interleaver ($T = 4$, $M = 1$) for lengths $L = 20$ and $L = 24$, $\mathrm{Rem}(L, T) = 0$ value (see Fig. 3), and bits 1 of the self-terminating pattern $(000100000000100\cdots0)_{L=L_1,L_2}$, that is x_3 and x_{12} are positioned in the common part of two interleavers. These interleavers generate another self-terminating pattern $(00000000000010010\cdots00)_{L=L_1,L_2}$ for the second RSC encoder. In this case both interleavers produce an identical weight for the second parity data. This indicates that increasing the interleaver length has no effect on the weight of the code.

Additionally, in optimised interleavers, the distance of bits located in the end part of the interleaved data with the last bit of data block is constant. For example, for the interleaver ($T = 4$, $M = 1$, $L = 24$) illustrated in Fig. 3, the distance between $x_{21} = x_{L-3}$ and $x_{23} = x_{L-1}$ would be four, which is the same as the equivalent bits for the interleaver with length ($T = 4$, $M = 1$, $L = 20$), that is between $x_{17} = x_{L-3}$ and $x_{19} = x_{L-1}$. Therefore the weights obtained from the self-terminating patterns positioned in the end part of the interleaver would be independent from the interleaver length L. In this case weights obtained from the end part of an interleaved data with the short length can be utilised as weights of the code with the higher interleaver length [8].

Based on these properties of convolutional interleavers, it is possible to estimate the weight distribution of a turbo code with a desired length from an interleaver with the shorter length. In this method, according to the interleaver period, the minimum length of interleaved data is varied and is equal to $T(T-1)M$ value, that is when all the interleaver memories have valid data. For the interleaver ($T = 4$, $M = 1$) with the presented specifications in Fig. 2, the minimum length is equal to 12.

Considering self-terminating patterns with the weight i, the algorithm computes $W_L^{(i)} = (w_{L_1}^{(i)}, w_{L_2}^{(i)}, \ldots, w_{L_k}^{(i)})$ with the multiplicity $N_L^{(i)} = (n_{L_1}^{(i)}, n_{L_2}^{(i)}, \ldots, n_{L_k}^{(i)})$ as weight specifications for the turbo code using an interleaver (T, M) with length L. This is accomplished as follows:

- Design an interleaver (T, M) with the shortest length L_1 that satisfies
$$T(T-1)M \le L_1 < L - T \text{ and } \mathrm{Rem}(L_1, T) = \mathrm{Rem}(L, T).$$
- Compute the weight distribution of the code for the interleaver (T, M, L_1) from all self-terminating patterns with the weight i, as
$$W_{L_1}^{(i)} = (w_{L_{11}}^{(i)}, w_{L_{12}}^{(i)}, \ldots, w_{L_{1k}}^{(i)}) \quad \text{and}$$
$$N_{L_1}^{(i)} = (n_{L_{11}}^{(i)}, n_{L_{12}}^{(i)}, \ldots, n_{L_{1k}}^{(i)})$$
- Increase the interleaver length T units ($L_2 = L_1 + T$) and compute the weight distribution specification of the code for the new interleaver length as $W_{L_2}^{(i)}$ and $N_{L_2}^{(i)}$, where
$$W_{L_2}^{(i)} = (w_{L_{21}}^{(i)}, w_{L_{22}}^{(i)}, \ldots, w_{L_{2k}}^{(i)}) \quad \text{and}$$
$$N_{L_2}^{(i)} = (n_{L_{21}}^{(i)}, n_{L_{22}}^{(i)}, \ldots, n_{L_{2k}}^{(i)})$$
- If $n_{L_{1j}}^{(i)} = n_{L_{2j}}^{(i)}$, then $n_{L_j}^{(i)} = n_{L_{1j}}^{(i)}$ $(j = 1, 2, \ldots, k)$ else
$$n_{L_j}^{(i)} = n_{L_{1j}}^{(i)} + \frac{L - L_1}{T}(n_{L_{2j}}^{(i)} - n_{L_{1j}}^{(i)})$$
- $W_L^{(i)} = W_{L_1}^{(i)} = W_{L_2}^{(i)} = (w_{L_1}^{(i)}, w_{L_2}^{(i)}, \ldots, w_{L_k}^{(i)})$
$$N_L^{(i)} = (n_{L_1}^{(i)}, n_{L_2}^{(i)}, \ldots, n_{L_k}^{(i)})$$

Table 4: Weight distribution of 4-state turbo code for two input bitstreams $(100100\cdots0)_L$, $(11100\ldots0)_L$ and their cyclical shifts for the interleaver ($T = 10$, $M = 1$) with different lengths and identical Rem(L, T) value

Pattern	$(100100\cdots0)_L$			$(11100\cdots0)_L$		
L	92	102	512	92	102	512
ω	N_ω	N_ω	N_ω	N_ω	N_ω	N_ω
10	0	0	0	3	3	3
11	1	1	1	0	0	0
12	0	0	0	0	0	0
13	2	2	2	1	1	1
14	1	1	1	2	2	2
15	2	2	2	0	0	0
16	3	3	3	1	1	1
17	2	2	2	1	1	1
18	5	5	5	2	2	2
19	2	2	2	0	0	0
20	5	5	5	2	2	2
21	6	6	6	37	45	373
22	7	7	7	2	2	2
23	4	4	4	0	0	0
24	1	1	1	0	0	0
25	9	9	9	1	1	1
26	46	53	340	2	2	2
27	2	2	2	1	1	1
28	5	5	5	2	2	2
29	9	9	9	0	0	0
30	46	53	340	2	2	2

Table 5: Combined 100011 pattern with other low-weight patterns of Table 2

Weight	Pattern
5	$100011\underbrace{00\cdots0}_{d''}1001 \quad d''=0,\ldots,L-10$
6	$100011\underbrace{00\cdots0}_{d''}100011 \quad d''=0,\ldots,L-12$
6	$100011\underbrace{00\cdots0}_{d''}110001 \quad d''=0,\ldots,L-12$
6	$100011\underbrace{00\cdots0}_{d''}101010 \quad d''=0,\ldots,L-12$
6	$100011\underbrace{00\cdots0}_{d''}111 \quad d''=0,\ldots,L-9$

Table 4 gives the calculated low weights of the four-state turbo code (1, 5/7) for the specified input bitstreams and the interleaver ($T = 10$, $M = 1$) with minimum lengths $L = 92$ and $L = 102$. The weights obtained from these lengths have been utilised to determine the weight of the code for the interleaver length $L = 512$. The results show that the code for weights $\omega = 26$, $\omega = 30$ and $\omega = 21$ has high multiplicities. These specifications are achieved from an input bitstream that simultaneously return both RSC encoders to the zero state.

These calculations indicate that multiplicities of some weights will remain constant for different interleaver lengths. These weights are mainly produced by input bitstreams with weights located in the end part of the interleaved data.

3.2 Weight distribution from higher input bitstreams weights

It is possible to combine low-weight self-terminating patterns with each other making other self-terminating patterns with a higher weight. This is accomplished by separating the original self-terminating patterns from each other with a number of zero bits. Due to increasing the weight of input bitstream, the number of new self-terminating patterns is increased in such a way that makes it impossible to consider all of them in the weight distribution of the code, even for the short interleaver lengths.

To get a codeword with a low weight, those self-terminating patterns that generate lower weights are combined. This leads to apply self-terminating patterns with the minimum acceptable number of zeros between 1s. The weight distribution of the code from these combined patterns can be calculated by the algorithm presented in the previous Section. Table 5 gives combinations of the (100011) pattern with other presented patterns of Table 2. The weight of the four-state turbo code $(1, 5/7)$ applying the combined patterns with weights four, five and six has been calculated and illustrated in Fig. 4. Fig. 4*a* to *e* show combinations of identical self-terminating patterns, while Fig. 4*f* gives weight distributions of the code due to combining different self-terminating patterns from Table 2. The results indicate that the weight with high multiplicities is created by identical patterns, while combination of different patterns give similar weights with low multiplicities. In conducted calculations, the distance between two basic patterns in every combined pattern d'' does not exceed 145.

3.3 Effect of tail bits on weight distribution of code

The algorithm should consider the effect of tail bits generated on the basis of the trellis termination of the first RSC encoder. Tail bits are always located in the last part of every systematic data block and they will be located near to or in the end part of the interleaved data. With this assumption, the number of patterns that can make low-weight codewords will be increased by increasing the length of the interleaver and constraint length of RSC codes. This may take place when bits 1 of the pattern are close to each other and positioned near to the end part of the input bitstream. If this condition is not satisfied, at least one RSC encoder provides higher weight, which

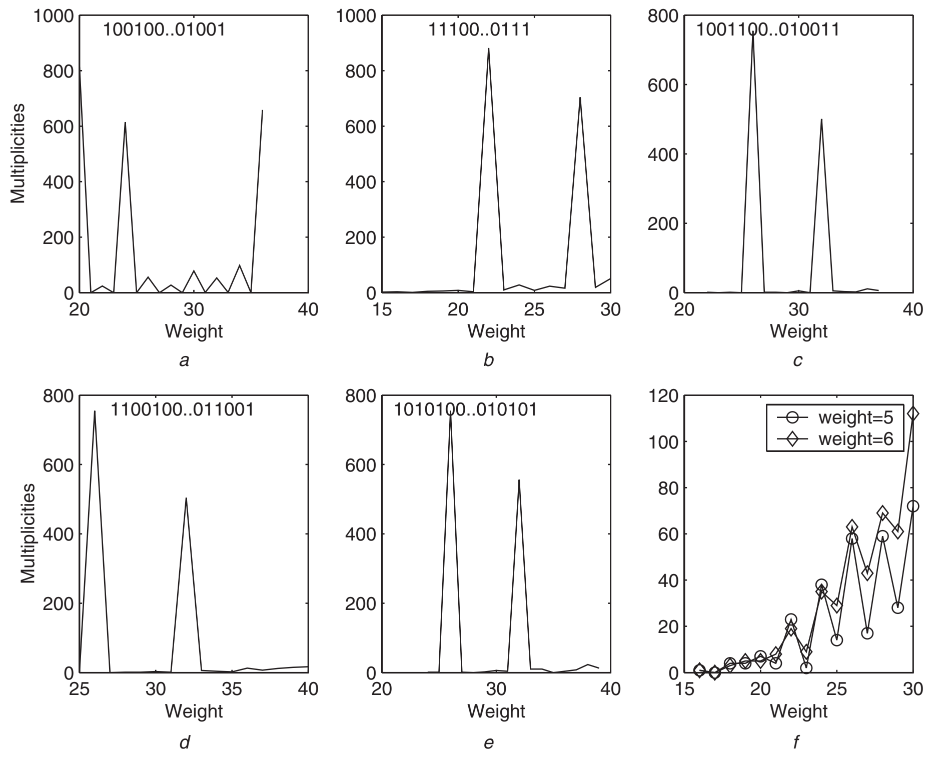

Fig. 4 *Weight distribution of turbo code (1, 5/7) using length L = 1024 and combined input bitstreams of Table 2*

Table 6: Effect of tail bits and bits 1 position of patterns on codeword weight of turbo code (1, 35/23)

Bit 1 position of input bitstream	Tail bits pattern	Codeword weight
(1016, 1017, 1018, 1019)	0010	26
(1015, 1016, 1017, 1018)	0010	29
(1014, 1015, 1016, 1017)	0010	31
(1019, 1020)	0111	22
(1020)	0011	16
(1010)	1001	57
(995)	1001	41

consequently increases the weight of the code. Table 6 shows the effect of tail bits and bits 1 position on the weights computed for the 16-state turbo code (1, 35/23).

4 Simulation results

The proposed algorithm is applied for determination of their BER upper bound values, which give significant results of the code performance in the error floor region [13]. For a maximum-likelihood (ML) decoding of this code in the presence of additive white gaussian noise (AWGN), its probability of error (BER) is upper bounded by:

$$\text{BER} \leq \sum_d \frac{N_d \tilde{w}_d}{L} Q\left(\sqrt{d\frac{2RE_b}{N_0}}\right) \qquad (2)$$

where R, E_b/N_0, N_d and $\tilde{w}_d$ denote the code rate, signal-to-noise ratio per information bit, number of multiplicities and average weight of information of weight d, respectively.

In the weight calculation, input self-terminating patterns with weight no greater than four have been considered. Trellis termination and truncation are applied for the first and second RSC encoders, respectively. Insertion of stuff bits to the interleaver memories is conducted after trellis termination of the first RSC encoders. Thus stuff bits have no effect on the weight of the systematic and first parity data and can be deleted from the mentioned data parts. For simplicity, stuff bits are considered to make the systematic and the first parity data with a length as long as the length of the second parity data. This leads the code rate 1/3 for the simulation and analysis. At the decoder, the decoding process of the first parity data is only accomplished for the original bitstreams minus stuff bits [7]. In simulations, the information received from AWGN channel is decoded by the soft output Viterbi algorithm (SOVA) [14]. Ten iterations for the first and third examples and 15 iterations for the second example have been considered. Again, the maximum distance between two low weight patterns in a combined pattern is set to 145.

Fig. 5 shows performance of the four-state turbo code (1, 5/7) with the interleaver ($T = 10$, $M = 1$) and length $L = 512$. For different interleaver lengths with identical Rem(L, T) = 2 value the algorithm computes the weight of the code. In this example the codeword weights for interleaver lengths $L = 92$ and $L = 102$ have been computed and then their results have been extrapolated for the desired length, i.e. $L = 512$. The algorithm gives the minimum distance $d_{free} = 10$ with w_{free}=3 and $N_{free} = 3$. The new upper bound gives more accuracy for the code performance for all signal-to-noise ratios. Among

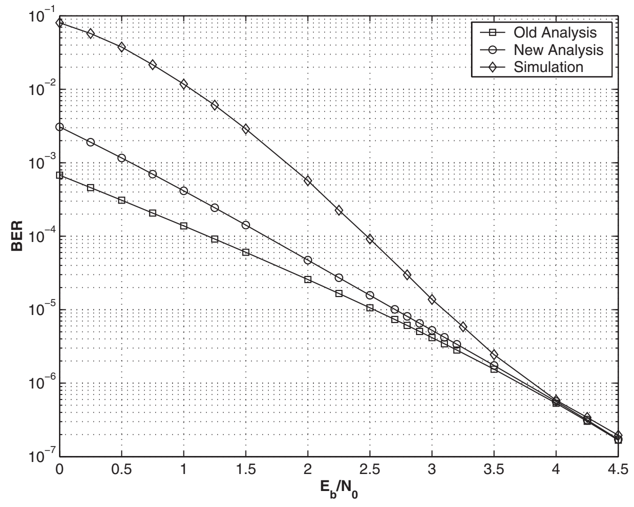

Fig. 5 *Analysis and simulation results of 4- state turbo code (1, 5/7) with interleaver (T = 10, M = 1) and length L = 512*

different combined low-weight patterns, the pattern $(0\cdots00100100\cdots01110\cdots0)_L$ produces the minimum weight with the following specifications: $d = 16$, $N_d = 2$, $\omega_d = 6$. In this code, other combined patterns generate weights with high multiplicities far from the free distance and other low weights, and are not affecting the code performance in the error-floor region.

Fig. 6 illustrates analysis of the code for the interleaver ($T = 20$, $M = 1$) and length $L = 1024$. In this example the code weight distribution is determined by the interleaver lengths $L = 382$ and $L = 402$. The result gives $d_{free} = 10$ with w_{free}=3 and N_{free}=3 as free distance specifications of the code. In the considered example, almost all of the low-weight codewords affecting the code performance are located in the end part of the interleaved data. However, many combined low-weight patterns were found that are not located in this part. This effect has been verified by the analysis of the code with and without the combined patterns. The relevant graphs illustrate about 0.2 dB difference in the error-floor region between the results obtained when these two approaches are taken. In addition, analysis of the code considering the combined patterns in the end of the interleaved data shows performance closer to the simulation results than the upper bound obtained without considering

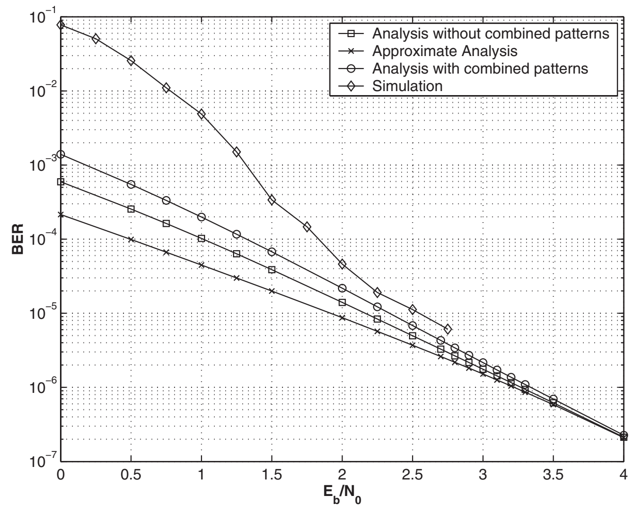

Fig. 6 *Analysis and simulation results of the 4-state turbo code (1, 5/7) with interleaver (T = 20, M = 1) and length L = 1024*

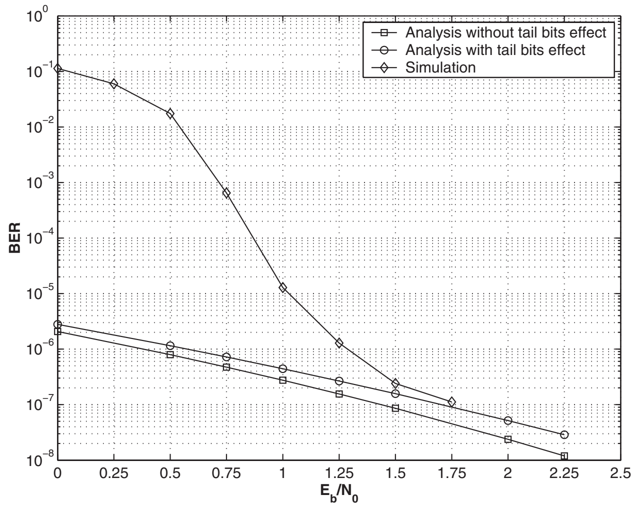

Fig. 7 *Analysis and simulation results of 16-state turbo code (1, 35/23) with interleaver ($T = 35$, $M = 1$) and length $L = 4096$*

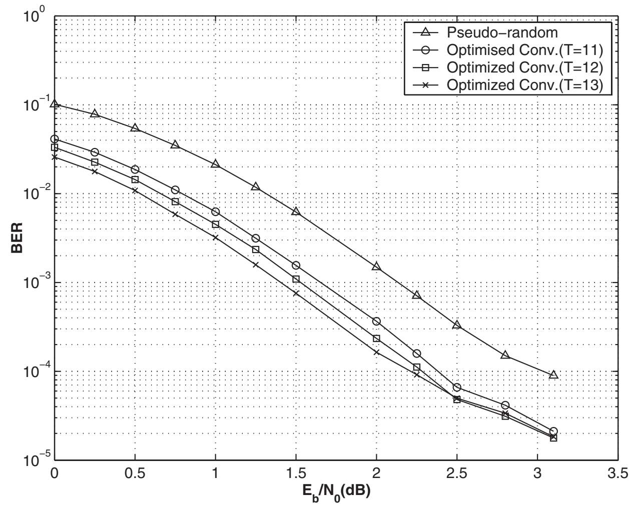

Fig. 8 *Performance of 4-state turbo codes (1, 5/7) with different interleavers and length $L = 169$*

the effect of combined patterns. The tail bits weight effect on the code performance has been confirmed in these above examples. The specifications of the minimum weight of the code is given by $(d,\ N_d,\ \omega_d) = (22,\ 1,\ 3)$ and $(d,\ N_d,\ \omega_d) = (26,\ 1,\ 2)$ values for the interleavers ($T = 10$, $M = 1$) and ($T = 20$, $M = 1$), respectively.

More verification has been conducted for the 16-state turbo code (1, 35/23) with the interleaver ($T = 35$, $M = 1$) and length $L = 4096$. For this purpose, low-weights of the relevant code are calculated from interleaver lengths $L = 1226$ and $L = 1261$. The algorithm gives some low weights from the combined low-weight input bitstreams that return both RSC encoders to the zero state. For the combined two 10011 patterns, that is $(00\cdots01001100\cdots0001001100\cdots0)_L$, the calculated minimum weight is 27 with two multiplicities. As expected, only a few low weights contribute to the code performance. In comparison with two previous examples the weight of the tail bits significantly affects the performance of these codes. Based on this assumption, the free-distance specifications of the code is calculated by $d_{free} = 16$, $N_{free}=1$ and $w_{free} = 3$. This effect on the upper bound has been illustrated as a separate graph in Fig. 7, which gives better approximation to the code performance in the error-floor region.

5 Summary and conclusions

A new and simple algorithm for the calculation of free-distance and low-weight distribution of the turbo code with optimised convolutional interleavers was presented. The algorithm computes the weight of the code with short interleaver lengths and then, based on the interleaver properties, the results are extrapolated to the desired length. For turbo codes with different interleavers, simulation results confirm the analysis of the code in the error-floor region.

The proposed weight distribution algorithm was utilised further to compare the performance of optimised convolutional interleavers with nonoptimised convolutional interleavers. It was confirmed that in the case of similar numbers of stuff bits for both interleavers, optimised interleavers outperform the nonoptimised interleavers [15].

Fig. 8 and 9 show the performance of different convolutional interleavers and their comparisons with pseudorandom interleavers for the four-state turbo code (1, 5/7) with the rate 1/3. For a short bitstream length $L = 169$, the pseudorandom interleaver requires 169 memory units, while better performance has been achieved applying convolutional interleavers with only 55 to 78 memory units.

However, for the bitstream length $L = 1024$, the performance is weaker than the pseudorandom interleaver in the error-floor region. This is due to the existence of a low free-distance value for the code. Although for the code with the longer bitstream lengths an interleaver with the higher period is applied, the distance between adjacent bits is not increased proportionally to the interleaver constituent parameters, that is period and space values, due to deletion of stuff bits in the end part of the interleaved data. In fact in this part, the distance between adjacent bits of the original bitstreams is almost constant for convolutional interleavers with different periods. Therefore for a code designed with different interleaver lengths, similar low-weight specifications are expected. For example, the four-state turbo code (1, 5/7) has a free distance value 10 with the multiplicity three, when constructed using convolutional interleavers ($T = 11$, $M = 1$, $L = 169$) and ($T = 20$, $M = 1$, $L = 1024$). Calculating the upper bound value from (3) for the code with the convolutional interleaver ($T = 20$, $M = 1$, $L = 1024$), gives higher *BER*

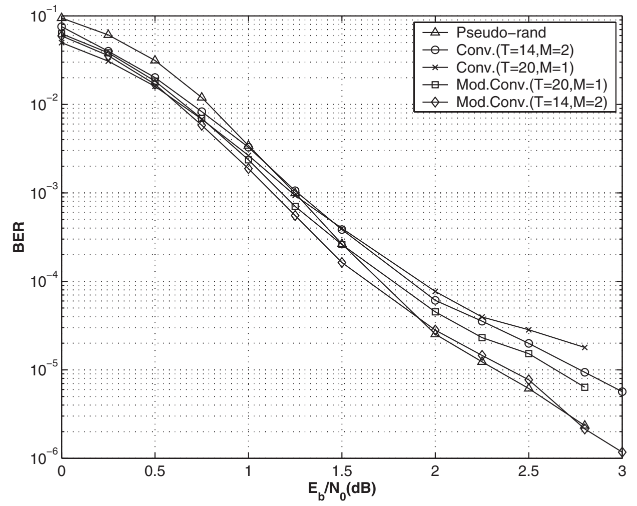

Fig. 9 *Performance of 4-state turbo codes (1, 5/7) with different interleavers and length $L = 1024$*

values compared with the code constructed with the pseudorandom interleaver. In simulation results illustrated in Fig. 9 it is clearly observed that for $BER > 10^{-4}$, convolutional interleavers ($T = 20$, $M = 1$) and ($T = 14$, $M = 2$) have better performance than the pseudorandom interleaver, while at $BER \leq 10^{-4}$, that is where the error floor is presented their performance is degraded [6].

Some modifications were accomplished improving the code performance in this region. In the proposed example the modified convolutional interleaver ($T = 14$, $M = 2$) performs similarly to the pseudorandom interleaver in the error floor region. With careful modification it is possible to construct convolutional interleavers having close or even better performance to most of the conventional block interleavers. In this comparison, convolutional interleavers were designed with the number of stuff bits, not exceeding 5% of the overall number of bits.

References

1 Perez, L.C., Seghers, J., and Costello, D.J.: 'A distance spectrum interpretation of turbo codes', *IEEE Trans. Inf. Theory*, 1996, **42**, (1), pp. 1698–1709

2 Pierleoni, P., Garello, R., and Benedetto, S.: 'Computing the free distance of turbo codes and serially concatenated codes with interleavers: algorithms and applications', *IEEE J. Sel. Areas Commun.*, 2001, **19**, pp. 800–812

3 Rosnes, E., and Ytrehus, Ø.: 'Improved algorithms for high rate turbo code weight distribution calculation'. Proc. 10th Int. Conf. on Telecommunications (ICT), March 2003, vol. 1, pp. 104–110

4 Yeh, P., Yilmaz, A., and Stark, W.: 'On the error analysis of turbo codes: Weight Spectrum Estimation (WSE) scheme'. Proc. IEEE Int. Symp. on Information Theory (ISIT), June 2003, p. 439

5 Huebner, D.J., and Costello, A.: 'A simple method of approximating the error floor of turbo codes with S-type permutors'. Proc. Int. Symp. on Information Theory (ISIT), 27 June–2 July 2004, p. 473

6 Vafi, S.: 'On the turbo codes design with convolutional interleavers', A dissertation of PhD thesis, University of Wollongong, 2005

7 Vafi, S., and Wysocki, T.: 'Iterative turbo decoder design with convolutional interleavers'. Proc. 4th int. Symp. on Communication Systems, Networks and Digital Signal Processing, Newcastle (CSNDSP), UK, 2004, pp. 124–127

8 Vafi, S., and Wysocki, T.: 'Computation of the free-distance and low-weight distribution of turbo codes with convolutional interleavers'. Proc. 15th IEEE Int. Symp. on Personal, Indoor and Mobile Radio Communications (PIMRC), September 2004, pp. 1356–1359

9 Forney, G.D.: 'Burst-correcting codes for the classic bursty channel', *IEEE Trans. Commun.*, 1971, **COM-19**, pp. 772–781

10 Hall, E.K., and Wilson, G.: 'Stream-oriented turbo codes', *IEEE Trans. Inf. Theory*, 2001, **47**, (5), pp. 1813–1831

11 Vafi, S., and Wysocki, T.: 'Performance of convolutional interleavers with different spacing parameters in turbo codes'. Proc. 6th Australian Workshop on Communications Theory, 2005, pp. 8–12

12 Dolinar, S., and Divsalar, D.: 'Weight distributions for turbo codes using random and nonrandom permutations'. TDA Progress Report 15 August 1995, pp. 56–65,

13 Takeshita, O., Collins, O.M., Massey, P.C., and Costello, D.J.: 'On the frame-error rate of concatenated turbo codes', *IEEE Trans. Commun.*, April 2001, **49**, (4), pp. 602–608

14 Hagenauer, J., Offer, E., and Papke, L.: 'Iterative decoding of binary block and convolutional codes', *IEEE Trans. Inf. Theory*, 1996, **42**, (2), pp. 429–445

15 Vafi, S., and Wysocki, T.: 'On the performance of turbo codes with convolutional interleavers'. Proc. Asia-Pasific Conf. on Communications (APCC), October 2005, pp. 222–226