A Novel Technique To Evaluate the Transfer Function of Punctured Turbo Codes

Ioannis Chatzigeorgiou, Miguel R. D. Rodrigues, Ian J. Wassell
Digital Technology Group, Computer Laboratory
University of Cambridge, United Kingdom
Email: {ic231, mrdr3, ijw24}@cam.ac.uk

Rolando Carrasco
Department of EE&C Engineering
University of Newcastle, United Kingdom
Email: r.carrasco@ncl.ac.uk

***Abstract*— A novel approach for the calculation of the transfer function of a terminated punctured convolutional code, which can be seen as a convolutional block code, is presented in this paper. The transfer function of the convolutional block code can be then used to evaluate the transfer function of a punctured turbo code and derive a tight upper bound on the bit error probability. Furthermore, the approach is used to find puncturing patterns that generate punctured turbo codes with optimal performance.**

I. Introduction

Turbo codes, originally conceived by Berrou *et al*. [1] are widely known for their astonishing performance on the additive white Gaussian noise (AWGN) channel. Methods to evaluate an upper bound on the bit error probability (BEP) of a parallel-concatenated coding scheme have been proposed by Divsalar *et al*. [2] as well as Benedetto and Montorsi [3]. In addition, guidelines for the optimal design of the constituent convolutional codes were presented in [4].

The rate of a turbo code can be increased by puncturing the outputs of the turbo encoder. Guidelines and design considerations for punctured turbo codes have been derived by analytical [5], [6], [7] as well as simulation-based approaches [8], [9], [10]. The upper bounds on the BEP are evaluated based on [3], while Kousa and Mugaibel [7] propose a combination of [2] and [3].

The motivation for this paper is to propose an alternative method for the evaluation of the transfer function of a constituent convolutional block code in a turbo encoder, which is essential for the calculation of an upper bound on the BEP. More specifically, the transfer function of a convolutional block code is computed using an augmented version of the modified state diagram of the original convolutional code. The augmented state diagram can then be further manipulated so as to yield the transfer function of a punctured convolutional block code.

II. Transfer Function of Convolutional Codes

A detailed description of the properties of convolutional codes as well as expressions required to evaluate the performance of the maximum likelihood (ML) decoder, are presented in [11]. In particular, it is shown that an upper bound on the BEP of the ML decoder for binary phase shift keyed (BPSK) modulation on an AWGN channel can be obtained if the transfer function of the convolutional code is known.

A generic form for the transfer function $T(W, D, L)$ of a convolutional code is:

$$T(W, D, L) = \sum_{w,d,l} T_{w,d,l} W^w D^d L^l, \tag{1}$$

where $T_{w,d,l}$ denotes the number of paths that start from the zero state and remerge with the zero state after l steps, i.e., their length is l, and are associated with an input sequence of weight w, and an output sequence of weight d.

An example of a recursive convolutional encoder with code rate 1/2, recursive generator polynomial 3 and forward generator polynomial 2, i.e., RSC(1,2/3), is shown in Fig.1(a). The state diagram of the code is illustrated in Fig.1(b). A branch corresponds to the transition from a state to another state. Each branch is labelled by the input word that caused the transition and the output word generated by the transition.

The modified state diagram of the convolutional code, presented in Fig.1(c), can be used to obtain its transfer function. In order to avoid circulation around the self-loop, we split the state X_0 into two separate states, the start state X_S and the end state X_E. Furthermore, we label the branches according to the Hamming weight of the input and output words. The exponent of W corresponds to the weight of the input word whereas the exponent of D corresponds to the weight of the output word. Each branch also has term L, which represents a time-step. We express each state of the diagram as a function of the other states, so as to obtain the state equations. Upon solving these equations for the ratio X_E/X_S,

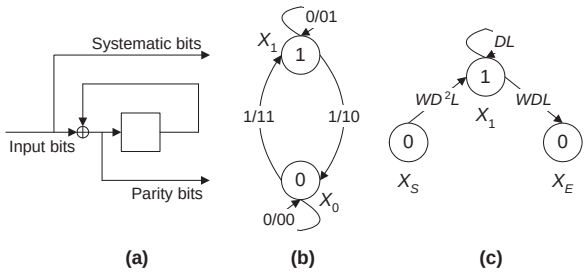

Fig. 1. Block diagram (a), state diagram (b) and modified state diagram (c) of RSC(1,2/3)

This work is supported by EPSRC under Grant GR/S46437/01.

we obtain the transfer function for RSC(1,2/3):

$$T(W,D,L) = W^2D^3L^2 + W^2D^4L^3 + W^2D^5L^4 + W^2D^6L^5 + \ldots, \quad (2)$$

which tells us that there is one path of length 2 generated by an input sequence of weight 2 which produces an output sequence of weight 3 (i.e. $W^2D^3L^2$), and so on.

III. Transfer Function of Convolutional Block Codes

A. Weight Enumerating Functions

A parallel concatenated convolutional code (PCCC) is formed by two convolutional encoders, each of memory ν, and an interleaver of length $N-\nu$ [1]. The input to the PCCC is a block of N information bits, including the ν bits used for trellis termination. We refer to each constituent terminated convolutional code, as a *convolutional block code*. It was shown in [2] and [3] that the performance of a PCCC depends upon the transfer functions, or equivalently on the input-output weight enumerating functions (IOWEFs), of the constituent convolutional block codes. The IOWEF of a convolutional block code assumes the form:

$$B(W,D) = \sum_{w,d} B_{w,d} W^w D^d, \quad (3)$$

where $B_{w,d}$ denotes the number of codeword sequences with weight d generated by an input sequence of weight w.

In the case of a rate-k_0/n_0 convolutional code, k_0 input bits cause a transition from a state to another state, at a time-step. A block of N input bits will cause N/k_0 transitions after N/k_0 time-steps, resulting in a path of length N/k_0. When $k_0 = 1$, the values for the total number of time-steps, the path length and the block size coincide. The trellis of the corresponding convolutional block code is the truncation of the trellis of the rate-k_0/n_0 original convolutional code at step N/k_0. More specifically, the transfer function $T(W,D,L)$ of a convolutional code provides all paths that start from the zero state, diverge from the all-zero path at step 1 and at some point remerge with the zero state and remain at it. The IOWEF $B(W,D)$ of the convolutional block code provides all paths of length N that start from the zero state, can remerge with and diverge from the zero state more than once and terminate at the zero state.

Although the IOWEF $B(W,D)$ of a convolutional block code relates the weight of an input sequence with the weight of the generated codeword sequences composed of the n_0 output sequences of the encoder, it does not provide any information about the weight of each individual output sequence. We define the generalized input-output weight enumerating function (G-IOWEF) of a rate-k_0/n_0 convolutional block code, as follows:

$$B(W,Z_1,\ldots,Z_{n_0}) = \sum_{w,z_1,\ldots,z_{n_0}} B_{w,z_1,\ldots,z_{n_0}} W^w Z_1^{z_1} \ldots Z_{n_0}^{z_{n_0}}, \quad (4)$$

where $B_{w,z_1,\ldots,z_{n_0}}$ denotes the number of codeword sequences with weight $z_1 + \ldots + z_{n_0}$ generated by an input sequence of weight w. The value of z_i, $i = 1,\ldots,n_0$, corresponds to the weight of the i-th output sequence of the encoder. The IOWEF can be derived from the G-IOWEF by setting:

$$B_{w,d} = \sum_{z_1+\ldots+z_{n_0}=d} B_{w,z_1,\ldots,z_{n_0}} \quad (5)$$

and substituting $B_{w,d}$ in (3), which is equivalent to setting $Z_1 = \ldots = Z_{n_0} = D$ in (4).

The G-IOWEF of a rate-1/2 RSC block code assumes the form:

$$B(W,U,Z) = \sum_{w,u,z} B_{w,u,z} W^w U^u Z^z, \quad (6)$$

where $B_{w,u,z}$ denotes the number of codeword sequences with weight $u+z$ generated by an input sequence of weight w. The value of u corresponds to the weight of the output systematic sequence, whereas z corresponds to the weight of the output parity check sequence of the encoder. Similarly, the IOWEF of the convolutional block code can be derived from the G-IOWEF by setting:

$$B_{w,d} = \sum_{u+z=d} B_{w,u,z} \quad (7)$$

and substituting $B_{w,d}$ in (3).

The G-IOWEF can describe both unpunctured and punctured convolutional block codes. However, in the case of unpunctured convolutional block codes, the additional information provided by the G-IOWEF is redundant, thus the IOWEF is preferred.

B. Proposed Technique

In this subsection we suggest a modification of the state diagram of a convolutional code, from which the IOWEF or the G-IOWEF of the corresponding convolutional block code can be directly derived. We focus on the derivation of the G-IOWEF because it is straightforward to extend our technique to the case of punctured convolutional codes.

For consistency, we continue to pursue the example of Fig.1. By adding a new state X_0 as well as the circulation loop, the state diagram of the convolutional block code shown in Fig.2 can be obtained from the modified state diagram presented previously in Fig.1(c). These additions enable an indefinite number of remergings with the zero state for an indefinite

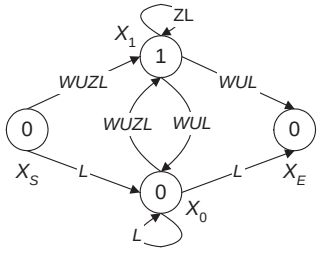

Fig. 2. Augmented state diagram of the convolutional block code RSC(1,2/3).

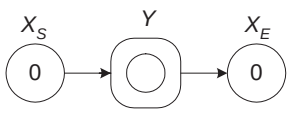

Fig. 3. Generic diagram of a convolutional block code.

period of time. The new "augmented" diagram in Fig.2 also has a start state X_S from which branches only emerge, and an end state X_E where branches only terminate. Furthermore, since we are interested in the G-IOWEF of the convolutional block code, the format of the label of each branch changed from $W^w D^d L$ to $W^w U^u Z^z L$, where $u = w$ and $z = d - u$, based on the definition of the G-IOWEF in (6) and its relation with the IOWEF in (7).

A generic representation of the augmented state diagram of a convolutional block code is shown in Fig.3, where the box labeled Y represents the set of all possible states of the convolutional encoder. The link that connects the start state X_S with Y in the generic diagram, represents the set of all branches of the augmented diagram that connect X_S with different states of the convolutional encoder. The same logic applies to the link that connects Y with the end state X_E. The circle within Y denotes that the states which belong to the set are interconnected.

As usual, we derive the state equations and we solve them for the ratio X_E/X_S. The result will be a sum of two terms:

$$\frac{X_E}{X_S} = f(L) + f(W, U, Z, L). \tag{8}$$

The first term $f(L)$ is a function of L only and represents the set of all paths with zero input weight and zero codeword weight, i.e., all the all-zero paths for different block lengths. These paths start from state X_S, stay at state X_0 for an indefinite number of steps by circulating around the self-loop and finally terminate at state X_E. The all-zero paths are not of interest in the computation of $B(W, U, Z)$ and are ignored.

The second term $f(W, U, Z, L)$ represents the set of all paths that start from the zero state and end at the zero state for various block lengths. It can be expressed as:

$$f(W, U, Z, L) = \sum_{w,u,z,l} B_{w,u,z,l} W^w U^u Z^z L^l, \tag{9}$$

where $B_{w,u,z,l}$ is a nonnegative integer.

If we are interested in a specific block size N, we need to consider only the terms L^l of $f(W, U, Z, L)$ with $l = N$. We can rewrite (9) as follows:

$$\begin{aligned} f(W, U, Z, L) = & \sum_{\substack{w,u,z,l \\ l \neq N}} B_{w,u,z,l} W^w U^u Z^z L^l + \\ & + \sum_{\substack{w,u,z,l \\ l = N}} B_{w,u,z,l} W^w U^u Z^z L^l \\ = & \; f_{l \neq N}(W, U, Z, L) + f_{l=N}(W, U, Z, L). \end{aligned} \tag{10}$$

For a given block size N, the G-IOWEF $B(W, U, Z)$ of the convolutional block code can be derived from $f_{l=N}(W, U, Z, L)$ by setting $L = 1$, i.e.,

$$B(W, U, Z) = f_{l=N}(W, U, Z, L)\Big|_{L=1}. \tag{11}$$

Different techniques for the evaluation of the IOWEF of a convolutional block code were also presented by Divsalar *et al.* [2] as well as Benedetto and Montorsi [3]. Divsalar *et al.* [2] suggested the derivation of the state transition matrix of the convolutional code and the computation of an inverse matrix, which is a function of the state transition matrix. Benedetto and Montorsi [3] proposed a two-stage technique. In the first stage, an intermediate transfer function of the convolutional code is evaluated, yielding terms having particular input and output weights. In the second stage, the IOWEF of the convolutional block code is computed, since each term of the IOWEF associated with a specific input and output weight, can be expressed as a function of the relevant terms of the intermediate transfer function, i.e., those associated with identical input and identical output weights.

The technique we propose [12] can be seen as a refinement of Benedetto's and Montorsi's approach. Owing to the introduction of the augmented state diagram, the IOWEF or the G-IOWEF of the convolutional block code can be directly computed without the need of an intermediate transfer function. More importantly, our approach can be easily extended to punctured convolutional codes.

IV. Punctured Convolutional Codes

A rate-k/n punctured convolutional code is obtained by periodic elimination of specific codeword bits from the output of a rate-k_0/n_0 convolutional encoder. A puncturing pattern $\mathbf{P}$ can be represented by a $n_0 \times M$ matrix as follows:

$$\mathbf{P} = \begin{bmatrix} p_{11} & \cdots & p_{1M} \\ \vdots & \ddots & \vdots \\ p_{n_0,1} & \cdots & p_{n_0,M} \end{bmatrix}, \tag{12}$$

where M is the puncturing period and $p_{ij} \in \{0, 1\}$, with $i = 1, \ldots, n_0$ and $j = 1, \ldots, M$. For $p_{ij} = 0$ the corresponding output bit is punctured.

For the case of a punctured RSC code with two output streams, i.e., $n_0 = 2$, the puncturing pattern $\mathbf{P}$ of the code consists of the puncturing pattern $\mathbf{P}_U$ of the systematic stream and the puncturing pattern $\mathbf{P}_Z$ of the parity stream:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_U \\ \mathbf{P}_Z \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1M} \\ p_{21} & p_{22} & \cdots & p_{2M} \end{bmatrix}. \tag{13}$$

The puncturing patterns of both the systematic and parity streams are row vectors. The puncturing pattern $\mathbf{P}$ can also be seen as a matrix that consists of column vectors, namely column puncturing vectors (CPVs) $\mathbf{P}_j$, with $j = 1, \ldots, M$, that determine which bits will be punctured at each time-step. Accordingly, the puncturing pattern $\mathbf{P}$ can be expressed as:

$$\mathbf{P} = [\mathbf{P}_1 \quad \mathbf{P}_2 \quad \ldots \quad \mathbf{P}_M] = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1M} \\ p_{21} & p_{22} & \cdots & p_{2M} \end{bmatrix}. \tag{14}$$

V. Transfer Function of Punctured Convolutional Block Codes

In this section we extend the concept of the augmented state diagram to include the case of punctured convolutional codes. For simplicity, we continue to consider the rate-1/2 RSC(1,2/3) code. In order to construct the augmented state diagram of the punctured RSC(1,2/3) block code, we need to introduce the CPV into the labeling procedure of each branch. We see in Fig.2 that a transition from state X_i to state X_k,

$\{i,k\} \in \{0,1\}$, is labeled $W^w U^u Z^z L$, where w is the weight of the input word that caused a transition, which generated a systematic output with weight u and a parity output with weight z. Let us concentrate on the same transition from X_i to X_k when puncturing occurs. If $\mathbf{P}_j$, $j = 1, \ldots, M$, is the active CPV at a specific time-step, the label of the branch will change to $W^{w'} U^{u'} Z^{z'} L$, where w', u' and z' are related to w, u, z as well as the elements of $\mathbf{P}_j$, i.e., p_{1j} and p_{2j}, as follows:

$$w' = w, \quad u' = u \cdot p_{1j}, \quad z' = z \cdot p_{2j}. \tag{15}$$

The values of u' and z' depend on the active CPV, therefore the label of the branch that connects X_i with X_k cannot be a constant term, since it can assume up to M different values due to the M available CPVs.

To overcome this problem, we introduce M sets of states. Each set Y_j contains all possible states of the convolutional encoder. When $\mathbf{P}_j$ is the active CPV, a transition to state X_i in set Y_j occurs. At the next time-step, $\mathbf{P}_{j+1}$ becomes the active CPV and a transition from state X_i in set Y_j to state X_k in set Y_{j+1} takes place. As time progresses, the CPVs are repeated periodically, i.e. $\mathbf{P}_1$,$\mathbf{P}_2$,...,$\mathbf{P}_M$,$\mathbf{P}_1$,..., resulting in transitions to states that belong to sets Y_1,Y_2,...,Y_M,Y_1,..., respectively. Therefore, the problem of having M different terms associated with a branch that connects state X_i with state X_k, is overcome by having M branches, each one of which pairs state X_i of a set with state X_k of a different set.

So as to better understand the concept of the augmented state diagram of a punctured convolutional block, we give an example for a puncturing period of $M = 2$. In order to increase the code rate of RSC(1,2/3) from 1/2 to 2/3, we use the following puncturing pattern:

$$\mathbf{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \text{ with } \mathbf{P}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \text{ and } \mathbf{P}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{16}$$

where $\mathbf{P}_1$ and $\mathbf{P}_2$ are CPVs. Since 1 in 4 codeword bits is punctured, 3 codeword bits are transmitted for every $M = 2$ information bits, therefore the code rate of the punctured convolutional code is 2/3.

The augmented state diagram of the rate-2/3 RSC(1,2/3) block code is presented in Fig.4. $\mathbf{P}_1$ is the active CPV during the transitions, represented by solid lines in Fig.4, that originate from states in set Y_2 and terminate at states in set Y_1. Since both elements of $\mathbf{P}_1$ are equal to 1, the outputs of the encoder are not punctured, therefore the labels of the branches that connect the states of set Y_2 with the states of set Y_1 are identical to the labels of the corresponding branches of the augmented diagram in Fig.2. $\mathbf{P}_2$ is the active CPV during the transitions, represented by dashed lines in Fig.4, that originate from states in set Y_1 and terminate at states in set Y_2. In this case, the parity output of the encoder is punctured, therefore term Z does not appear in any of the branch labels. To complete the augmented diagram, states X_S and X_E have been included. Since the encoder starts from state X_S, $\mathbf{P}_1$ is the active CPV during the transition from X_S to a state in set Y_1 at the first time-step. At the last time-step, the encoder returns to the zero state, i.e., a transition to state X_E occurs. In order to terminate the code, those states of each set which are connected to state X_0 of a different set, must also be connected to state X_E.

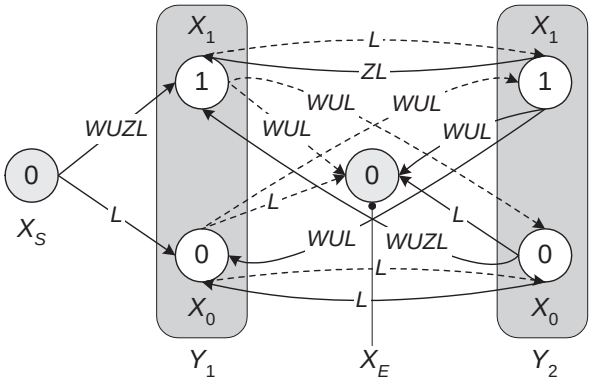

Fig. 4. Augmented state diagram of the rate-2/3 RSC(1,2/3) block code.

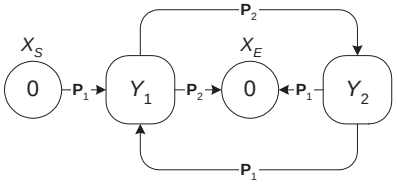

Fig. 5. Generic diagram of a punctured convolutional block code ($M = 2$).

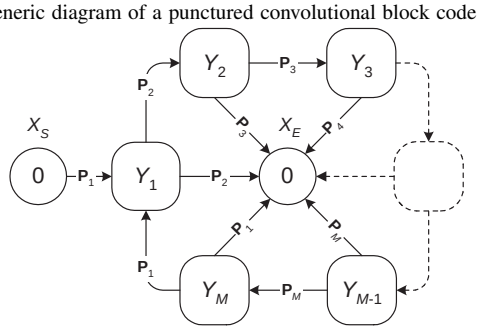

Fig. 6. Generic diagram of a punctured convolutional block code.

A generic representation of the augmented state diagram of a convolutional block code, which is punctured using a pattern of period $M = 2$, is shown in Fig.5. Both sets Y_1 and Y_2 contain all possible states of the convolutional encoder. A link between two sets represents the collection of branches of the augmented diagram that connect the states of the two different sets. Each link is labeled with the associated CPV. The same logic applies to the link that connects the start state X_S with Y_1 as well as the links that connect all sets to the end state X_E.

The general case of a convolutional block code, which is punctured using a pattern of period M, is shown in Fig.6. As previously, we derive the state equations and we solve them for the ratio X_E/X_S. The result for the case of a punctured convolutional block code, of rate k_0/n_0 before puncturing, is a sum of two terms, $f(L)$ and $f(W, Z_1, \ldots, Z_{n_0}, L)$. For simplicity, as in (8), we consider an RSC code of rate

1/2 before puncturing, therefore the second term becomes $f(W,U,Z,L)$. The first term $f(L)$ is a function of L and can be ignored. The second term $f(W,U,Z,L)$, which represents the set of all paths that start from the zero state and end at the zero state for various block lengths, is also a sum:

$$f(W,U,Z,L)=\sum_{j=1}^{M} f^{\mathbf{P}_j}(W,U,Z,L). \tag{17}$$

The polynomial $f^{\mathbf{P}_j}(W,U,Z,L)$ represents all paths of length $l=j,M+j,2M+j,\ldots$, which terminate at state X_E when $\mathbf{P}_j$ is the active CPV.

In order to compute the G-IOWEF of the convolutional block code for a specific block size $N>1$, we need to consider only the terms L^l of $f(W,U,Z,L)$ with $l=N$. Taking into account (10) and (17), we conclude that the G-IOWEF $B(W,U,Z)$ assumes the form:

$$B(W,U,Z) = f_{l=N}^{\mathbf{P}_j}(W,U,Z,L)\Big|_{L=1} \tag{18}$$

$$\text{for } j = N-kM, \tag{19}$$

where $k=0,1,2,\ldots$, $j\in\{1,2,\ldots,M\}$ and $N>1$.

In practice, we do not need to calculate all polynomials $f^{\mathbf{P}_j}(W,U,Z,L)$, for $j\in\{1,2,\ldots,M\}$, so as to derive the G-IOWEF of the convolutional block code for a block size of N. We can remove the links that connect X_E with all sets except for the link associated with $\mathbf{P}_j$, where j is derived from (19). For example, for $M=6$ and $N=10$, we find from (19) that $j=4$ for $k=0$, therefore we only need to keep the link between set Y_3 and X_E, associated with $\mathbf{P}_4$. Thus, when we solve the state equations for X_E/X_S, we directly derive the required polynomial $f^{\mathbf{P}_4}(W,U,Z,L)$ instead of the sum of polynomials $f(W,U,Z,L)$.

VI. PCCCs on the AWGN Channel

We assume that P is a rate-1/3 PCCC formed by two rate-1/2 convolutional encoders, $C1$ and $C2$, each of memory ν and a uniform interleaver of length $N-\nu$. The systematic and parity streams of the first encoder as well as the parity stream of the second encoder are punctured using the patterns $\mathbf{P}_U^{C1}$, $\mathbf{P}_Z^{C1}$ and $\mathbf{P}_Z^{C2}$, respectively. The puncturing pattern $\mathbf{P}^P$ of the PCCC assumes the form:

$$\mathbf{P}^P=\begin{bmatrix}\mathbf{P}_U^{C1}\\ \mathbf{P}_Z^{C1}\\ \mathbf{P}_Z^{C2}\end{bmatrix}=\begin{bmatrix}p_{11} & p_{12} & \cdots & p_{1M}\\ p_{21} & p_{22} & \cdots & p_{2M}\\ p_{31} & p_{32} & \cdots & p_{3M}\end{bmatrix}, \tag{20}$$

where M is the puncturing period.

The G-IOWEF of each punctured constituent code is given by:

$$\begin{aligned} B^{C1}(W,U,Z)&=\sum_{w,u,z} B_{w,u,z}^{C1} W^w U^u Z^z,\\ B^{C2}(W,U{=}1,Z)&=\sum_{w,z} B_{w,z}^{C2} W^w Z^z, \end{aligned} \tag{21}$$

respectively. The systematic stream of the second encoder is not transmitted, therefore it does not contribute to the overall weight of the turbo codewords and it is eliminated by setting $U{=}1$ in $B^{C2}(W,U,Z)$.

The G-IOWEF $B^P(W,U,Z)$ of the punctured PCCC can be derived from the G-IOWEFs $B^{C1}(W,U,Z)$ and $B^{C2}(W,U{=}1,Z)$ of the constituent codes, elaborating on the expressions in [3]. The IOWEF $B^P(W,D)$ of the punctured PCCC finally assumes the form:

$$B^P(W,D)=\sum_{w,d} B_{w,d}^P W^w D^d, \tag{22}$$

where the non-negative integers $B_{w,d}^P$ are derived by combining the G-IOWEF $B^P(W,U,Z)$ with (7).

The ML upper bound of the punctured PCCC is then approximated by:

$$P_B\le\sum_{w,d}\frac{w}{N}B_{w,d}^P Q\left(\sqrt{\frac{2R_P E_b}{N_0}\cdot d}\right) \tag{23}$$

where R_P is the code rate of the punctured PCCC.

VII. Results

The proposed technique for the derivation of the G-IOWEF of a convolutional block code was applied for the case of a rate-1/3 PCCC employing two RSC(1,5/7) constituent codes. All puncturing patterns that result in code rates of 1/2 and 2/3 are considered. Punctured turbo codes can be systematic (S-PCCC), partially systematic (PS-PCCC) or non-systematic (NS-PCCC), depending on the puncturing of the systematic output of the turbo encoder.

In Fig.7 the performance of suboptimal iterative decoding applying the BCJR decoding algorithm [13] after 10 iterations is plotted against the corresponding theoretical bound. The input block size is $N=36$ bits. Rate 1/2 and 2/3 punctured PCCCs are obtained using the patterns in Table I. Based on our technique, we found that the ML performance of rate 1/2 and 2/3 PS-PCCCs, obtained using patterns $\mathbf{P}_{P1}$ and $\mathbf{P}_{P2}$ respectively, is better than that of traditional rate 1/2 and 2/3 S-PCCCs [1], [7], obtained using $\mathbf{P}_{S1}$ and $\mathbf{P}_{S2}$ respectively. In the case of S-PCCCs, the performance of the simulated suboptimal turbo decoder converges to the corresponding ML bound. However, in the case of PS-PCCCs not only does the performance of the simulated suboptimal turbo decoder not converge to the ML bound, but the results are worse than

TABLE I

Puncturing patterns for rate 1/2 and 2/3 PCCCs

	S-PCCC			PS-PCCC		
Rate	Punc. Pattern	d_{f}	$d_{\text{f,eff}}$	Punc. Pattern	d_{f}	$d_{\text{f,eff}}$
1/2	$\mathbf{P}_{S1}=\begin{bmatrix}1111\\1010\\0101\end{bmatrix}$	3	6	$\mathbf{P}_{P2}=\begin{bmatrix}0010\\1101\\1111\end{bmatrix}$	4	7
2/3	$\mathbf{P}_{S2}=\begin{bmatrix}1111\\1000\\0010\end{bmatrix}$	2	2	$\mathbf{P}_{P2}=\begin{bmatrix}1100\\0011\\0110\end{bmatrix}$	2	4

for the S-PCCC of the same rate. These results emphasize the importance of the systematic stream, which is used by both constituent decoders of the suboptimal turbo decoder. Puncturing of the systematic stream causes severe degradation to the BEP performance, deviating markedly from the ML bound. Therefore, even if a PS-PCCC has better properties, i.e., free distance d_{f} and effective free distance $d_{\mathrm{f,eff}}$, than a S-PCCC of the same rate, the use of suboptimal iterative decoding means that the PS-PCCC does not necessarily outperform the S-PCCC.

As the input block size of a PCCC increases, the BEP performance of the code improves considerably [1], [3]. We observe that the suboptimal turbo decoder is able to cope with the punctured systematic stream as the block size is increased, and eventually the performance of a PS-PCCC converges to the ML bound. An S-PCCC achieves a worse upper bound, however it converges quicker to it. In Fig.8 we see that for a block of size $N = 1000$, PS-PCCCs perform worse than S-PCCCs at low values of E_b/N_0, however as E_b/N_0 increases, PS-PCCCs eventually outperform S-PCCCs. The same observations are also made in [14] and [15].

VIII. Conclusion

In this paper we have presented a new technique to evaluate the transfer function of a convolutional block code and we have extended it for the case of punctured convolutional block codes. The transfer function of the convolutional block code can then be used to evaluate the transfer function of a turbo code and derive an ML bound on the BEP.

We have also demonstrated that this technique can be used to derive the best puncturing patterns in terms of BEP performance. If the patterns produce a systematic turbo code, the suboptimal turbo decoder converges to the ML bound, even for small input block sizes. If the patterns produce a partially systematic turbo code, the suboptimal turbo decoder converges to the ML bound at high values of E_b/N_0 only if large block sizes, or equivalently large interleaver sizes, are used. As a consequence, high-rate punctured turbo encoders could be pattern-adaptive, i.e., select the pattern that achieves the best BEP performance, depending on the E_b/N_0 at which they operate.

Future work will include investigation and identification of constituent convolutional codes, which are more robust to puncturing of their systematic output, allowing the construction of punctured turbo codes that quickly converge to the ML bound.

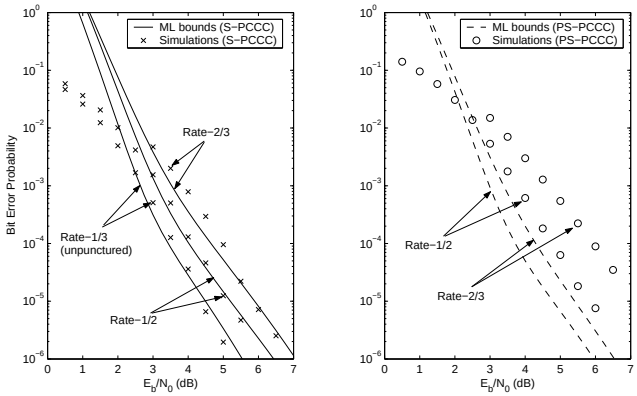

Fig. 7. Comparison between ML bounds and simulation results for rates 1/3, 1/2 and 2/3 ($N = 36$, 10 iterations).

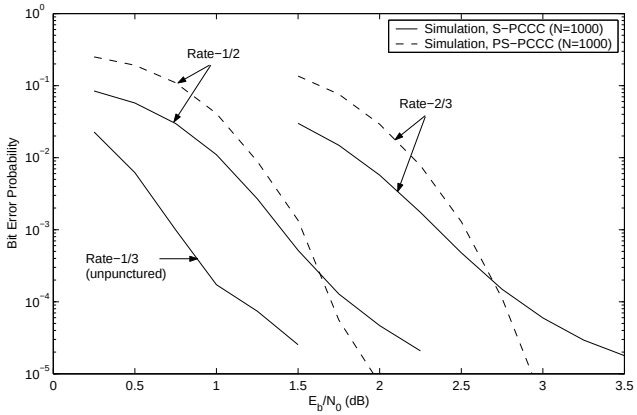

Fig. 8. Simulation results for rates 1/3, 1/2 and 2/3 ($N = 1000$, 10 iterations).

References

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

[2] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Transfer function bounds on the performance of turbo codes," JPL, Cal. Tech., TDA Progr. Rep. 42-121, Aug. 1995.

[3] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409–429, Mar. 1996.

[4] ——, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, no. 5, pp. 591–600, May 1996.

[5] Ö. Açikel and W. E. Ryan, "Punctured turbo-codes for BPSK/QPSK channels," *IEEE Trans. Commun.*, vol. 47, no. 9, pp. 1315–1323, Sept. 1999.

[6] F. Babich, G. Montorsi, and F. Vatta, "Design of rate-compatible punctured turbo (RCPT) codes," in *Proc. Int. Conf. Comm. (ICC'02)*, New York, USA, Apr. 2002, pp. 1701–1705.

[7] M. A. Kousa and A. H. Mugaibel, "Puncturing effects on turbo codes," *Proc. IEE Comm.*, vol. 149, no. 3, pp. 132–138, June 2002.

[8] M. Fan, S. C. Kwatra, and K. Junghwan, "Analysis of puncturing pattern for high rate turbo codes," in *Proc. Military Comm. Conf. (MILCOM'99)*, New Jersey, USA, Oct. 1999, pp. 547–500.

[9] I. Land and P. Hoeher, "Partially systematic rate 1/2 turbo codes," in *Proc. Int. Symp. Turbo Codes*, Brest, France, Sept. 2000, pp. 287–290.

[10] I. Chatzigeorgiou, M. R. D. Rodrigues, I. J. Wassell, and R. Carrasco, "Punctured binary turbo-codes with optimized performance," in *Proc. Vehicular Tech. Conf. (VTC-Fall'05)*, Texas, USA, Sept. 2005.

[11] A. J. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Trans. Commun. Technol.*, vol. 19, no. 5, pp. 751–772, Oct. 1971.

[12] I. Chatzigeorgiou, M. R. D. Rodrigues, I. J. Wassell, and R. Carrasco, "A novel technique for the evaluation of the transfer function of parallel concatenated convolutional codes," in *Proc. Int. Symp. Turbo Codes*, Munich, Germany, Apr. 2006 (to appear).

[13] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimising symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[14] Z. Blazek, V. K. Bhargava, and T. A. Gulliver, "Some results on partially systematic turbo codes," in *Proc. Vehicular Tech. Conf. (VTC-Fall'02)*, Vancouver, Canada, Sept. 2002, pp. 981–984.

[15] S. Crozier, P. Guinand, and A. Hunt, "On designing turbo-codes with data puncturing," in *Proc. Canadian Workshop on Inf. Theory*, Montreal, Canada, 2005.