Punctured Convolutional Codes Revisited: the Exact State Diagram and Its Implications

Jing Li (Tiffany)
Department of Electrical and Computer Engineering
Lehigh University, Bethlehem, PA 18015

Erozan Kurtas
Seagate Research
Seagate Technology, Pittsburgh, PA 15222

***Abstract*— Accurate analysis of (non-punctured) convolutional codes using the state diagram and transfer function has been well-established; but for punctured convolutional codes, approximation has been the typical treatment. This paper discusses a simple way to derive the exact state diagram and transfer function of a punctured convolutional code. The key is to represent the punctured code in an equivalent closed-form form. It is shown that such a representation is always possible and that the new code typically has the same or fewer number of states. Implications and applications of this finding on performance bounds, puncturing pattern design, decoder implementation for punctured convolutional and punctured turbo codes are also discussed.**

I. Introduction

Convolutional codes are a class of important codes due to their flexibility in code length, soft decodability (e.g. using the BCJR algorithm or soft output Viterbi algorithm (SOVA)), short decoding delay (e.g. using windowed Viterbi algorithm) and their role as component codes in parallelly/serially concatenated codes. Puncturing allows convolutional codes to flexibly change rates and is widely used in applications where high code rates are required (e.g. optical communications and digital recording systems) and where rate adaptivity is desired [1][2].

The state diagram and transfer function (SD-TF) approach is the most popular approach for performance analysis of convolutional codes (see for example, [1]). However, the method is well-defined only when the code is non-punctured, i.e. having a closed-form generator matrix. For punctured codes, most approaches reply on either heuristic search or *approximated* state diagrams. In this paper, we propose an efficient and systematic way to derive the exact state diagram and transfer function of a punctured convolutional code and subsequently extend it to turbo codes. The key is to represent the punctured code in an equivalent (non-punctured) closed form. It is shown that (i) such a transformation exists for both recursive and nonrecursive codes, and (ii) the new representation typically has the same or fewer number of states in trellis than that of the original representation (i.e. trellis of the mother code). The latter may be exploited for implementation of a simpler trellis decoder.

The research is supported by the National Science Foundation under Grant No. CCF-0430634, by Seagate Technology, and by the Commonwealth of Pennsylvania, Department of Community and Economic Development, through the Pennsylvania Infrastrusture Technology Alliance.

The proposed method makes it easier to evaluate the code performance of a punctured convolutional code (or a convolutional-based code) using maximum-likelihood (ML) bounds. It also facilitates the design for good puncturing pattern and the implementation for Slepian-Wolf codes [5].

The rest of the paper is organized as follows. Section II summarizes a few possible approximated SD-TF approaches for punctured convolutional codes. Section III discusses the proposed method of computing the exact state diagram of the punctured code. Section IV extends the discussion to code design, performance bounds and Slepian-Wolf code. Section V concludes the paper.

II. Approximated SD-TF for Punctured Codes

Since the codes we discuss are all linear codes, without loss of generality, let us use the all-zero sequence as the reference codeword. Let $A_{w,d}$ be the input-output weight enumerator, which denote the number of single error event having input weight w and output weight d. By single error event, we refer to a path in the trellis that diverges from and merges back to the all-zero sequence only once. Several of the performance bounds, including the (truncated) union bound and the Divsalar simple bound [3], require the knowledge of all or a first few terms of $A_{w,d}$. A heuristic way to obtain $A_{w,d}$ for a punctured trellis code is via exhaustive search or a walk through the trellis. This leads to the exact values of $A_{w,d}$, but is limited to only the few terms before the complexity becomes prohibitive. For a more systematic and elegant approach, or to obtain the entire spectrum of $A_{w,d}$ for the punctured code (which is needed, for example, for the Divsalar simple bound), researchers have modified the conventional SD-TF approach that was developed for non-punctured convolutional codes.

Below we briefly discuss three modified SD-FT approaches for punctured codes via a simple example. Consider a mother generator matrix $[1, \frac{1}{1+D}]$ and a puncturing pattern $[1, 1; 0, 1]$ which punctures off the odd-positioned parity check bits, leading to a code rate of $2/3$. The trellis and the open-loop state diagram of the mother code is shown in Fig. 1(A).

Modified Method I - Averaged Trellis: The simplest approximation is to account for the puncturing effect by relabeling the branch output using the average of the odd- and even-staged output. This assumes that the four branches

connecting the two states (see Fig. 1(A)) are statistically independent and equally likely. The corresponding trellis and state diagram are shown in Fig. 1(B), and the resulting transfer function can be derived using the Mason's rule [1] as

$$T_(w,d)^{(I)} = \frac{X^2Y^2 + X^2Y^3}{1-Y},$$
$$=(X^2Y^2+X^2Y^3)(1+Y+Y^2+\cdots), \quad (1)$$

where the power of X and Y denote the input and output weight, and the coefficient of term X^wY^d is $A_{w,d}$.

Modified Method II - Extended Trellis: A more sophisticated strategy is to take the odd- and even-staged trellis and combine them in an *extended* trellis through "trellis product". As shown in Fig. 1(C), the extended trellis has $2 \times 2 = 4$ states, obtained from coupling the states of two consecutive stages, i.e. states at time k–1 and k are combined to form new states $< k-1, k >$, and so are states at k and $k+1$. Similarly, the branches of the extended trellis are labeled with the cascade of inputs/outputs of the two relevant stages. The corresponding transfer function can be computed as

$$T_(w,d)^{(II)} = \frac{X^4Y^5}{1-Y-X^4Y^5},$$
$$=X^4Y^5\big(1+(Y+X^4Y^5)+(Y+X^4Y^5)^2+\cdots\big). \quad (2)$$

We note that while this method captures the puncturing nature better than the previous one, it is nevertheless not accurate. This is because that the new state diagram considers only (00) as the all-zero or reset state, whereas in reality, both (00) and (01) can be the diverging state, and both (00) and (10) can be the re-merging state.

Modified Method III - Condensed Trellis: The third method also separately considers odd- and even-staged trellis, but instead of trellis product, it squeezes trellis by taking away the states between the two stages, i.e. states at time k in Fig. 1(D). Thus the *condensed* trellis preserves the same number of states as before (2 in this case), but will instead have parallel branches between any pair of connecting states at consecutive stages. In general, if the puncturing period is t, then the condensed trellis will have 2^{t-1} parallel branches. The following transfer function can be derived from Fig. 1(D):

$$T_(w,d)^{(III)} = X^2Y^2+\frac{4X^2Y^3}{1-Y-X^2Y^3},$$
$$=X^2Y^2+4X^2Y^3(1+Y+X^2Y^3+(Y+X^2Y^3)^2+\cdots). \quad (3)$$

Although hard to prove, we note that this third method actually yields the true state diagram and transfer function (assuming input sequences have even lengths; if not, pad zeros). This will become clear after we compare the results here with those from the proposed, provenly-accurate method.

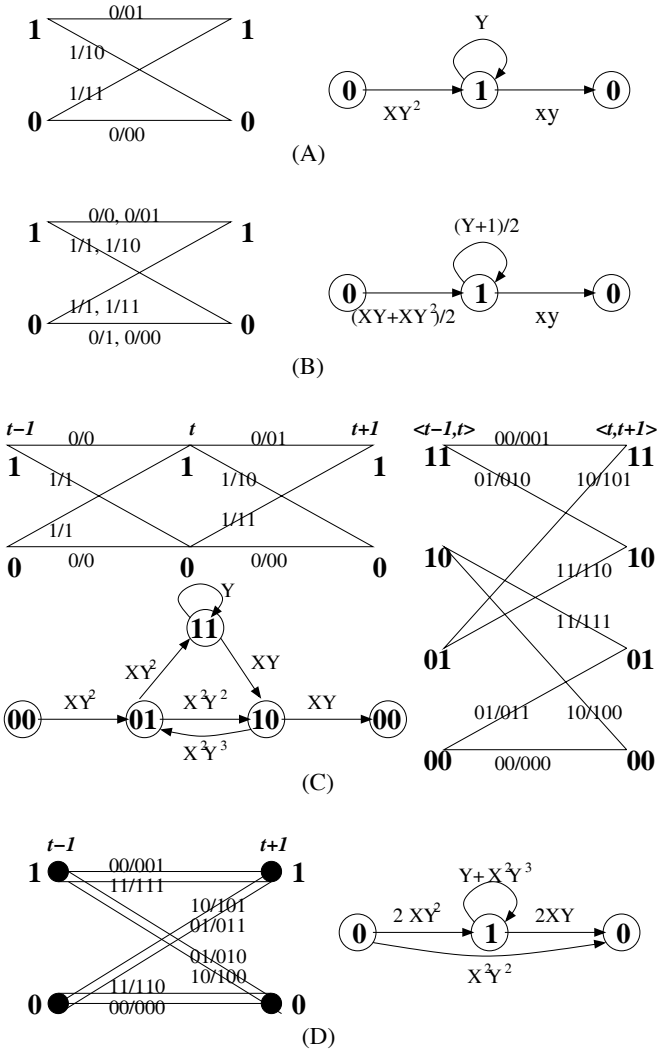

Fig. 1. (A) Non-punctured mother code. (B) Modified method I. (C) Modified method II. (D) Modified method III.

III. The Proposed Approach

A. A Non-punctured Representation for Punctured Codes

For (provenly) accurate analysis of a punctured code, we propose to first represent a punctured convolutional code (i.e. its generator matrix) in an equivalent closed-form (non-punctured) representation and to subsequently apply the convolutional SD-TF approach. Since punctured convolutional codes are still linear codes and since the coded bits still exhibit periodic relations, it should not be surprising that a punctured convolutional code should also be implementable using "non-punctured" finite-state linear circuits and could therefore adopt a closed-form generator matrix. Below we introduce an efficient method to convert a general punctured convolutional code to a non-punctured equivalency.

Consider a punctured convolutional code, $\mathbf{C}_p$, obtained

from puncturing a rate $R_0 = k/n$ mother code $\mathbf{C}_0$. Let G_0 denote the $k \times n$ generator matrix of $\mathbf{C}_0$ and let t denote the puncturing period. The new closed-form representation of $\mathbf{C}_p$, G_p, can be obtained by first expanding G_0 to an equivalent matrix G_t of dimensionality $kt \times nt$, and then extracting the unwanted columns (i.e. the columns corresponding to the punctured bits). The question is how to efficiently expand an arbitrary generator matrix by a factor of t in rows and columns.

Before proceeding to the main result, let us first discuss the notations. Let $\bar{a} \triangleq [a_0, a_1, \cdots, a_m]$ be a binary sequence. Let $A(D) = \sum_{i=0}^{m} a_i D^i$ be its equivalent representation in the $\mathcal{D}$-domain. The sequence $\bar{a}$ can be split in t sub sequences with respect to the modulo-t positions, termed the *cyclic-t phases*:

$$\begin{aligned} \bar{a}_0^{(t)} &\triangleq [a_0, a_t, a_{2t}, \cdots], \\ \bar{a}_1^{(t)} &\triangleq [a_1, a_{t+1}, a_{2t+1}, \cdots], \\ \cdots &\quad \cdots \\ \bar{a}_{t-1}^{(t)} &\triangleq [a_{t-1}, a_{2t-1}, a_{3t-1}, \cdots]. \end{aligned}$$

The corresponding $\mathcal{D}$-domain representations of these phases are given by

$$A_j^{(t)}(D) = \sum_{i=0}^{n} a_{it+j} D^i, \quad j = 0, 1, \cdots, t-1. \tag{4}$$

Clearly, the entire sequence $A(D)$ can adopt a poly-phase representation using $A_j^{(t)}(D)$'s as:

$$A(D) = A_0^{(t)}(D^t) + DA_1^{(t)}(D^t) + \cdots + D^{t-1}A_{t-1}^{(t)}(D^t). \tag{5}$$

For notational convenience, the superscript (t) will be omitted where there is no confusion.

Define $\Gamma_A^{(t)}(D) \triangleq$

$$\begin{bmatrix} A_0(D) & A_1(D) & \cdots & A_{t-2}(D) & A_{t-1}(D) \\ DA_{t-1}(D) & A_0(D) & \cdots & A_{t-3}(D) & A_{t-2}(D) \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ DA_1(D) & DA_2(D) & \cdots & DA_{t-1}(D) & A_0(D) \end{bmatrix},$$

as the *t-cyclic phase elementary matrix*, or simply, the *elementary matrix*, of a feed-forward polynomial $A(D)$. We see that $\Gamma_A^{(t)}(D)$ is a $t \times t$ square matrix that contains t^2 $\mathcal{D}$-domain feed-forward polynomials. It is easy to verify that $\Gamma_A^{(t)}(D)$ has the following properties:

Lemma: (Properties of a t-cyclic phase elementary matrix)

- $\Gamma_A^{(t)}(D)$ is a full rank matrix for all $A(D) \neq 0$.
- If $A(D) = 1$, then $\Gamma_A^{(t)}(D)$ is an identity matrix.

Theorem: For a rate k/n convolutional code with generator matrix $G(D) \triangleq [G_{ij}(D)]_{k \times n}$, where each entry $G_{ij}(D) = U_{ij}(D)/V_{ij}(D)$, and $U_{ij}(D)$ and $V_{ij}(D)$ are feed-forward polynomials, the equivalent expanded generator matrix G_t can be obtained by replacing each entry $G_{ij}(D)$, with a $t \times t$ square matrix $\Gamma_{U_{ij}}^{(t)}(D)(\Gamma_{V_{ij}}^{(t)}(D))^{-1}$.

Proof: It is enough to prove that each element $F_{ij}(D) = U_{ij}(D)/V_{ij}(D)$ can be expanded to a $t \times t$ square matrix $\Gamma_{U_{ij}}^{(t)}(D)(\Gamma_{V_{ij}}^{(t)}(D))^{-1}$. For notational simplicity, let us drop the subscripts ij. Let $X(D) = X_0(D^t) + DX_1(D^t) + \cdots + D^{t-1}X_{t-1}(D^t)$ be the input sequence and $Y(D) = Y_0(D^t) + DY_1(D^t) + \cdots + D^{t-1}Y_{t-1}(D^t)$ be the corresponding output sequence, i.e.,

$$Y(D) = X(D)\frac{U(D)}{V(D)}. \tag{6}$$

We need to show that

$$\begin{aligned} &\big[Y_0(D), Y_1(D), \cdots, Y_{t-1}(D)\big] = \\ &\quad \big[X_0(D), X_1(D), \cdots, X_{t-1}(D)\big]\Gamma_U^{(t)}(D)\big(\Gamma_V^{(t)}(D)\big)^{-1}, \end{aligned} \tag{7}$$

Expanding (6) yields

$$\begin{aligned} &\big(Y_0(D^t) + DY_1(D^t) + \cdots + D^{t-1}Y_{t-1}(D^t)\big)\big) \\ &\quad \big(V_0(D^t) + DV_1(D^t) + \cdots + D^{t-1}V_{t-1}(D^t)\big)\big) \\ &= \big(X_0(D^t) + DX_1(D^t) + \cdots + D^{t-1}X_{t-1}(D^t)\big) \\ &\quad \big(U_0(D^t) + DU_1(D^t) + \cdots + D^{t-1}U_{t-1}(D^t)\big). \end{aligned} \tag{8}$$

Gathering the terms with respect to the order of the power, we obtain

$$\begin{aligned} &X_0(D^t)U_0(D^t) + D^t X_1(D^t)U_{t-1}(D^t) + \\ &\quad D^t X_2(D^t)U_{t-2}(D^t) + \cdots + D^t X_{t-1}(D^t)U_1(D^t) \\ &= Y_0(D^t)V_0(D^t) + D^t Y_1(D^t)V_{t-1}(D^t) + \\ &\quad D^t Y_2(D^t)V_{t-2}(D^t) + \cdots + D^t Y_{t-1}(D^t)V_1(D^t), \\ &DX_0(D^t)U_1(D^t) + DX_1(D^t)U_0(D^t) + \\ &\quad D^{t+1}X_2(D^t)U_{t-1}(D^t) + \cdots + D^{t+1}X_{t-1}(D^t)U_2(D^t) \\ &= DY_0(D^t)V_1(D^t) + DY_1(D^t)V_0(D^t) + \\ &\quad D^{t+1}Y_2(D^t)V_{t-1}(D^t) + \cdots + D^{t+1}Y_{t-1}(D^t)V_2(D^t), \\ &\cdots \\ &D^{t-1}X_0(D^t)U_{t-1}(D^t) + D^{t-1}X_1(D^t)U_{t-2}(D^t) + \\ &\quad D^{t-1}X_2(D^t)U_{t-3}(D^t) + \cdots + D^{t-1}X_{t-1}(D^t)U_0(D^t) \\ &= D^{t-1}Y_0(D^t)V_{t-1}(D^t) + D^{t-1}Y_1(D^t)V_{t-2}(D^t) + \\ &\quad D^{t-1}Y_2(D^t)V_{t-3}(D^t) + \cdots + D^{t-1}Y_{t-1}(D^t)V_0(D^t), \end{aligned}$$

which can be simplified to:

$$\begin{aligned} &X_0(D)U_0(D) + DX_1(D)U_{t-1}(D) + \\ &\quad DX_2(D)U_{t-2}(D) + \cdots + DX_{t-1}(D)U_1(D) \\ &, = Y_0(D)V_0(D) + DY_1(D)V_{t-1}(D) + \\ &\quad DY_2(D)V_{t-2}(D) + \cdots + DY_{t-1}(D)V_1(D) \end{aligned} \tag{9}$$

$$\begin{aligned} &X_0(D)U_1(D) + X_1(D)U_0(D) + \\ &\quad DX_2(D)U_{t-1}(D) + \cdots + DX_{t-1}(D)U_2(D) \\ &= Y_0(D)V_1(D) + Y_1(D)V_0(D) + \\ &\quad DY_2(D)V_{t-1}(D) + \cdots + DY_{t-1}(D)V_2(D), \end{aligned} \tag{10}$$

$$
\begin{aligned}
&\cdots\\
&X_0(D)U_{t-1}(D) + X_1(D)U_{t-2}(D) +\\
&\quad X_2(D)U_{t-3}(D) + \cdots + X_{t-1}(D)U_0(D)\\
=&Y_0(D)V_{t-1}(D) + Y_1(D)V_{t-2}(D) +\\
&\quad Y_2(D)V_{t-3}(D) + \cdots + Y_{t-1}(D)V_0(D). \qquad (11)
\end{aligned}
$$

Rewrite the above linear equations in a matrix form, we get

$$
\begin{aligned}
&\big[Y_0(D), Y_1(D), \cdots, Y_{t-1}(D)\big]\\
&\times \underbrace{\begin{bmatrix} V_0(D), & V_1(D), & \cdots, & V_{t-1}(D)\\ DV_{t-1}(D), & V_0(D), & \cdots, & V_{t-2}(D)\\ DV_{t-2}(D), & DV_{t-1}(D), & \cdots, & V_{t-3}(D)\\ \cdots, & \cdots, & \cdots, & \cdots\\ DV_1(D), & DV_2(D), & \cdots, & V_0(D)\end{bmatrix}}_{\Gamma_V^{(t)}(D)}\\
=&\big[X_0(D), X_1(D), \cdots, X_{t-1}(D)\big]\\
&\times \underbrace{\begin{bmatrix} U_0(D), & U_1(D), & \cdots, & U_{t-1}(D)\\ DU_{t-1}(D), & U_0(D), & \cdots, & U_{t-2}(D)\\ DU_{t-2}(D), & DU_{t-1}(D), & \cdots, & U_{t-3}(D)\\ \cdots, & \cdots, & \cdots, & \cdots\\ DU_1(D), & DU_2(D), & \cdots, & U_0(D)\end{bmatrix}}_{\Gamma_U^{(t)}(D)}
\end{aligned}
$$

Since $\mathcal{V}^{(t)}(D)$ is a full-rank square matrix, it follows that (7) holds. □

We note that the above practice of expanding a generator matrix is applicable to any convolutional code, be it systematic or non-systematic, recursive or non-recursive. For example, when the code is a non-recursive convolutional code, i.e., $V_{ij}(D) = 1$ for all $1 \le i \le k$ and $1 \le j \le n$, then each term in the generator matrix becomes $G_{ij}(D) = U_{ij}(D)/V_{ij}(D) = U_{ij}(D)$, and the corresponding $t \times t$ expansion is

$$\Gamma_{U_{ij}}^{(t)}(D)(\Gamma_{V_{ij}}^{(t)}(D))^{-1} = \Gamma_{U_{ij}}^{(t)}(D)I_k(D) = \Gamma_{U_{ij}}^{(t)}(D). \quad (12)$$

B. Examples

Example I: Consider the following example where a rate 3/4 recursive systematic convolutional (RSC) code is punctured from a rate 1/2 mother code with generator matrix $[1, \frac{1+D+D^2+D^3}{1+D^2+D^3}]$ and puncturing pattern $[1, 1; 0, 1]$ (see Fig. 2(A)).

Let $U(D) \triangleq 1 + D + D^2 + D^3$ and $V(D) \triangleq 1 + D^2 + D^3$. The 2-cyclic phases of $U(D)$ and $V(D)$ are

$$U_0(D) = 1 + D, \qquad U_1(D) = 1 + D, \qquad (13)$$

$$V_0(D) = 1 + D, \qquad V_1(D) = D, \qquad (14)$$

which lead to elementary matrices:

$$
\begin{aligned}
\Gamma_U^{(2)}(D) &= \begin{bmatrix} 1+D, & 1+D\\ D+D^2, & 1+D\end{bmatrix},\\
\Gamma_V^{(2)}(D) &= \begin{bmatrix} 1+D, & D\\ D^2, & 1+D\end{bmatrix},
\end{aligned}
$$

According to the Theorem, the expanded matrix of G is given by

$$
\begin{aligned}
G_t &= \begin{bmatrix} I_2, & \Gamma_U^{(2)}\left(\Gamma_V^{(2)}\right)^{-1}\end{bmatrix}_{2\times 4}\\
&= \left[I_2, \begin{bmatrix} 1+D, & 1+D\\ D+D^2, & 1+D\end{bmatrix}\begin{bmatrix} \frac{1+D}{1+D^2+D^3}, & \frac{D}{1+D^2+D^3}\\ \frac{D^2}{1+D^2+D^3}, & \frac{1+D}{1+D^2+D^3}\end{bmatrix}\right]\\
&= \begin{bmatrix} 1, & 0, & \frac{1+D^3}{1+D^2+D^3}, & \frac{1+D}{1+D^2+D^3}\\ 0, & 1, & \frac{D+D^2}{1+D^2+D^3}, & \frac{1+D^3}{1+D^2+D^3}\end{bmatrix} \qquad (15)
\end{aligned}
$$

Puncturing the parity bits in odd positions means deleting the last column of G_t. Hence, the punctured code takes the following closed form

$$\begin{bmatrix} 1, & 0, & \frac{1+D}{1+D^2+D^3}\\ 0, & 1, & \frac{1+D^3}{1+D^2+D^3}\end{bmatrix}. \qquad (16)$$

The linear circuits implementation of the punctured code in non-closed-form and closed-form are shown in Fig 2.

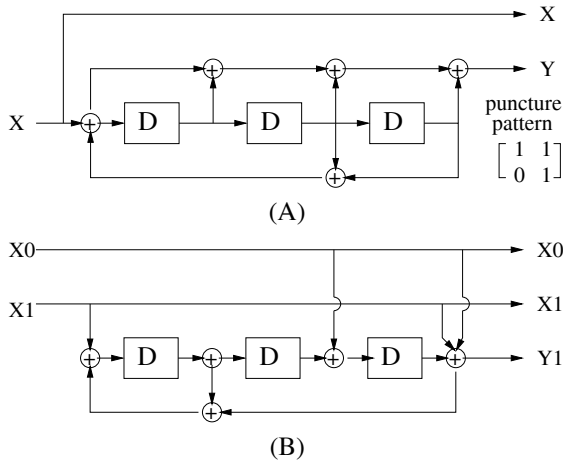

Fig. 2. A rate 2/3 punctured RSC code. (A) Punctured representation. (B) The equivalent (non-punctured) closed form.

Example II: In the above example, the new closed-form representation (Fig. 2(B)) results in the same number of states as that of the mother code (Fig. 2(A)). It is also possible for the expanded generator matrix to have fewer number of states. This happens when the code is recursive and when (all) the feedback polynomial(s) are*factorisable*. For example, an RSC convolutional code with generator matrix $[1, \frac{1+D^2+D^3}{1+D+D^2+D^3}]$ has memory 3 and hence 8 states in the trellis. Its equivalent expanded generator matrix takes the form of:

$$\begin{bmatrix} 1, & 0, & \frac{1}{1+D^2}, & \frac{1}{1+D^2}\\ 0, & 1, & \frac{D}{1+D^2}, & \frac{1}{1+D^2}\end{bmatrix}. \qquad (17)$$

Thus, the linear circuits implementation of the new representation (as well as any of its punctured codes) has only 2 delay elements, leading to only 4 states in the trellis. This feature may be exploited for implementation of a

simpler decoder for convolutional codes and turbo codes. We note that for turbo codes, the component RSC codes are typically chosen to have primitive (and hence non-factorisable) feedback polynomials. Nevertheless, as pointed out in [4], non-primitive feedback polynomials may also be desirable in certain cases, since although they make the turbo code perform worse at the water-fall region, but the error floor appears at a lower stage.

Example III: Consider the same example we used in Section II, i.e., $G(D) = [1, 1/(1+D)]$ generator matrix with $[1, 1; 0, 1]$ puncturing pattern. The closed-form representation of the punctured code is obtained by deleting the third column of the expanded matrix

$$G_t = \begin{bmatrix} 1, & 0, & \frac{1}{D}, & \frac{1}{D}, \\ 0, & 1, & 1, & \frac{1}{D} \end{bmatrix}. \tag{18}$$

We see that it will produce the same state diagram and transfer function as Method III in Section II (see Fig. 1(C)). This indicates that the heuristic "condensed trellis" approach is also accurate, but the proposed method can be rigorously proven. Further, as discussed in Section IV, the availability of closed-form generator matrices make it convenient to exploit punctured convolutional/turbo codes in other applications including Slepian-Wolf coding [5].

IV. Implications and Applications

Among other applications, the proposed method facilitates performance analysis using ML bounds, code design, decoder implementation and Slepian-Wolf coding.

Performance Bounds: The general form of union bounds on bit error rate is given by

$$P_b \leq \sum_w w \sum_d A_{w,d} P_2(d), \tag{19}$$

where $A_{w,d}$ is the IOWE that is dependent on the code, and $P_2(d)$ is the pair-wise error probability (PEP) that is dependent on the channel. For additive white Gaussian noise (AWGN) channels,

$$P_2(d) = Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right) \leq \frac{1}{2}\exp\left(-\frac{dRE_b}{N_0}\right). \tag{20}$$

For convolutional codes, $A_{w,h}$ can be obtained from the coefficients of the transfer function. For turbo codes formed from parallel or serial concatenation of two convolutional codes, $A_{w,h}$ can be computed using the concept of a *uniform interleaver*:

$$\text{Parallel}: \quad A_{w,d} = \sum_{d_1+d_2=d} \frac{A^{(1)}_{w,d_1} A^{(2)}_{w,d_2}}{\binom{L}{w}}, \tag{21}$$

$$\text{Serial}: \quad A_{w,d} = \sum_{l=1}^{L} \frac{A^{(o)}_{w,l} A^{(i)}_{l,d}}{\binom{L}{l}} \tag{22}$$

where L is the interleaver size, $A^{(1)}$ and $A^{(2)}$ are the IOWE's for the first and second component code of the parallel code, and $A^{(o)}$ and $A^{(i)}$ are the IOWE's for the outer and inner component code of the serial code. Hence, using the exact transfer function and the above formulas, union bounds for punctured convolutional codes as well as punctured serial/parallel turbo codes can be conveniently evaluated. For example, the performance of a convolutional code with transfer function $T_{w,d}(X, Y)$ over AWGN channel is upper bounded by:

$$P_b \leq \frac{1}{2} \left.\frac{\partial T_{w,d}(X,Y)}{\partial X}\right|_{x=1, Y=\exp(-RE_b/N_0)}, \tag{23}$$

where $T_{w,d}(X, Y) = X^2Y^2 + \frac{4X^2Y^3}{1-Y-X^2Y^3}$.

Puncturing Pattern: The performance of a punctured convolutional/turbo code not only depends on the mother code, but also the puncturing pattern [2]. Computer search, accompanied with lengthy Monte-Carlo simulation, has been the conventional method of searching for good puncturing pattern. Now that a simple and systematic way is available to evaluate the distance spectrum of a punctured code, simulation is no longer needed and the search process can be considerably expedited as well as made more accurate.

Decoder Implementation: As mentioned previously, since the new expanded generator matrix may have fewer states than the original generator matrix, a simpler trellis decoder may be implemented for the mother code as well as any of its punctured codes.

Slepian-Wolf Coding: Additionally, the availability of a closed-form representation for punctured convolutional/turbo codes also makes them readily applicable in Slepian-Wolf coding using the SF-ISF framework[5]. Specifically, using the simple construction method described in [5], syndrome formers (SF) and inverse syndrome formers (ISF) can be conveniently derived from the closed-form generator matrices, and subsequently form an efficient Slepian-Wolf source codec.

V. Conclusion

A efficient and general method is proposed to derive the equivalent closed-form generator matrix, and subsequently the exact state-diagram and transfer function, of a punctured convolutional code. The proposed method has implication and application in a number of places including code design and analysis for both punctured convolutional codes and punctured turbo codes.

References

[1] S. Lin, and D. J. Costello, *Error Control Coding*, Prentice Hall, 2nd edition, 2004.

[2] R. D. Wesel, X. Liu, and W. Shi, "Trellis Codes for Periodic Erasures," *IEEE Trans. Commun.*, pp 938-947, June 2000.

[3] D. Divsalar, "A simple tight bound on error probability of block codes with application to turbo codes," *TMO Progress Report 42-139*, Nov. 1999.

[4] O. Takeshita, O. Collins, P. Massey and D.Costello Jr, "A note on asymmetric turbo codes," *IEEE Comm. Let*, pp. 69-71, March, 1999.

[5] Z. Tu, J. Li, and R. S. Blum, "An effi cient SF-ISF approach for the Slepian-Wolf source coding problem," to appear *Eurasip J. on Applied Signal Processing*.