# Puncturing effects on turbo codes

M.A. Kousa and A.H. Mugaibel

**Abstract:** The authors address various important issues related to punctured turbo codes. A modified technique for finding the transfer function of punctured turbo codes is developed. This modified technique provides a means of deterministic evaluation of the weight distribution of the code, as well as a possibility of studying various puncturing patterns. These advantages are shown with some illustrative examples. Moreover, the paper explains some characteristics of the puncturing pattern, and arrives at useful guidelines for the design of a good puncturing matrix.

## 1 Introduction

Turbo codes (TCs) demonstrate a means of closely approaching the Shannon capacity of a communication channel. The natural rate of a turbo code with two component codes is 1/3. Puncturing is commonly used to yield turbo codes of rates greater than the rate of the mother code. The need for puncturing arises in other applications as well. In speech or image compression some bits may be more significant than others, thus requiring a higher level of protection. Such unequal error protection can be achieved by puncturing [1]. Kim *et al.* [2] present an interesting scheme that uses punctured TCs to improve the performance of a soft handover in wideband CDMA (W-CDMA) systems. However, by far the most important application of puncturing is found in type-II adaptive hybrid FEC/ARQ schemes used over time varying channels [3–5].

Puncturing is used in these schemes to generate rate compatible punctured codes (RCPCs), a family of codes with different rates, from the same family of encoder/decoder pairs. Punctured convolutional codes were first introduced by Cain *et al.* [6]. Since then, a set of excellent papers on the analysis of punctured convolutional codes has appeared. The paper by Hagenauer [7] is undoubtedly the most cited paper in this field. As a result of all these efforts, punctured convolutional codes are now well understood. The situation is not the same for punctured turbo codes. Most of the results reported in the literature are based only on simulation. Some results even contradict each other [8].

This paper attempts to provide a better understanding of the effects of puncturing on the performance of turbo codes. Our approach is based on a modified technique for the evaluation of the weights of a punctured turbo code. The modified technique is shown to yield a more accurate enumeration of the weight distribution and lead to a tighter upper bound on the bit error rate of the code. Moreover, the modified technique facilitates the task of investigating different puncturing patterns.

© IEE, 2002

*IEE Proceedings* online no. 20020230

*DOI:* 10.1049/ip-com:20020230

Paper first received 11th July 2001 and in revised form 25th January 2002

M.A. Kousa is with the Electrical Engineering Department, King Fahd University of Petroleum & Minerals, P.O. Box 470, Dhahran 31261, Saudi Arabia

A.H. Mugaibel is with the Department of Electrical & Computer Engineering, Virginia Polytechnic Institute & State University, Blacksburg, Virginia 24061, USA

## 2 Elements of turbo codes

A simplified schematic of the turbo encoder is shown in Fig. 1. 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 a frame of input bits followed by the parity check bits from the first encoder, then the parity bits from the second encoder, i.e. the augmentation $X_2\ X_1\ X_0$.

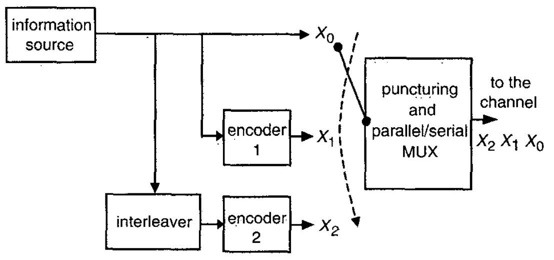


**Fig. 1** *Simplified turbo-encoder*

The convolutional code in every branch is called the constituent code (CC). The CCs can have similar or different generator functions. We will concentrate on the most widely used configuration where the two branches have the same CC.

Puncturing is usually introduced to increase the rate of the turbo code. Equivalently, one may use higher rate component codes. A comparison between these two approaches is interesting, but is beyond the scope of this work.

A puncturing matrix $\boldsymbol{P}$ of period $p$ applied to a turbo code having $N$ output branches can be represented by:

$$\boldsymbol{P} = \begin{bmatrix} g_{11} & \cdots & g_{1p} \\ \cdots & g_{ik} & \cdots \\ g_{N1} & \cdots & g_{Np} \end{bmatrix} \tag{1}$$

where every row corresponds to one output branch, i.e. the first row corresponds to the systematic branch, the second row corresponds to the first parity branch and so on. Note that $g_{ik} \in \{0,1\}$ where 0 implies that the corresponding bit is punctured. A degree of freedom in controlling the code rate can be gained by increasing $p$.

If $w(.)$ is the Hamming weight operator, then the rate of the code after puncturing with the puncturing matrix $\boldsymbol{P}$ is

$$R = \frac{p}{w(\boldsymbol{P})} \tag{2}$$

For the unpunctured case $w(\boldsymbol{P}) = Np$ and (2) reduces to $R = 1/N$.

## 3 Weight distribution of unpunctured turbo codes

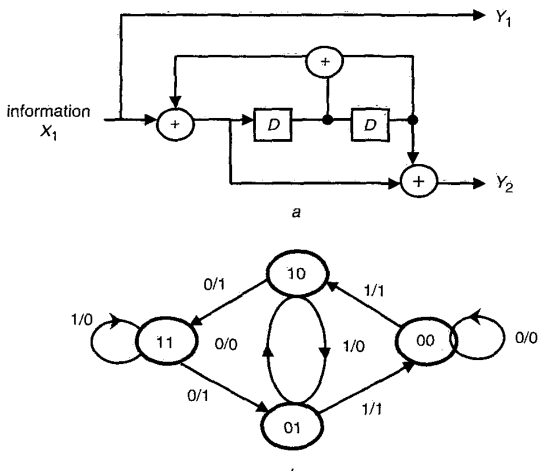
Evaluation of the performance of turbo codes requires a knowledge of the weight distribution, which can be obtained from the transfer function. We outline here the steps for calculating the transfer function of turbo codes. As a first step, we review the procedure for finding the transfer function of the constituent code.

### *3.1 Transfer function of the constituent code*

The constituent code, CC, is the basic building block of the turbo-encoder. Many researchers have used the (1,5/7 ,5/7) convolutional encoder for the CCs. It has an excellent performance compared to other encoders of the same complexity [9], because of its primitive feedback polynomial. The derivation that follows is based on this code, but the same technique is applicable to any other code.

The encoder of the CC can be represented in many ways. In Fig. 2, the block diagram, the state diagram and the state transition matrix of the selected code are shown. The state diagram is useful to enumerate all paths and their corresponding weights. In general, the transitions in the state diagram are labelled with the input/output weights of the corresponding branch. For a systematic code, the labels may be simplified to input/parity weights, as shown in Fig. 2*b*.

It is convenient to replace each branch label with the polynomial $W^w Z^z$, where $W$ and $Z$ are dummy variables



$$A(W,Z) = \begin{pmatrix} 1 & 0 & WZ & 0 \\ WZ & 0 & 1 & 0 \\ 0 & W & 0 & Z \\ 0 & Z & 0 & W \end{pmatrix}$$

c

**Fig. 2** *Different representations of the (1,5/7) encoder*
*a* Block diagram
*b* State diagram
*c* Transition matrix

introduced to facilitate the enumeration of the input weight and parity weight, respectively. For every branch, $w$ and $z$ are either 0 or 1. Using this notation, the information contained in the state diagram can be transformed to the transition matrix $A(W, Z)$ shown in Fig. 2*c*.

For a frame of length $k$ we define the frame transition matrix $\boldsymbol{F}(W, Z)$ as:

$$\boldsymbol{F}(W,Z) = A^k(W,Z) \tag{3}$$

The transfer function can be written in the form

$$T(W,Z) = \sum_{w,z} A_{w,z} W^w Z^z \tag{4}$$

where the coefficients $A_{w,z}$ denote the corresponding multiplicity of the codewords, i.e. the number of codewords having systematic weight $w$ and parity weight $z$. The transfer function $T(W, Z)$ may be found exclusively in the (0, 0) element of the matrix $\boldsymbol{F}(W, Z)$ [10].

For systematic codes, the overall output weight is given by the sum of the powers $w$+$z$. Therefore the weight distribution function of the code can be found from $T(W,Z)$ by setting $W = Z = H$, where $H$ is a dummy variable whose power is equal to the total weight of the codeword. The weight distribution function has the form

$$T(H) = \sum_h A_h H^h \tag{5}$$

where the coefficient $A_h$ is the number of codewords with Hamming weight $h$. Of prime importance in the evaluation of the weight distribution of turbo codes is the conditional transfer function of the constituent code, $T_w(Z)$. The function $T_w(Z)$ gives the weight distribution of the parity sequences that results from a weight-$w$ input sequence. It is represented by the summation

$$T_w(Z) = \sum_z A_{w,z} Z^z \tag{6}$$

Obviously, the transfer function and the conditioned transfer function are related by

$$T(W,Z) = \sum_w W^w T_w(Z) = \sum_w T_w(Z, W) \tag{7}$$

where

$$T_w(Z, W) = W^w T_w(Z) \tag{8}$$

is the weight distribution of the output for weight-$w$ input.

### *3.2 Transfer function of turbo codes*

Owing to the presence of the interleaver, the transfer function of turbo codes cannot be found directly. Some researchers [9, 10] resort to finding the transfer function of the turbo code from the conditional transfer function using (7). The conditional transfer function for the turbo code, $T_{w,TC}(Z)$, can be formulated from the corresponding functions of the CCs $T_{w,C_1}(Z)$ and $T_{w,C_2}(Z)$ for a particular interleaver, where the subscripts $TC$, $C_1$ and $C_2$ have been added to identify the three functions.

The role of a particular interleaver can only be incorporated by exhaustive enumeration of all possible input–output pairs, which is a lengthy process for large $k$. To overcome this difficulty, Benedetto and Montorsi [10] introduced the uniform interleaver, defined as follows: A uniform interleaver of length $k$ is a probabilistic device that maps a given input word of weight $w$ into all distinct $C$ permutations of it with equal probability of $1/C^k_w$.

The uniform interleaver cannot be used in practice, since one is required to use a particular random interleaver. It is not clear in this regard how a uniform interleaver behaves compared to a random interleaver. In [10] it was shown that for each value of the signal-to-noise ratio, the performance obtained with the uniform interleaver is achievable by at least one random interleaver. However, [11] demonstrated, by simulation, that the performance of some practical short interleavers exceeds that suggested by the uniform interleaver.

Under the assumption of a uniform interleaver, the conditional transfer function for the turbo code is given by [10]

$$T_{w,TC}(Z) = \frac{T_{w,C_1}(Z) \cdot T_{w,C_2}(Z)}{\binom{k}{w}} \tag{9}$$

Equation (9) is obtained based on the assumption that, for a given input weight, every output of the branch may be appended to every output of the second branch with the same probability. By substituting (9) in (7), we obtain the transfer function of the turbo code, and hence the coefficients $\{A_{w,z}\}$ for all possible values of $w$ and $z$.

The information obtained about the weight distribution is used to find the bit error probability for the turbo code [10]:

$$P_b \approx \frac{1}{2}\sum_m D_m \mathrm{erfc}\left(\sqrt{m\frac{R_c E_b}{N_0}}\right) \tag{10}$$

where $R_c$ is the code rate, $E_b/N_0$ is the signal to noise ratio (SNR) of the AWGN channel and $D_m$ is obtained from the weight distribution according to:

$$D_m = \sum_{z+w=m} \frac{w}{k} A_{w,z} \tag{11}$$

## 4 Weight distribution of punctured turbo codes

For unpunctured systematic codes, the overall output weight is equal to the sum of the input weight and the parity weight. The information about the input weight and the corresponding parity weight is fully contained in the transfer function, $T(W, Z)$. Unfortunately, this is not the case for punctured codes, which makes the task of finding their weight distribution a tedious one.

A brief description of the algorithm required for the evaluation of the transfer function of punctured turbo codes follows. We treat separately the case of puncturing parity bits and that of puncturing systematic (information) bits.

### 4.1 Puncturing parity bits

Let the frame size $k$ be a multiple of the puncturing period $p$. For illustration, let the puncturing period $p = 4$ and the puncturing pattern be 1101. This means that the third symbol in every sub-sequence of four symbols is punctured. To compute the frame transition matrix $\boldsymbol{F}(W, Z)$, we compute first the period transition matrix $\boldsymbol{B}$, i.e. the transition matrix over one period,

$$\boldsymbol{B}(W,Z) = \boldsymbol{A}(W,Z)\boldsymbol{A}(W,Z)\boldsymbol{A}(W,1)\boldsymbol{A}(W,Z) \tag{12}$$

Setting the parity variable $Z$ in the third matrix to '1' indicates that the third bit is not transmitted.

In general, for puncturing period $p$ and arbitrary puncturing vector $\boldsymbol{P}^i$ (where $\boldsymbol{P}^i$ is a vector corresponding to the $i$th parity row of the puncturing matrix $\boldsymbol{P}$), we form the period transition matrices for the two parity branches $\boldsymbol{B}_{C_1}$ and $\boldsymbol{B}_{C_2}$ using the matrix multiplication:

$$\boldsymbol{B}_{C_i}(W,Z) = \prod_{j=1}^{p} \boldsymbol{A}(W, Z^{\boldsymbol{P}^i(j)}) \quad i = 1,2 \tag{13}$$

Where $\boldsymbol{P}^i(j)$ is the $j$th element of the vector $\boldsymbol{P}^i$.

The frame transition matrix $\boldsymbol{F}(W, Z)$ can then be computed as the $(k/p)$th power of $\boldsymbol{B}(W, Z)$, i.e.

$$\boldsymbol{F}_{C_i}(W,Z) = \boldsymbol{B}_{C_i}^{k/p}(W,Z) \quad i = 1,2 \tag{14}$$

The above procedure can be used to calculate the weight distribution for different puncturing patterns of each constituent code. The resultant matrices are used, as outlined in Section 3, to find the conditional transfer functions, which are then substituted in (9) to find the composite transfer function of the turbo code under uniform interleaving.

### 4.2 Puncturing systematic bits

When systematic bits are punctured, the weight of the output systematic branch, say $y$, may be different from the weight of the input sequence, $w$. Unfortunately, the procedure to find the transfer function, explained in Section 3, does not have the capability to keep a record of both.

For an input sequence of length $k$ and Hamming weight $w$, if $M$ bits in a period of $p$ are punctured then the total number of punctured bits is $(kM/p)$. The punctured systematic sequence weight $y$ is then bounded as

$$w - (kM/p) \le y \le w \tag{15}$$

The probability $p(y|w)$ of producing a weight-$y$ systematic output by puncturing a weight-$w$ input sequence is often found in a probabilistic way. It is usually assumed that all patterns of $kM/p$ punctured bits out of the $k$ bits are possible, and that they are all equiprobable. Based on these assumptions, it follows that

$$p(y|w) = \frac{C_{w-y}^{kM/p} \times C_{y}^{k-(kM/p)}}{C_{w}^{k}} \tag{16}$$

In addition to being probabilistic, the above technique does not account for the particular puncturing pattern nor the particular CC used. In the following Section, we present a modified algorithm for finding the transfer function, which is free from these shortcomings.

## 5 Modified transfer function of punctured codes

As mentioned above, when puncturing is applied to the systematic sequence, the weight of the output systematic branch may be different from the weight of the input sequence. Hence, the overall output weight does not equal the sum of the input and parity check weights. This necessitates keeping a separate record for each.

To accomplish this task we will modify the state diagram by appending the variable $Y$ to each branch. The power of $Y$ can be 0 or 1 depending on whether the systematic bit is punctured or not. The resultant modified transfer function and the conditional transfer function are denoted by $T(W, Z, Y)$ and $T_w(Z, Y)$, respectively.

The modified conditional transfer function is represented by the following summation:

$$T_w(Z,Y) = \sum_{y,j} A_{y,j} Z^j Y^y \tag{17}$$

The systematic branch is considered to be a part of the first constituent code. Therefore only the transition matrix of the first constituent code has to be modified to account for the variable $Y$. The variable $Y$ is introduced in the transition matrix by replacing $W$ by $WY$.

To illustrate the above procedure, let us find the transfer function of the (1,5/7,5/7) code under puncturing. For the (1,5/7) constituent code the modified transition matrix will be

$$A(W,Z,Y) = \begin{pmatrix} 1 & 0 & WZY & 0 \\ WZY & 0 & 1 & 0 \\ 0 & WY & 0 & Z \\ 0 & Z & 0 & WY \end{pmatrix} \tag{18}$$

Consider the puncturing matrix,

$$\boldsymbol{P} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{19}$$

The transition matrix over one period for the first constituent code is

$$\boldsymbol{B}_{C_1}(W,Z,Y) = \boldsymbol{A}(W,Z,Y)\boldsymbol{A}(W,1,Y)\boldsymbol{A}(W,Z,1) \tag{20}$$

whereas that for the second constituent code, $C_2$, is

$$\boldsymbol{B}_{C_2}(W,Z) = \boldsymbol{A}(W,1)\boldsymbol{A}(W,Z)\boldsymbol{A}(W,Z) \tag{21}$$

The transition matrices for $C_1$ and $C_2$ over a frame of length $k$ are then calculated as

$$\boldsymbol{F}_{C_1}(W,Z,Y) = \boldsymbol{B}_{C_1}^{k/p}(W,Z,Y) \tag{22}$$

$$\boldsymbol{F}_{C_2}(W,Z) = \boldsymbol{B}_{C_2}^{k/p}(W,Z) \tag{23}$$

The resultant $\boldsymbol{F}_{C_1}$ and $\boldsymbol{F}_{C_2}$ are used to calculate the composite transfer function using the assumption of a uniform interleaver and the relation in (9)

In fact, the variables $Z$ and $Y$ in (22) and (23) can be replaced by the variable $H$. This simplification is based on the observation that from this point onward the contribution of the punctured systematic and parity branches to the overall output weight need not be distinguished. This will simplify the calculation of the frame transition matrix $\boldsymbol{F}_{C1}$, which will then be a function of two variables $W$ and $H$, with the power of the variable $H$ carrying the overall output weight for the first CC.

The above approach allows the exact determination of $y$ for a given $w$, and hence is referred to as deterministic. The next Section shows that the deterministic approach leads to a more accurate evaluation of the weight distribution of turbo codes that the probabilistic approach.

## 6 Deterministic approach versus probabilistic approach

Let us start by a simple example for demonstration. Consider the popular (7,4) Hamming code. All the codewords resulting from an input of weight two ($w = 2$) are shown in Table 1. The first four bits are the systematic bits $b_0\ b_1\ b_2\ b_3$. From the Table it is clearly seen that

$$T_2(Z) = 3Z + 3Z^2 \tag{24}$$

For a turbo code built of two (7, 4) Hamming codes and the uniform interleaver, the conditional transfer function is given by

$$T_{2,TC}(Z) = \frac{T_{2,C_1}(Z) \times T_{2,C_2}(Z)}{6} = 1.5Z^2 + 3Z^3 + 1.5Z^4 \tag{25}$$

Note that the sum of the coefficients of $Z$ is 6, which is the total number of codewords with input weight two.

If the two systematic bits $b_1$ and $b_3$ are punctured, then the output conditional transfer function for the punctured code can be calculated using either the probabilistic or the deterministic approaches. For the probabilistic approach, (16) yields:

$$p(y=0|w=2) = 1/6,\ p(y=1|w=2) = 2/3 \text{ and } p(y=2|w=2) = 1/6 \tag{26}$$

The output conditional transfer function for the punctured turbo code is then given by

$$T_{2,TC}(Z,Y) = (1.5Z^2 + 3Z^3 + 1.5Z^4)\left(\frac{1}{6} + \frac{2}{3}Y + \frac{1}{6}Y^2\right) \tag{27}$$

If the punctured bits were $b_0$ and $b_2$ instead of $b_1$ and $b_3$ the same result would be obtained. Moreover, the effect of puncturing, represented by the second bracket, is totally isolated from the first bracket, which is determined by the specific code used. This means that the amount of degradation in performance due to puncturing is independent of the specific code used. Both observations represent weaknesses of the probabilistic approach.

Let us now find the output conditional transfer function using the deterministic method developed in this paper. Table 2 traces the effect of puncturing bits $b_1$ and $b_3$ on the weight distribution of the code. After puncturing, the output conditional transfer function for the first constituent code is given by

$$T_{2,C_1}(Z,Y) = 2YZ + Z^2 + 2YZ^2 + Y^2Z \tag{28}$$

**Table 1: Codewords as a result of weight-two input for (7,4) Hamming encoder**

| Systematic | | | | Parity | | | Polynomial representation |
|---|---|---|---|---|---|---|---|
| $b_0$ | $b_1$ | $b_2$ | $b_3$ | $b_4$ | $b_5$ | $b_6$ | |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | $Z$ |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | $Z$ |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | $Z$ |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | $Z^2$ |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | $Z^2$ |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | $Z^2$ |
| Conditional distribution | | | | | | | $(3Z+3Z^2)$ |

**Table 2: Codewords as a result of input weight 2 for punctured (7,4) Hamming code when $b_1$ and $b_2$ are punctured**

| Systematic | | | | Parity | | | Polynomial |
|---|---|---|---|---|---|---|---|
| $b_0$ | $b_1$ | $b_2$ | $b_3$ | $b_4$ | $b_5$ | $b_6$ | representation |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | $Y^2Z$ |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | $YZ$ |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | $YZ$ |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | $Z^2$ |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | $YZ^2$ |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | $YZ^2$ |
| Conditional distribution | | $Y^2Z+2YZ+Z^2+2YZ^2$ | | | | | |

**Table 3: Conditional weight distribution of (7,4) Hamming turbo code using the probabilistic and deterministic techniques**

| Weight $j$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | Sum |
|---|---|---|---|---|---|---|---|---|
| Probabilistic | 0 | 0 | 0.25 | 1.5 | 2.5 | 1.5 | 0.25 | 6 |
| Deterministic | 0 | 0 | 0 | 1.5 | 3 | 1.5 | 0 | 6 |

and that for the second constituent code is given by

$$T_{2,C_2}(Z,Y) = 3Z + 3Z^2 \tag{29}$$

The conditional transfer function of the turbo code is given by

$$T_{2,TC} = \frac{T_{2,C_1} \times T_{2,C_2}}{6}$$
$$= 1.5H^3 + 3H^4 + 1.5H^5 \text{ (deterministic)} \tag{30}$$

whereas that obtained by the probabilistic approach is

$$T_{2,TC} = 0.25H^2 + 1.5H^3 + 2.5H^4 + 1.5H^5$$
$$+ 0.25H^6 \text{ (probabilistic)} \tag{31}$$

In obtaining (30) and (31) every $Y$ and $Z$ was replaced by $H$. The differences between the two methods are reported in Table 3. The deterministic method shows that the minimum output weight generated from a weight-2 input is not less than three, irrespective of the interleaver used, whereas the probabilistic method produces some codewords of weight 2.

It is worth noting that if $b_0$ and $b_2$ are punctured instead of $b_1$ and $b_3$ the deterministic approach yields a different weight distribution. This is illustrated in Table 4. Therefore, unlike the probabilistic method, the deterministic method is sensitive to the puncturing pattern.

Now let us turn to the (1,5/7,5/7) turbo code. The systematic sequence is punctured according to the vector [1 1 0]. Table 5 shows $D_m$, which is a measure of the weight distribution of the code defined in (11), up to the weight $m = 18$ using the two techniques. In particular, the results of the deterministic approach show that the minimum distance of the code cannot be less than 6, whereas the probabilistic approach allows the presence of codewords of weights 4 and 5. Also, the results of the deterministic approach show that odd-weight codewords do not exist, whereas the probabilistic approach fails to detect this structural information.

Fig. 3 shows a comparison between the bounds obtained using the probabilistic method and the deterministic method for the (1,5/7,5/7) turbo code. We considered two puncturing patterns of the systematic sequence. In the first case, one systematic bit is punctured in a period of two ($M = 1$, $p = 2$), whereas in the second case, one systematic bit is punctured in a period of three ($M = 1$, $p = 3$). The Figure shows that, although both bounds are comparable at low SNR (waterfall region), the deterministic bound is getting significantly tighter at high SNR (flattening region). This observation is consistent for both puncturing patterns. The reason behind this behaviour can be explained by noting that, at high values of SNR, the code performance is dominated by the minimum distance of the code, which is underestimated by the probabilistic approach.

**Table 4: Codewords as a result of input weight 2 for punctured (7,4) Hamming code when $b_0$ and $b_2$ are punctured**

| Systematic | | | | Parity | | | Polynomial |
|---|---|---|---|---|---|---|---|
| $b_0$ | $b_1$ | $b_2$ | $b_3$ | $b_4$ | $b_5$ | $b_6$ | representation |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | $Z$ |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | $YZ$ |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | $Z^2$ |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | $Y^2Z^2$ |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | $YZ^2$ |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | $YZ^2$ |
| Conditional distribution | | | $Z+YZ+Z^2+Y^2Z^2+2YZ^2$ | | | | |

**Table 5: $D_m$ for the (1,5/7,5/7) turbo code and $P = [1\,1\,0]$ using the probabilistic and the deterministic techniques**

| Codeword weight $m$ | $D_m$ [probabilistic] | $D_m$ [deterministic] |
|---|---|---|
| 3 | 0 | 0 |
| 4 | 0.00001758369 | 0 |
| 5 | 0.00010925460 | 0 |
| 6 | 0.00032662980 | 0.0004947929 |
| 7 | 0.00080264170 | 0 |
| 8 | 0.00390303700 | 0.0097513440 |
| 9 | 0.01206679000 | 0 |
| 10 | 0.01856358000 | 0.0366604400 |
| 11 | 0.02618730000 | 0 |
| 12 | 0.03786333000 | 0.0728424400 |
| 13 | 0.04912956000 | 0 |
| 14 | 0.07454436000 | 0.1423494000 |
| 15 | 0.10620620000 | 0 |
| 16 | 0.17585560000 | 0.3456495000 |
| 17 | 0.28859980000 | 0 |
| 18 | 0.51055000000 | 1.0028360000 |

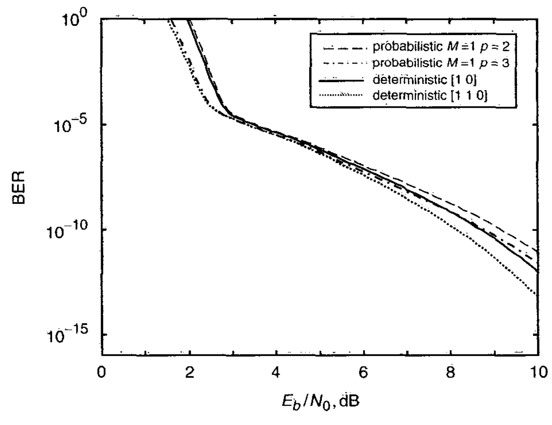


**Fig. 3** *Comparison of the upper bounds on the BER using the probabilistic and the deterministic techniques*

## 7 Puncturing patterns

In principle, puncturing can be applied to information bits and/or parity bits. However, it is reported that puncturing systematic bits may have a drastic effect on the code performance [12]. Therefore, it is advisable to avoid puncturing systematic bits and confine puncturing to the parity bits.

The fact that the two parity sequences play an identical role at the decoder suggests that puncturing should be distributed equally between the parity branches. It is also intuitive to suggest that the punctured bits in any sequence be well scattered, as adjacent puncturing destroys the structure of the code.

To test the above conjectures, four puncturing matrices are investigated:

$$\begin{aligned} \boldsymbol{P}_1 &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{P}_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ \boldsymbol{P}_3 &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \boldsymbol{P}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{aligned} \tag{32}$$

All four matrices have a period of four ($p = 4$) and a total number of punctured bits equal to 4, thus ensuring a fair comparison between them with respect to the resultant code ratio. In $\boldsymbol{P}_1$, puncturing is equally distributed between the two parity branches and maximally scattered within each branch. In $\boldsymbol{P}_2$, puncturing is not equally distributed, whereas in $\boldsymbol{P}_3$ it is not well scattered. The puncturing matrix $\boldsymbol{P}_4$ is the extreme case of puncturing the second parity sequence completely and leaving the first parity sequence intact. In fact, this puncturing pattern transforms a turbo code to a normal convolutional code.

Fig. 4 shows the performance of the turbo code under the puncturing matrices. Clearly, $\boldsymbol{P}_1$ has the best performance, $\boldsymbol{P}_2$ and $\boldsymbol{P}_3$ are slightly inferior to $\boldsymbol{P}_1$ but $\boldsymbol{P}_4$ is much worse. These results support the previous conjectures.

The procedure for formulating the block transition matrix requires computing the transition matrix over one period. The transition matrix, $\boldsymbol{B}$, over one period is computed as in (13). Multiplying $B$ by itself $(k/p)$ times results in the required transition matrix over the entire frame. Owing to the grouping property of matrix multiplication, cyclic shifts will not affect the overall result, assuming the sequence is relatively long. Noting that cyclic shifting of columns does not change the puncturing pattern, with respect to the two conditions stated in the previous Section, one can also arrive at the same conclusion. The independence of the performance to cyclically shifting the columns of the puncturing matrix is verified by simulation, as shown in Fig. 5. $\boldsymbol{P}_5$ is obtained from $\boldsymbol{P}_1$ by one cyclic shift of columns.

$$\boldsymbol{P}_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{P}_5 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \tag{33}$$

Fig. 5 shows that $\boldsymbol{P}_1$ and $\boldsymbol{P}_5$ have essentially the same performance. This property has been tested for different interleavers and the behaviour was found to be consistent.

It should be noted that permutation of columns other than cyclic shifting might affect the performance, as it might modify the scattering of punctured bits. In Fig. 3*a*, $\boldsymbol{P}_3$ is obtained from $\boldsymbol{P}_1$ by interchanging columns 1 and 4. Fig. 4 shows that the two patterns yield a different performance. Although the difference is not great, it will be more pronounced as the puncturing period increases.

The results of this Section lead to the following set of guidelines for constructing a good puncturing matrix: puncturing systematic bits should be avoided; puncturing should be applied equally to parity sequences; within every parity sequence, punctured bits should be scattered as much as possible and puncturing matrices obtained by cyclically shifting the columns have essentially the same performance.

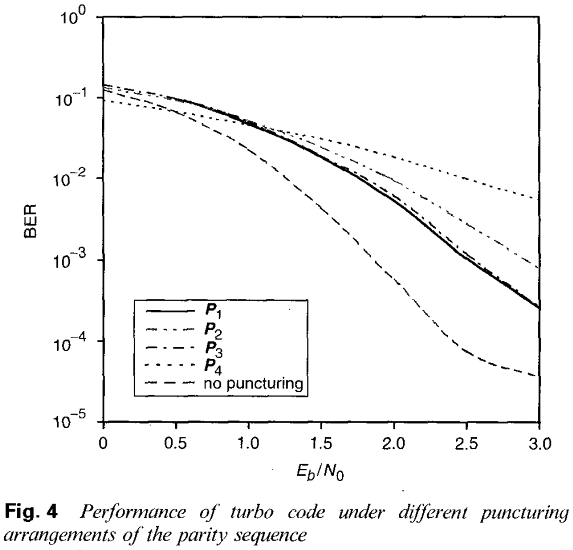


**Fig. 4** *Performance of turbo code under different puncturing arrangements of the parity sequence*
See (32) for definitions of puncturing matrices

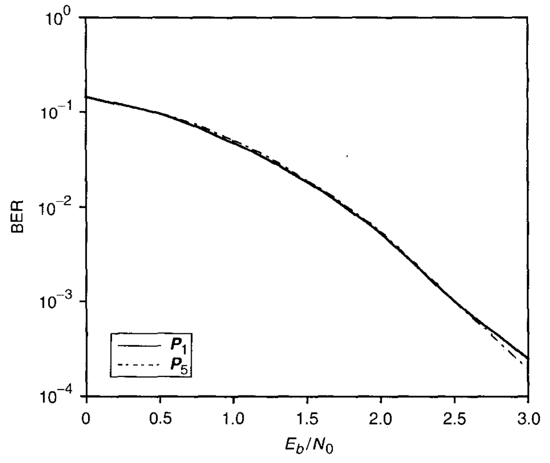


**Fig. 5** *Performance under column permutations on the puncturing matrix*
See (33) for definitions of puncturing matrices

## 8 Conclusion

A modified technique for the evaluation of the transfer function of punctured turbo codes has been developed. The modified technique provides a more accurate enumeration of the weights and, consequently, a tighter upper bound on the bit-error rate. It also permits the study and comparison of different puncturing patterns. Our investigation leads to a set of useful hints in the design of a good puncturing matrix.

We believe that the techniques and findings of this paper are useful for any further research on punctured turbo codes.

## 9 Acknowledgment

The authors would like to thank KFUPM for supporting this research.

## 10 References

1 CAIRE, G., and BIGLIERI, E.: 'Parallel concatenated codes with unequal error protection', *IEEE Trans. Commun.*, 1998, **46**, (5), pp. 565–567

2 KIM, B., and KWON, S.: 'A new soft handover scheme using punctured turbo codes in the wideband CDMA system'. Proceedings of the IEEE Conference on VTC, 2001, pp. 1420–1424

3 CHAN, W., GERANIOTIS, E., and NGUYEN, V.: 'An adaptive hybrid FECI ARQ protocol using turbo codes'. Proceedings of the 6th ICUPC, 1997, Vol. 2, pp. 541–545

4 LAW, C.F., LAU, C., and KO, T.M.: 'A modified adaptive FEC/ARQ protocol using turbo codes with incremental redundancy transmission'. Proceedings of the IEEE conference on VTC, 1999, pp. 1670–1674

5 ABOU-EL-AZM, A., EL-FISHAWY, N., and MOHAMMED, F.: 'Improving the transmission efficiency in the mobile communication systems using turbo codes'. Proceedings of the IEEE conference on VTC, 2000, pp. 1836–1843

6 CAIN, J.B., CLARK, G.C., and GEIST, J.M.: 'Punctured convolutional codes of rate (n−1)/n and simplified maximum likelihood decoding', *IEEE Trans. Inf. Theory*, 1979, **IT-20**, pp. 79–100

7 HAGENAUER, J.: 'Rate compatible punctured convolutional codes (RCPC-codes) and their application', *IEEE Trans. Commun.*, 1988, **36**, pp. 389–400

8 MO, F., KWATRA, S., and KIM, J.: 'Analysis of puncturing patterns for high rate turbo codes'. Proceedings of the Military Communications Conference, 1999, pp. 547–550

9 BENEDETTO, S., GARELLO, R., and MONTORSI, G.: 'A search for good convolutional codes to be used in the construction of turbo codes', *IEEE Trans. Commun.*, 1998, **46**, (9), pp. 1101–1105

10 BENEDETTO, S., and MONTORSI, G.: 'Unveiling turbo-codes: some results on parallel concatenated coding schemes', *IEEE Trans. Inf. Theory*, 1996, **42**, (2), pp. 409–428

11 BURR, A., and WHITE, G.: 'Comparison of iterative decoder performance with union bounds for short frame turbo codes', *Ann. Telecommun.*, 1999, **54**, (3–4), pp. 201–207

12 BERROU, C., and GLAVIEUX, A.: 'Near optimum-error correcting coding and decoding: turbo-codes', *IEEE Trans. Commun.*, 1996, **44**, pp. 1261–1271