Tailbiting MAP Decoders

John B. Anderson, *Fellow, IEEE*, and Stephen M. Hladik, *Member, IEEE*

***Abstract*—We extend the MAP decoding algorithm of Bahl *et al.* to the case of tailbiting trellis codes. An algorithm is given that is based on finding an eigenvector, and another that avoids this. Several examples are given. The algorithm has application to turbo decoding and source-controlled channel decoding.**

***Index Terms*— Decoding, iterative decoding, MAP decoding, trellis decoding.**

I. Introduction

The calculation of probability information, or "soft output," during data decoding is the object of a MAP (maximum *a posteriori* probability) decoder. One kind of MAP decoder is the BCJR algorithm, named after the authors of an early paper [1] that proposed the scheme. The BCJR algorithm is a MAP decoder for trellis codes and a Markov data source. The scheme's output is the probability of each trellis state at each stage, given the observed channel outputs; alternately, the output can be the probability of each trellis transition. From this information can be derived individual data *a posteriori* probabilities. The BCJR algorithm is distinct from the Viterbi algorithm, which determines the most likely *sequence* of trellis states. The BCJR algorithm is a component in iterative decoding methods, such as turbo decoding [4] and replication decoding [5], which pass likelihood information between parts of the decoding in the form of state and symbol probabilities.

The algorithm given in [1], and all of its applications until now, require that the distribution of the starting and ending trellis states be known, or at least assigned an *a priori* probability. Usually, a state such as the 0 state is assigned probability 1 at both ends of the trellis. Forcing the encoded trellis path to a certain state at the end of decoding is called terminating the path. Forcing is accomplished by transmitting known trellis branches, a procedure that reduces the encoding rate, particularly if the block length is short. Without path termination, a decoder has a weaker error probability near the end of the decoded sequence.

An alternative to path termination that does not reduce the rate or degrade the error probability is tailbiting. In this encoding technique, the start and end encoder states are constrained to be identical; that is, a trellis codeword starts from the state at which it will eventually end. The state is otherwise unknown *a priori*. The transmission can be viewed as a path around a circular trellis, as shown in Fig. 1 for a four-state trellis. An early paper with a full explanation of tailbiting is Ma and Wolf [6].

Manuscript received September 10, 1996; revised May 2, 1997. This paper was presented in part at the IEEE International Symposium on Information Theory, Ulm, Germany, July 1997.

J. B. Anderson is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180-3590 USA.

S. M. Hladik is with the Communications Program, General Electric Research and Development Center, Schenectady, NY 12309 USA.

Publisher Item Identifier S 0733-8716(98)00169-3.

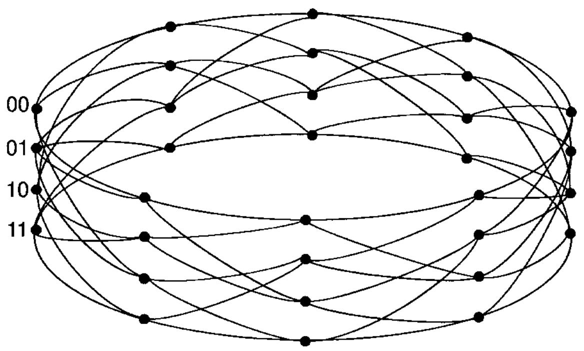


Fig. 1. Example of circular tailbiting trellis with four states.

To process a received trellis codeword, a BCJR decoder must begin from the conditional distributions of the start and end state, given the observed channel output; these two states are both state 0 with probability 1 in terminated decoding. With a tailbiting codeword, the distributions are unknown. Although a BCJR decoder could begin here with uniform start and end distributions $(1/M, \cdots, 1/M)$, with M the number of states, the result would be inferior to working with an (as yet uncalculated) true distribution of the first trellis state. In this paper, we extend BCJR to the case where meaningful start and end state distributions are unknown at the outset. Our schemes thus allow tailbiting to be applied in iterative decoding and decoding with *a priori* source data probabilities, as has been proposed by Hagenauer [2]. Tailbiting allows these methods to be used with short codes without rate loss. We present the basic algorithm and two practical extensions in Sections II and III, and give examples of their application in Section IV.

II. The Tailbiting BCJR Algorithm

The object of the basic BCJR algorithm is to compute the probabilities that a state m is sent at time t, given that the observed channel outputs are $y_1, \cdots, y_L$; that is, it seeks

$$\Pr\{S_t = i | Y_1^L\} \tag{1}$$

where $S_t = i$ denotes the event that the encoder is in state i at trellis stage t, and Y_1^L is shorthand for the variables $Y_1, \cdots, Y_L$. Note that Y_t^L applies to stage t, and in the case of the rate-$1/a$ convolutional encoders, a an integer, that are used as examples in this paper, Y_t comprises a channel outputs. Instead of finding (1), BCJR actually finds

$$\lambda_t(i) = \Pr\{S_t = i, Y_1^L\} \tag{2}$$

which is the product of (1) and $\Pr\{Y_1^L\}$, the probability that Y_1^L is observed.

Following [1], we define three sets of working probabilities. It is convenient to express these as matrices and vectors. Define the matrix

$$\Gamma_t(i,j) \triangleq \Pr\{S_t = j, Y_t | S_{t-1} = i\}, \qquad t = 1, \cdots, L \tag{3}$$

where i, j run over the M states. Define the row vector

$$\alpha_t(i) \triangleq \Pr\{S_t = i, Y_1^t\}, \qquad t = 1, \cdots, L. \tag{4}$$

Define the column vector

$$\beta_t(j) \triangleq \Pr\{Y_{t+1}^L | S_t = j\}, \qquad t = 1, \cdots, L-1. \tag{5}$$

The Γ_t function is the input to the algorithm, and α_t and β_t are computed during the progress of the algorithm. Given a set of Γ_t, the steps of the basic BCJR are as follows.

1) Form the row vectors $\alpha_1, \cdots, \alpha_L$ by the *forward recursion*

$$\alpha_t = \alpha_{t-1}\Gamma_t, \qquad t = 1, \cdots, L. \tag{6}$$

2) Form the column vectors $\beta_{L-1}, \cdots, \beta_1$ by the *backward recursion*

$$\beta_t = \Gamma_{t+1}\beta_{t+1}, \qquad t = L-1, \cdots, 1. \tag{7}$$

3) Form the row vectors $\lambda, \cdots, \lambda_L$ by

$$\lambda_t = \alpha_t \cdot \beta_t, \qquad t = 1, \cdots, L. \tag{8}$$

Here, the operator "$\cdot$" means component-by-component multiplication. From the $\lambda_t, \Pr\{S_t = i | Y_1^L\} = \lambda_t(i)/\Pr\{Y_1^L\}$ may be found by forming $\Sigma_i \Pr\{S_t = i, Y_1^L\} = \Sigma_i \lambda_t(i)$, which is $\Pr\{Y_1^L\}$.

For the recursions (6) and (7) to apply at $t = 1$ and $L-1$, respectively, it must be that $\alpha_o(i)$ is the probability the encoder starts in state i before stage 1 and $\beta_L(j)$ is the probability it ends in state j after stage L. In the standard application of the BCJR algorithm, $\alpha_o = (1, 0, \cdots, 0)$ initializes the forward and $\beta_L = (1, 0, \cdots, 0)$ initializes the backward recursion; these are equivalent to assuming that the encoder starts and ends in state 0. Note that the sum of the components of an α_t is the probability that Y_1^t is observed; if the vector is normalized to unit sum, it becomes the probability of the states at time t, given the observation Y_1^t up to t. The BCJR also computes the probability of a state transition from i to j during stage t; this is given by

$$\begin{aligned}\sigma_t(i \to j) &\triangleq \Pr\{S_{t-1} = i, S_t = j, Y_1^L\} \\ &= \alpha_{t-1}(i)\Gamma_t(i,j)\beta_t(j).\end{aligned} \tag{9}$$

That λ and σ are truly the probabilities claimed is proven in [1].

The BCJR procedure assumes that the trellis evolves in a Markov fashion and that the channel noise is memoryless. Under such assumptions, the probability $\Gamma_t(i,j)$ in (3) becomes

$$\begin{aligned}&\Pr\{S_t = j, Y_t | S_{t-1} = i\} \\ &\quad = \Pr\{Y_t, \text{trellis transition } i \to j\} \\ &\quad = \Pr\{Y_t | i \to j\} \Pr\{i \to j\}.\end{aligned} \tag{10}$$

In Section IV, we specialize (10) to rate-$1/a$ convolutional codes; then the state transitions are driven by binary data symbols, and $\Pr\{i \to j\}$ becomes the *a priori* probability of a data bit. The factor $\Pr\{Y_t | i \to j\}$ becomes the probability of a set of a channel outputs, given the set of a encoder outputs.

We turn now to the new tailbiting BCJR algorithm. Now, α_o is unknown and must be computed by the algorithm. The new algorithm depends on the fact that

$$\alpha_o = \frac{\alpha_o \Gamma_1 \cdots \Gamma_L}{\Pr\{Y_1^L\}}. \tag{11}$$

To prove this, we begin with an arbitrary starting distribution and iterate (6) L times, to get

$$\alpha_L = \alpha_o \Gamma_1 \cdots \Gamma_L. \tag{12}$$

Equation (12) holds for any starting distribution. Consider a particular one, namely

$$\alpha_o = \Pr\{S_o = j | Y_1^L\} = \Pr\{S_o = j, Y_1^L\}/\Pr\{Y_1^L\}$$

which is the distribution of S_o given all of the observed channel outputs. Because of tailbiting, $S_o \equiv S_L$, so that

$$\Pr\{S_o = j, Y_1^L\} = \Pr\{S_L = j, Y_1^L\} = \alpha_L(j), \qquad \text{all } j.$$

Consequently

$$\Pr\{Y_1^L\}\alpha_o = \alpha_L = \alpha_o \Gamma_1 \cdots \Gamma_L.$$

It is assumed that $\Gamma_1 \cdots \Gamma_L$ satisfies certain regularity conditions; some of these are given in Section IV.

Before listing the algorithm, we comment further on the solution of (11). From (11), α_o is the normalized left eigenvector of the matrix $\Gamma_1 \cdots \Gamma_L/\Pr\{Y_1^L\}$ for which the eigenvalue is 1. Such an eigenvalue must exist because (11) is an equation of probabilities; furthermore, no larger eigenvalue can exist (see, for example, [3]). Equivalently, α_o is the eigenvector of the largest eigenvalue of $\Gamma_1 \cdots \Gamma_L$, and its eigenvalue must be $\Pr\{Y_1^L\}$. However, we need not explicitly find $\Pr\{Y_1^L\}$ in order to find α_o, and it is convenient not to since $\Pr\{Y_1^L\}$ becomes very small as L grows. For example, as the product $\Gamma_1 \cdots \Gamma_L$ is built up step by step, we can scale by a convenient factor at each step in order to keep the matrix elements at a reasonable size. The direction of the eigenvector will remain unchanged, and at the end, the vector can be scaled to unit size. Some further simplification comes from the fact that we seek only the largest eigenvalue eigenvector and that the Γ matrices are very sparse. For both of these reasons, the eigensolution by no means has the complexity of the full eigenvector problem [3].

One detail remains, the initializing of the backward recursion. Several initial β_L have been proposed. We will argue in a forthcoming paper devoted to convolutional MAP decoding that the *right* eigenvector of $\Gamma_1 \cdots \Gamma_L$ is the proper starting vector in most applications.

A1: The Tailbiting BCJR Algorithm

We incorporate these facts into a procedure to find $\lambda_1, \cdots, \lambda_L$ for a tailbiting trellis. The scheme finds and uses $\Pr\{Y_1^L\}$; Algorithm A2 to follow avoids this.

Given $y_1, \cdots, y_L$.

1) Find the left eigenvector corresponding to the largest eigenvalue of $\Gamma_1 \cdots \Gamma_L$; scale it to sum to unity. This is α_o. The eigenvalue is $\Pr\{Y_1^L\}$.

2) Form the row vectors $\alpha_1, \cdots, \alpha_L$ by the forward recursion

$$\alpha_t = \alpha_{t-1}\Gamma_t, \qquad t = 1, \cdots, L-1.$$

3) Form the column vectors $\beta_{L-1}, \cdots, \beta_1$ by the backward recursion

$$\beta_t = \Gamma_{t+1}\beta_{t+1}, \qquad t = L-1, \cdots, 1$$

with β_L equal the right eigenvector of $\Gamma_1 \cdots \Gamma_L$.

4) Form $\lambda_t = \alpha_t \cdot \beta_t, t = 1, \cdots, L$.

From λ_t may be found $\Pr\{S_t = i | Y_1^L\}$ as explained before, and $\sigma_t(i \to j)$ may be found as in (9).

In most BCJR applications, $\Pr\{Y_1^L\}$ is used only to scale λ. When $\Pr\{Y_1^L\}$ is not explicitly required, the following procedure, which uses successive renormalizations, controls the precision of the calculations and leads to the identical $\Pr\{S_t = i | Y_1^L\}$. The scheme depends on the fact that scaling of any or all of the α_t, β_t, or Γ_t during recursions (6) or (7) simply leads to scaled λ_t in (8); but $\Pr\{S_t = i | Y_1^L\}$ is always available by normalizing λ_t to have unit sum.

In what follows, the α_t and β_t are renormalized to unit sum at every recursion step; the superscript "o" indicates a unit-sum vector.

A2: Tailbiting BCJR Algorithm (with Renormalizing)

Given $y_1, \cdots, y_L$.

1) Find the left eigenvector corresponding to the largest eigenvalue of $\Gamma_1 \cdots \Gamma_L$. This is α_o. Do not find the eigenvalue. Any renormalizing is allowed during the calculation.

2) Form the normalized row vectors $\alpha_o^o, \cdots, \alpha_L^o$ by repeating the recursion

$$\alpha_t = \alpha_{t-1}^o \Gamma_t$$
$$\alpha_t^o = \alpha_t \Big/ \sum_i \alpha_t(i), \qquad t = 1, \cdots, L-1.$$

3) Starting from the right eigenvector as in A2, form the normalized column vectors by repeating

$$\beta_t = \Gamma_{t+1}\beta_{t+1}^o$$
$$\beta_t^o = \beta_t \Big/ \sum_j \beta_t(j), \qquad t = L-1, \cdots, 1.$$

4) Form

$$\lambda_t = \alpha_t^o \cdot \beta_t^o$$
$$\lambda_t^o = \lambda_t \Big/ \sum_i \lambda_t(i), \qquad t = 1, \cdots, L.$$

III. A Tailbiting BCJR Without the Eigenvector

The renormalizing scheme just presented contains within it a way to avoid an explicit solution for an eigenvector. The fact is that if we iterate the forward recursion (6) enough times and properly renormalize the outcomes, then the resulting sequence of outcomes $\alpha_1, \alpha_2, \cdots, \alpha_L, \alpha_{L+1}, \cdots, \alpha_{2L}, \alpha_{2L+1}, \cdots$ will converge to repetitions of $\alpha_1^o, \cdots, \alpha_L^o$. This is because repeated multiplication of a vector by a matrix G gives a vector that converges in ratio to $\lambda_{\max}^n v$, where $\lambda_{\max}$ is the largest eigenvalue of G and v is its eigenvector (see [3]).

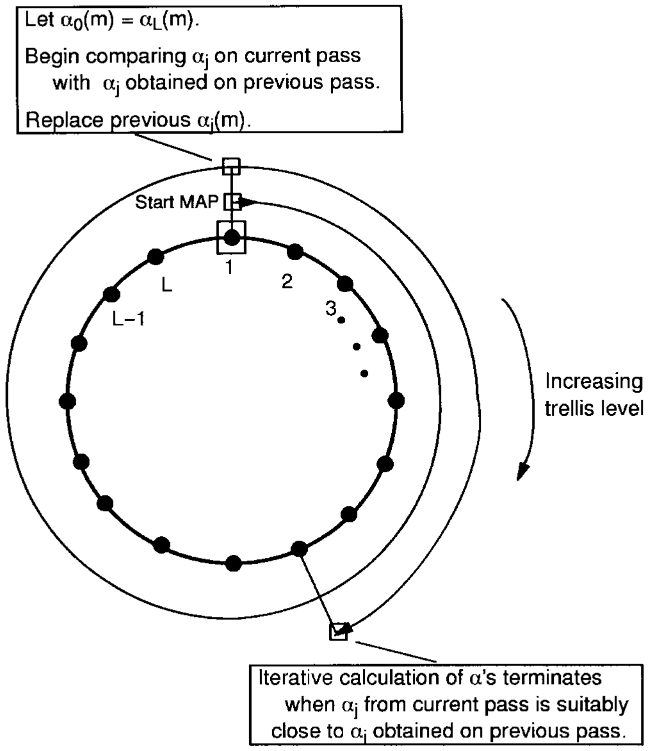

Let α0(m) = αL(m).
Begin comparing αj on current pass with αj obtained on previous pass.
Replace previous αj(m).
Start MAP
1
2
3
L
L-1
Increasing trellis level
Iterative calculation of α's terminates when αj from current pass is suitably close to αj obtained on previous pass.

Fig. 2. Time line for tailbiting BCJR decoder Algorithm A3. All alphas are renormalized to unit sum.

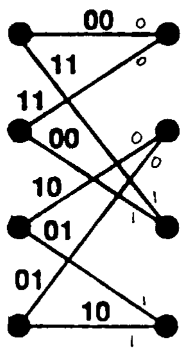
A more precise justification of this is as follows. Define $\boldsymbol{G} = K_o\Gamma_1 \cdots \Gamma_L$, where K_o is a normalization chosen to make the largest eigenvalue of G equal to one. K_o will scale the length of the largest eigenvector, but leave it otherwise unchanged. Start the recursion with some starting distribution α_o. After L iterations, we obtain

$$\alpha_L^\dagger = K_o\Gamma_1 \cdots \Gamma_L \alpha_o = \boldsymbol{G}\alpha_o$$

where $\dagger$ denotes that this α_L is scaled by K_o. The set of iterations produces $\alpha_1^\dagger, \cdots, \alpha_L^\dagger$. Another set of L iterations produces a second set $\alpha_{L+1}^\dagger, \cdots, \alpha_{2L}^\dagger$, and so on to the nth set. (Take $\Gamma_n = \Gamma_m$, with $m = n \bmod L$.) The sequence $\alpha_o, \alpha_L^\dagger, \alpha_{2L}^\dagger, \cdots, \alpha_{nL}^\dagger$ must converge to a limit, and after normalization, this limit is the unit sum α_o. Similarly, the set $\alpha_{nL}^\dagger, \cdots, \alpha_{nL-1}^\dagger$ after normalizing converges to $\alpha_1^o, \cdots, \alpha_L^o$ in Algorithm A2.

Some choices of the initial α_o will lead to faster convergence than others, and in the absence of better knowledge, $\alpha_o = (1/M, \cdots, 1/M)$ seems a good choice. The justification just given makes it seem that a number of L cycles might be required for convergence, but in reality, the forward recursion "forgets" an erroneous starting α_o very rapidly. Fig. 2 shows how the forward recursion wraps around the trellis a second time; the extra trellis search required before an α_{L+t} becomes close to a first round α_t is called the *wrap depth*, *W*. It is a fraction of L in most applications. The basic iterations in A3 are the same as in the original BCJR, so that A3's complexity is a factor $(W + L)/L$ greater than the standard BCJR.

By a similar argument, a recursion backward around the circle will converge to the desired set $\beta_1^o, \cdots, \beta_L^o$.



(a)

Stage	Alpha			
1	0.696	0.044	0.216	0.044
2	0.192	0.506	0.192	0.110
3	0.168	0.420	0.168	0.243
4	0.252	0.066	0.615	0.066
5	0.534	0.160	0.147	0.160

Stage	Beta′			
1	0.168	0.168	0.420	0.243
2	0.192	0.192	0.506	0.110
3	0.696	0.216	0.044	0.044
4	0.534	0.147	0.160	0.160
5	0.252	0.615	0.066	0.066

Stage	Lambda			
1	0.518	0.033	0.401	0.047
2	0.152	0.399	0.399	0.049
3	0.518	0.401	0.033	0.047
4	0.532	0.038	0.387	0.042
5	0.532	0.387	0.038	0.042

Stage	P{data=0}
1	0.551
2	0.551
3	0.920
4	0.571
5	0.920

(b)

Fig. 3. (a) Trellis section for the first example. (b) Output of Algorithm A1, after normalizing.

The new algorithm is as follows.

A3: Tailbiting BCJR Algorithm (Without Eigenvectors)

Given $y_1, \cdots, y_L$, take $\alpha_o = (1/M, \cdots, 1/M)$.

1) Find a set of renormalized row vectors $\alpha_1^\dagger, \cdots, \alpha_L^\dagger$ by the recursion

$$\begin{aligned} \alpha_t &= \alpha_{t-1}\Gamma_t \\ \alpha_t^\dagger &= \alpha_t \Big/ \sum_i \alpha_t(i), \qquad t = 1, \cdots, L. \end{aligned}$$

Continue this recursion to find $\alpha_t^\dagger, t = L+1, L+2, \cdots,$ taking $\Gamma_t = \Gamma_n$, where $n = t \bmod L$. When $\|\alpha_t^\dagger - \alpha_n^\dagger\|$ is sufficiently small by a suitable measure, stop. A complete set of α vectors is available as the last L found.

2) Execute a similar procedure backward around the trellis circle, to find the set $\beta_1^\dagger, \cdots, \beta_L^\dagger$.

The remainder of the algorithm is the same as A2, step 4).

IV. Examples

The first example features a BSC with crossover $p = 0.1$ and a four-state trellis comprising five sections, one of which is shown in Fig. 3 with the data symbols that attach to each transition. The sequence 00 10 10 00 00 is received. The Γ matrices are

$$\begin{aligned} \Gamma_1 &= \Gamma_4 = \Gamma_5 \\ &= \begin{bmatrix} 0.5(1-p)^2 & 0 & 0.5p^2 & 0 \\ 0.5p^2 & 0 & 0.5(1-p)^2 & 0 \\ 0 & 0.5p(1-p) & 0 & 0.5p(1-p) \\ 0 & 0.5p(1-p) & 0 & 0.5p(1-p) \end{bmatrix} \\ \Gamma_2 &= \Gamma_3 \\ &= \begin{bmatrix} 0.5p(1-p) & 0 & 0.5p(1-p) & 0 \\ 0.5p(1-p) & 0 & 0.5p(1-p) & 0 \\ 0 & 0.5(1-p)^2 & 0 & 0.5p^2 \\ 0 & 0.5p^2 & 0 & 0.5(1-p)^2 \end{bmatrix}. \end{aligned}$$

Step 1) of A1 yields that the largest eigenvalue of $\Gamma_1 \cdots \Gamma_5$ is $P\{Y_1^5\} = 5.39 \times 10^{-4}$ and

$$\alpha_o = (0.534 \quad 0.1596 \quad 0.1468 \quad 0.1596)$$

after normalizing. The outcome of steps 2)–4) is summarized in Fig. 3. The σ can be found as well, and the probabilities of data 1 or 0 in the transmission; the last appear in the figure.

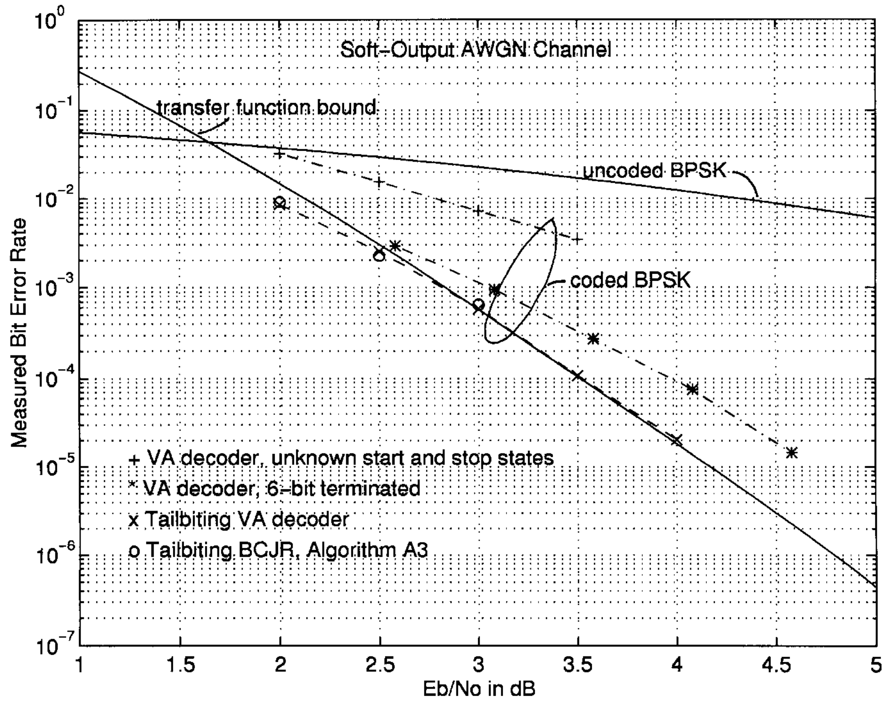


Fig. 4. Bit-error rates for four decoders working with the $R = 1/2$ memory 6 code (554, 744). $L = 48$ stages; tailbiting decoders use a 40-stage wrap depth. The tailbiting BCJR decoder puts out the most probable bit at each stage. Theoretical bit-error estimates with and without coding are shown; coding estimate is based on the first six terms of the transfer function BER bound.

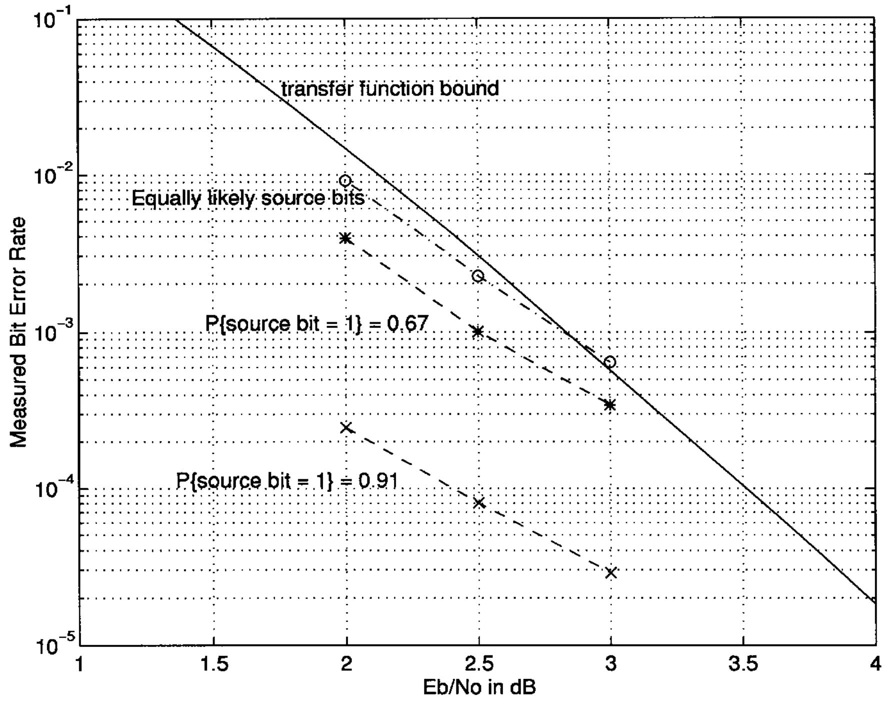


Fig. 5. Bit-error rates versus E_b/N_o for the tailbiting BCJR decoder when the source bit statistics are biased. The decoder puts out the data bit with the highest probability at each stage.

In this example, a five-stage code word 00000 was probably sent. The example can be repeated with a longer trellis or even with a very short one as long as it describes a valid tailbiting trellis code. Valid means that the encoder Markov chain is irreducible, and that any entire matrix sequence $\Gamma_1 \cdots \Gamma_L$ starts and ends with the same states.

In example 2, we turn to the antipodal-signaling AWGN channel and the best free-distance 64-state convolutional code with rate 1/2. The tailbiting trellis has 48 stages, and the wrap depth is 40 extra stages. Fig. 4 compares an unterminated and terminated standard Viterbi decoder (note that the terminated VA curve is moved to the right by the energy loss to the termination bits); also shown are a tailbiting VA decoder and the tailbiting BCJR decoder of Algorithm A3. It is clear that the two tailbiting decoders are superior for these short block lengths.

Fig. 4 assumes that the data bits are equiprobable. When they are not, the tailbiting BCJR decoder gains a significant advantage over the tailbiting VA. The Γ matrices are now modified ($\Pr\{i \to j\}$ in (10) is changed from 0.5 to the probability of the data bit causing the respective transition). Fig. 5 compares the 0.5/0.5 data distribution case to the skewed distributions 0.67/0.33 and 0.91/0.09.

V. Conclusions

We have demonstrated several algorithms that apply the idea of MAP decoding to tailbiting coding systems. Algorithm A3 is perhaps the most practical since, in our experience, it is the usual BCJR scheme with a moderately extended forward recursion. The algorithms in this paper apply especially when trellis codes are short, so that their rates are reduced too much by known-bit termination, or whenever *a priori* source probabilities are skewed. We foresee applications to turbo decoding with relatively short blocks and to source-controlled channel decoding.

References

[1] L. R. Bahl *et al.*, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[2] J. Hagenauer, "Source-controlled channel decoding," *IEEE Trans. Commun.*, vol. 43, pp. 2449–2457, Sept. 1995.

[3] A. S. Deif, *Advanced Matrix Theory for Scientists and Engineers*, 2nd ed. New York: Gordon and Breach, 1991.

[4] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. 44, pp. 1261–1271, Oct. 1996.

[5] G. Battail *et al.*, "Replication decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 332–345, May 1979.

[6] H. H. Ma and J. K. Wolf, "On tail biting convolutional codes," *IEEE Trans. Commun.*, vol. COM-34, pp. 104–111, Feb. 1986.

John B. Anderson (M'72–SM'82–F'87), for a photograph and biography, see this issue, p. 195.

Stephen M. Hladik (S'83–M'89) received the B.S. and M.S. degrees in electrical engineering from Union College, Schenectady, NY, in 1985 and the Ph.D. degree in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1989.

From 1985 to 1986, he worked at Eastman Kodak's U.S. Apparatus Division in Rochester, NY, where he was involved in the development of an electronic still image camera and player/recorder. Since September 1989, he has been with General Electric's Corporate Research and Development Center, Schenectady, NY, working on digital communication techniques and signal processing for modems. His work has included coding/decoding techniques for low signal-to-noise channels, demodulation and synchronization algorithms, antenna diversity combining, and the design, modeling, simulation, and analysis of communications systems. His research has been applied to voice and data communication systems, including satellite communications systems and equipment, digital cellular radiotelephone, digital land mobile radio, and an air-to-ground communication system for general aviation. He holds six U.S. patents with several pending.

Dr. Hladik is a member of the IEEE Communications, Vehicular Technology, and Information Theory Societies, and the Tau Beta Pi, Eta Kappa Nu, and Sigma Xi.