Concatenated Decoding with a Reduced-Search BCJR Algorithm

Volker Franz and John B. Anderson, *Fellow, IEEE*

***Abstract*—We apply two reduced-computation variants of the BCJR algorithm to the decoding of serial and parallel concatenated convolutional codes. One computes its recursions at only M states per trellis stage; one computes only at states with values above a threshold. The threshold scheme is much more efficient, and it greatly reduces the computation of the BCJR algorithm. By computing only when the channel demands it, the threshold scheme reduces the turbo decoder computation to one–four nodes per trellis stage after the second iteration.**

***Index Terms*—Complexity reduction, concatenated coding, decoders, error correction.**

I. Introduction

THERE has been great interest in recent years in coding systems that employ various kinds of code concatenation. These can be conveniently divided into serial and parallel systems, as shown in Fig. 1. In serial concatenated coding, an outer encoder feeds its output to a second, inner encoder, and the decoding is performed by corresponding inner and outer decoders. One recent paper on this subject is [7]. In parallel concatenated coding, the same data are encoded twice separately, and the encodings are sent and decoded in parallel. An important parallel method called turbo decoding was devised by Berrou *et al.* in 1993 [1]; we will describe it in detail in Section IV. In both serial and parallel coding, an important element of the decoder is the MAP (maximum *a posteriori* probability) decoder, a device that puts out the probability of individual trellis states or data symbols, rather than simply the most likely state or symbol, or information about an entire state sequence. Probabilities are needed because the two subdecoders in a serial or parallel decoder system need to exchange soft information in order to achieve good error performance. These flows are shown in Fig. 1 by dashed lines. For trellis encoding with Markov data, the implementation of the MAP decoder becomes a special scheme, the BCJR algorithm, named after the authors of [2].

Unfortunately, the BCJR is computationally intensive, and the purpose of this paper is to present a strong simplification of it that does not sacrifice the decoder error performance. The main simplification of BCJR heretofore has been the SOVA (soft-output Viterbi algorithm) of Hoeher and Hagenauer [3]. The SOVA associates a simplified estimate of the data symbol probability with each trellis state, and revises the estimate at each trellis stage. Later modifications of SOVA by Robertson and coworkers, such as [5] and [6], have attempted to simplify the scheme still further. While the basic SOVA does run significantly faster than the BCJR, it also fails to perform as well. The schemes that we present will exploit the fact that most probabilities in the working of the BCJR algorithm are very small. We look at ways to ignore these small probabilities and their related computation, without losing error performance. We find that the most successful strategy is simply to truncate small probabilities to zero, thus killing the computation that stems from them. The modified BCJR keeps track of and works with what information remains alive. When the channel noise is low, most probabilities are small, and little computation is performed; when the channel is poor, many trellis paths are probable, and the BCJR expands to its full computation.

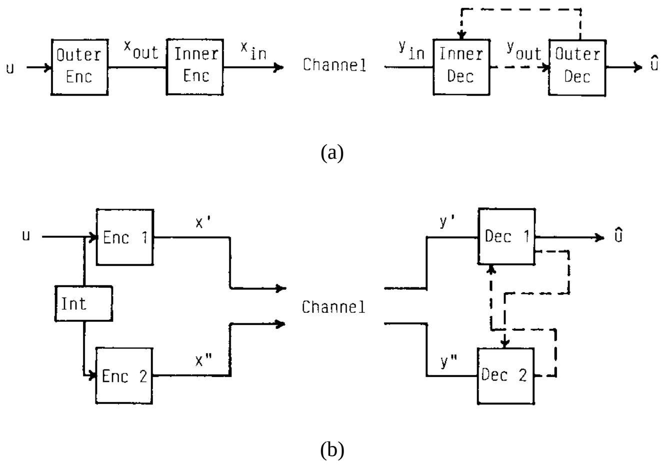


Fig. 1. (a) Serial and (b) parallel concatenation.

The BCJR algorithm works with vectors whose components are values at the trellis states and which evolve stage by stage. Section II will review the basic BCJR recursions that produce these vectors, and explain several strategies for truncation of vector components. One strategy keeps a fixed small number M of the vector components alive; the other keeps all those alive that lie above a certain threshold. Section III applies these two strategies to a rate-1/3 serial convolutional coding system. Section IV applies them to a rate-1/3 parallel "turbo" coding system. In both kinds of concatenation, the threshold method works best.

II. The Basic Algorithms

Before presenting test results, we review the algorithms that will be used to produce them.

Manuscript received September 12, 1996; revised April 29, 1997 and September 18, 1997.

V. Franz is with Siemens AG, Munich, Germany.

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

Publisher Item Identifier S 0733-8716(98)00226-1.

The BCJR algorithm performs two basic recursions on the stream of received channel outputs $Y_1 \cdots Y_L$. From the results of these recursions, it computes the probabilities of the encoded data and of the various transitions in the code trellis. It is useful in what follows to keep in mind the code trellis and to associate the computed values with nodes, stages, and transitions in the trellis. It is also convenient if we adopt matrix terminology.

The BCJR algorithm seeks the probabilities

$$\Pr\{S_t = i | Y_1 \cdots Y_L\} \tag{1}$$

the probability that the encoder was in state i at time t, given the frame of observed channel outputs. (Note that in this paper, the code will be a rate-1/2 convolutional one, and each Y_t is a pair of AWGN channel outputs.) Actually, the algorithm finds, instead, the probabilities

$$\boldsymbol{\lambda}_t(i) = \Pr\{S_t = i; Y_1 \cdots Y_L\}, \qquad t = 1, \cdots, L \tag{2}$$

which are those in (1) scaled by a constant, $\Pr\{Y_1 \cdots Y_L\}$. Next, define the matrix $\boldsymbol{\Gamma}_t$ by

$$\boldsymbol{\Gamma}_t(i,j) = \Pr\{S_t = j; Y_t | S_{t-1} = i\}, \qquad t = 1, \cdots, L. \tag{3}$$

There is one Γ_t for each trellis stage. In our case, it amounts to the probability that the transition i to j and the output Y_t occurs, given that the state at stage $t-1$ was i; any *a priori* probabilities on the data (and hence on the encoder transitions) enter the algorithm here. Finally, define the row vector

$$\boldsymbol{\alpha}_t(i) = \Pr\{S_t = i; Y_1 \cdots Y_t\}, \qquad t = 1, \cdots, L \tag{4}$$

and the column vector

$$\boldsymbol{\beta}_t(j) = \Pr\{Y_{t+1} \cdots Y_L | S_t = j\}, \qquad t = 1, \cdots, L-1. \tag{5}$$

The steps of the BCJR algorithm are then as follows.

1) Form the set of row vectors $\{\boldsymbol{\alpha}_t, t = 1, \cdots, L\}$, one for each trellis stage, by the *forward recursion*

$$\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1}\boldsymbol{\Gamma}_t, \qquad t = 1, \cdots, L. \tag{6}$$

2) Form the set of column vectors $\{\boldsymbol{\beta}_t, t = 1, \cdots, L-1\}$, one for each stage, by the *backward recursion*

$$\boldsymbol{\beta}_t = \boldsymbol{\Gamma}_{t+1}\boldsymbol{\beta}_{t+1}, \qquad t = L-1, \cdots, 1. \tag{7}$$

3) Form the set of row vectors $\{\boldsymbol{\lambda}_t, t = 1, \cdots, L\}$ in (2) by

$$\lambda_t(i) = \alpha_t(i)\beta_t(i), \qquad \text{all } i, t = 1, \cdots, L. \tag{8}$$

If the encoder starts in state 0, the forward recursion is initialized by the vector $\boldsymbol{\alpha}_o = (1, 0, \cdots, 0)$. If the encoder terminates in a known state q, the backward recursion begins from $\boldsymbol{\beta}_L = (0, \cdots, 0, 1, 0, \cdots, 0)$, where the 1 is in the qth position.

By summing the components of any $\boldsymbol{\lambda}_t, \Pr\{Y_1, \cdots Y_L\}$ can be obtained; normalizing (2) by this factor gives $\Pr\{S_t = i\}$, the probability that the encoder entered state i at time t. More generally, it can be shown that any complete $\boldsymbol{\alpha}_t$ or $\boldsymbol{\beta}_t$ can be scaled by any factor, and $\boldsymbol{\lambda}_t$ in (8) still yields the same $\Pr\{S_t = i\}$ when $\boldsymbol{\lambda}_t$ is normalized to unit sum. An important point in what follows is that we normalize each $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to unit sum as they are found. This is the key to the second of the reduced algorithms, and it is a practical requirement in any case because successive outcomes of the forward and backward recursions will otherwise diminish to insignificance.

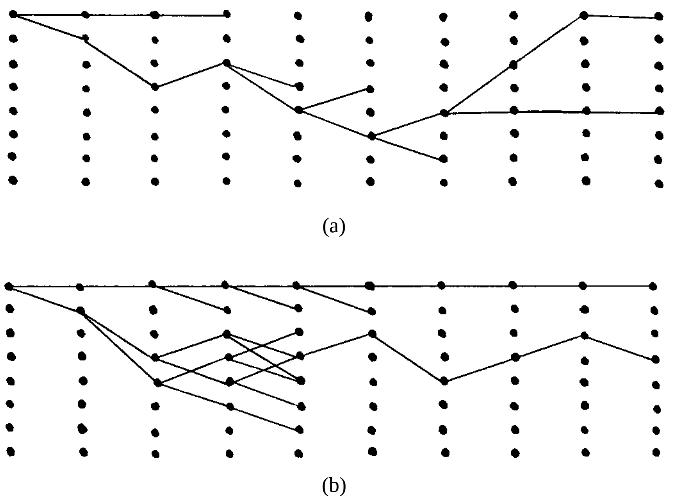


Fig. 2. Computation patterns in a reduced BCJR algorithm working in an eight-state trellis. A line connecting two nodes indicates that the left node value was used to compute the right node value, and that the right value survived the reduction process. (a) "M" computation pattern, keeping best two nodes at each stage. (b) "T" pattern, keeping nodes with values above a threshold.

The probability that a trellis transition occurred from $S_{t-1} = i$ to $S_t = j$ can be computed from

$$\Pr\{S_{t-1} = i; S_t = j; Y_1 \cdots Y_L\} = \alpha_{t-1}(i)\Gamma_t(i,j)\beta_t(j). \tag{9}$$

A similar comment about normalization applies to (9). The probability $\Pr\{u_t = 0\}$, the probability that data symbol u was 0, can be obtained by summing all of the $\lambda_t(i)$ that correspond to data 0 (feedforward codes) or all of the i, j in (9) that correspond to data 0 (for recursive codes). With binary data, $\Pr\{u_t = 1\} = 1 - \Pr\{u_t = 0\}$.

The SOVA replaces all of this calculation by a series of upgraded estimates of the probability $\Pr\{u_t = 0\}$. The scheme runs the Viterbi algorithm, except that at each stage, the spread of metrics entering a given node is noted. From this and from the estimate at stage $t-1$, an estimate of $\Pr\{u_t = 0\}$ is formed.

We will explore two reduced-computation versions of the BCJR algorithm. One is based on the M algorithm, a reduced trellis decoder that searches breadth-first through a trellis, retaining the best M-state paths at each level. In an M-BCJR algorithm, the forward recursion on α_{t-1} that produces α_t is performed using only the M largest components of α_{t-1}; the rest are declared dead and set to 0. The same scheme can be applied with the backward recursion, but because (8) and (9) are products of α and β components, it is simpler just to execute the backward recursion on the region of the trellis where the forward elements are alive. Fig. 2 shows an idealized M-BCJR computation pattern with $M = 2$.

In the second reduced scheme, components are set to zero whenever they fall below a threshold. At each stage, the α_t found is normalized to unit length. In finding α_t by (4), only the components of α_{t-1} are used that exceed the threshold;

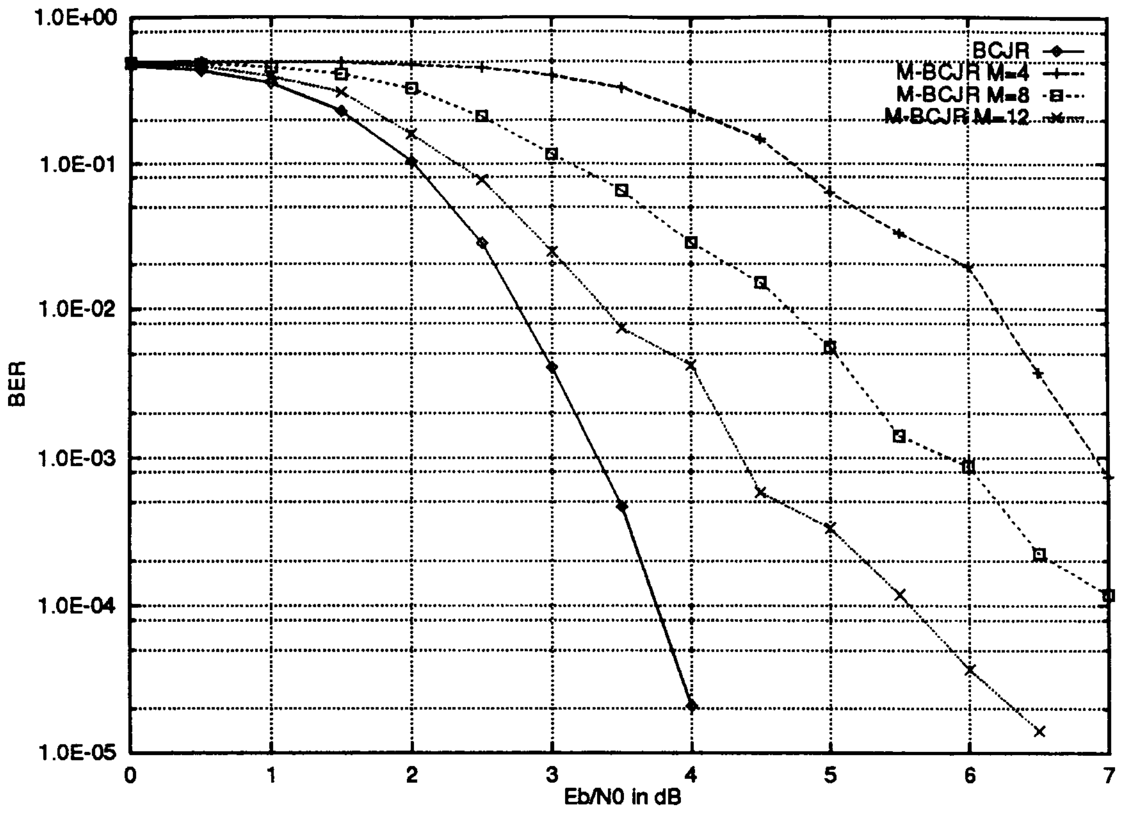


Fig. 3. BER performance against E_b/N_0 with serial concatenated rate-1/3 coding. BCJR inner decode is compared to M-BCJR with $M = 4, 8, 12$. 16-state encoders. VA outer decoder. E_b is energy per data bit.

the rest are set to zero. This is the T-BCJR scheme. As with M-BCJR, the backward recursion need only trace through regions where the trellis is alive after the forward recursion. Fig. 2 shows a sample computation pattern; unlike the M-BCJR pattern, the computation can vary widely. This can present difficulty in a hardware implementation with a maximum cycle rate, but is less of a problem in a software version, where it implies only that a certain size data buffer is available.

In both schemes, some method is needed to keep track of which computation nodes remain alive. If the set of whole $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ vectors are kept intact, then the 0 entries mark the killed nodes. Otherwise, a system of pointers to live nodes can be kept; this will save storage. In any case, almost all of the effort in the BCJR is expended in the real-number updating of the $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ set, and this effort is saved at the killed nodes.

III. Reduced Computation with Serial Concatenation

We report first on the testing of a rate-1/3 serial concatenated convolutional coding system that employs reduced computation. The outer encoder is a rate-2/3 encoder, obtained by puncturing various feedforward rate-1/2 encoders. Every second bit produced by the second set of taps is punctured, which results in three codeword bits for every two data bits. The inner encoder is a feedforward rate-1/2 encoder. The transmission frame is 212 data bits. Between the two encoders comes a random interleaver; its random pattern is changed at each frame, so as to provide a measure of the average performance of random interleaving. The outer encoder in every test is a Viterbi decoder. The best error performance in this system with code memories in the middle range (4–6) occurs when the inner decoder is the BCJR algorithm and the outer VA makes use of its soft output. When the inner decoder is a SOVA, performance is about 0.2 dB worse, and when it is just a hard-output VA, performance is perhaps 1 dB worse. However, our interest now is in a reduced BCJR.

Fig. 3 shows the bit-error rate (BER) performance against E_b/N_0 for several M, with an M-BCJR as the inner (first) decoder, a VA outer decoder, and memory-4 codes. The M-BCJR performs steadily better as the number of retained state paths grows, and apparently performs at the full BCJR level only when M reaches 16, the number of states in the full trellis. At least for small trellises, it seems that the M approach is not fruitful for the serial coding system. We will comment further on this later.

Here and throughout the paper, sufficiently many frames are transmitted during tests so that at least ten erroneous frames are received. This leads to an uncertainty that can be judged from the figures. This many frames are needed even for the BER measurements since the mechanism of decoder error is frame failure, not individual symbol failure.

A reduced-computation BCJR based on deletion of paths below a threshold seems to be much more successful. Fig. 4 shows the BER for the same system as Fig. 3, but with a T-BCJR having the thresholds shown (i.e., inner T-BCJR, outer VA, memory-4 codes). With threshold 0.001, the BER performance virtually equals the full BCJR. The average number of states that fall above the threshold is shown in Fig. 5, plotted against E_b/N_0, for several thresholds. At

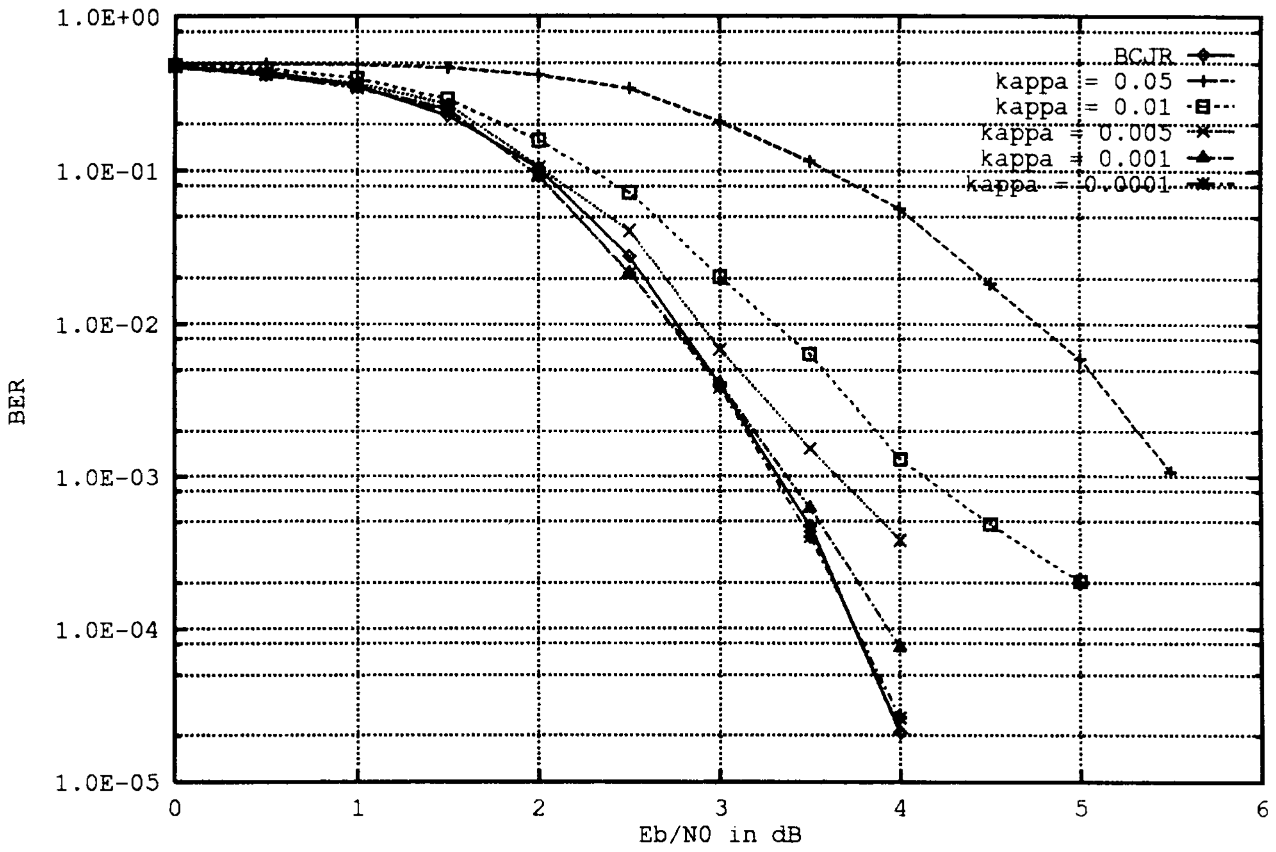


Fig. 4. BER against E_b/N_0, serial concatenation. BCJR inner decoder is compared to T-BCJR with threshold kappa = 0.0001–0.05.

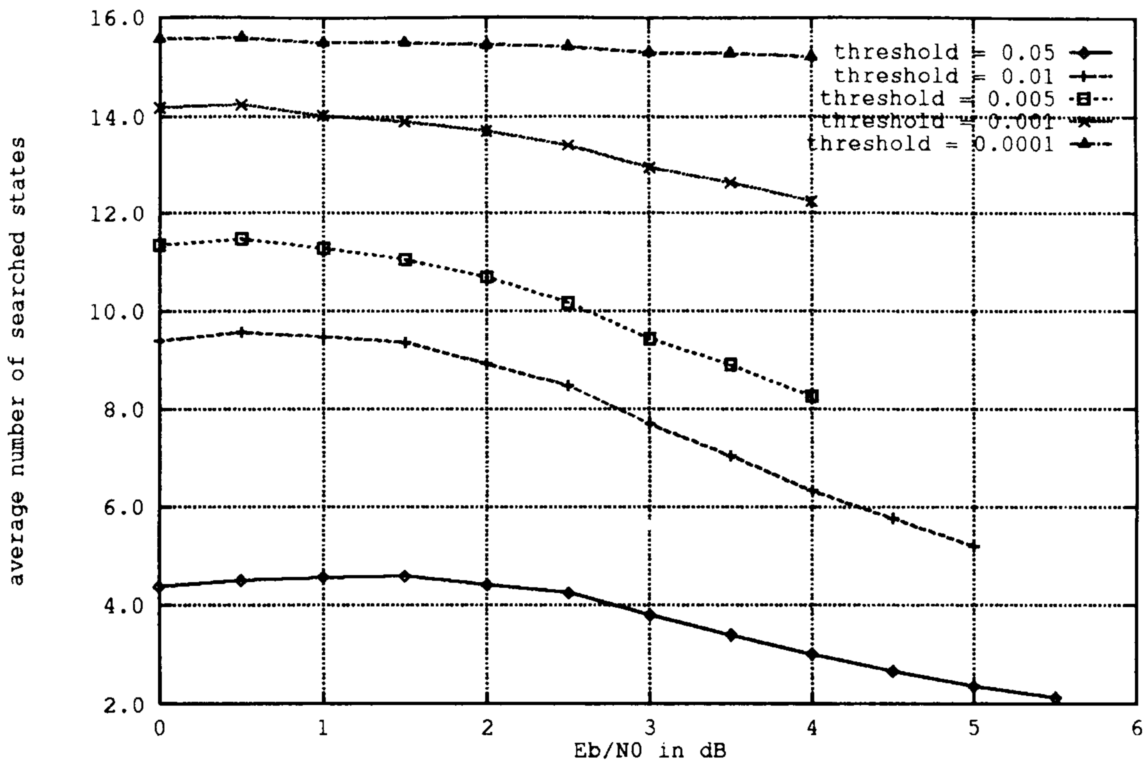
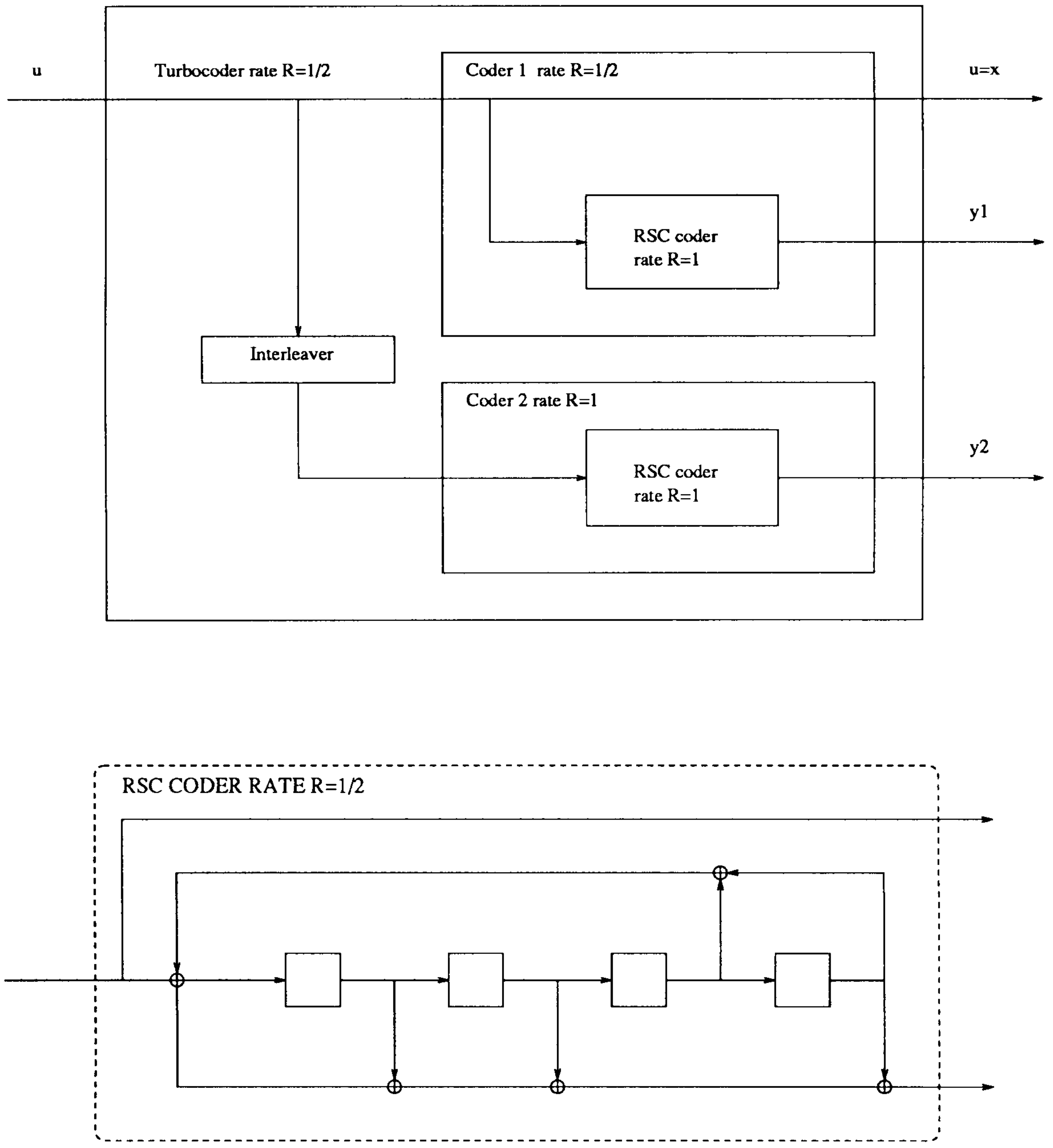


Fig. 5. Serial concatenation and the T-BCJR. Average number of live states per trellis stage against E_b/N_0 for several thresholds.



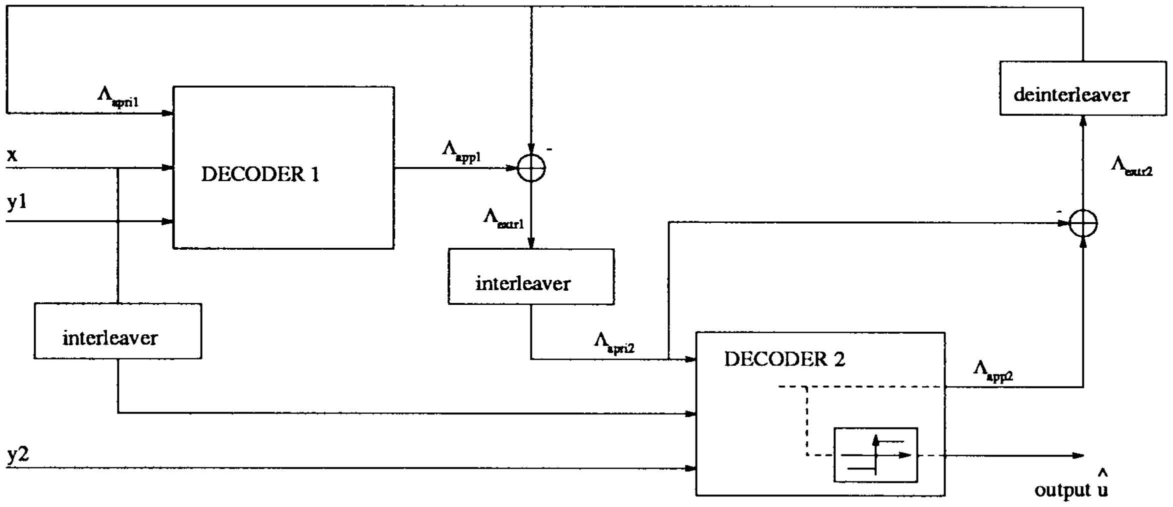
(a)

Fig. 6. (a) (Above) "Turbo" rate-1/3 parallel concatenated encoder. (Below) Rate-1/2 recursive systematic encoder with memory 4.

threshold 0.001, the T-BCJR reduces the live states in the recursions from 16 to the range 6–9, depending on the value of E_b/N_0.

IV. Reduced Computation with Parallel Concatenation

We next report on tests of a rate-1/3 parallel concatenated system, specifically, a "turbo" convolutional decoder. Since there are many variations on the turbo theme, we begin with a description of the particular one used here. As shown in Fig. 6(a), the frame of data $\{u_t\}$ at the encoder feeds a first rate-1/2 recursive systematic coder (RSC), and after an interleaving, the same data feed an identical second encoder; the parity bit output of both encoders is transmitted, while the data bit output of Encoder 2 is ignored. A sample RSC is shown in the figure. It is defined by its taps taken as octals, with a leading tap always present and the feedback set written first; the RSC in the figure is a (46, 72) RSC. Fig. 6(b) shows the iterative decoder that we use. The first iteration works as follows. Decoder 1 uses the received channel sequences x_t and $y_{t,1}$, and $\Lambda_{\text{apri1}}(u_t)$, the log-likelihood ratio derived from any *a priori* probabilities about those data that are known (if nothing is known, the ratio is 0). The outputs of Decoder 1 are the *a posteriori* log-likelihood ratios of $\{u_t\}$, denoted $\Lambda_{\text{app1}}(u_t)$. What is passed to Decoder 2 is not this ratio, but is rather the so-called *extrinsic information*, expressed by the



(b)

Fig. 6. *(Continued.)* (b) Turbo decoder structure.

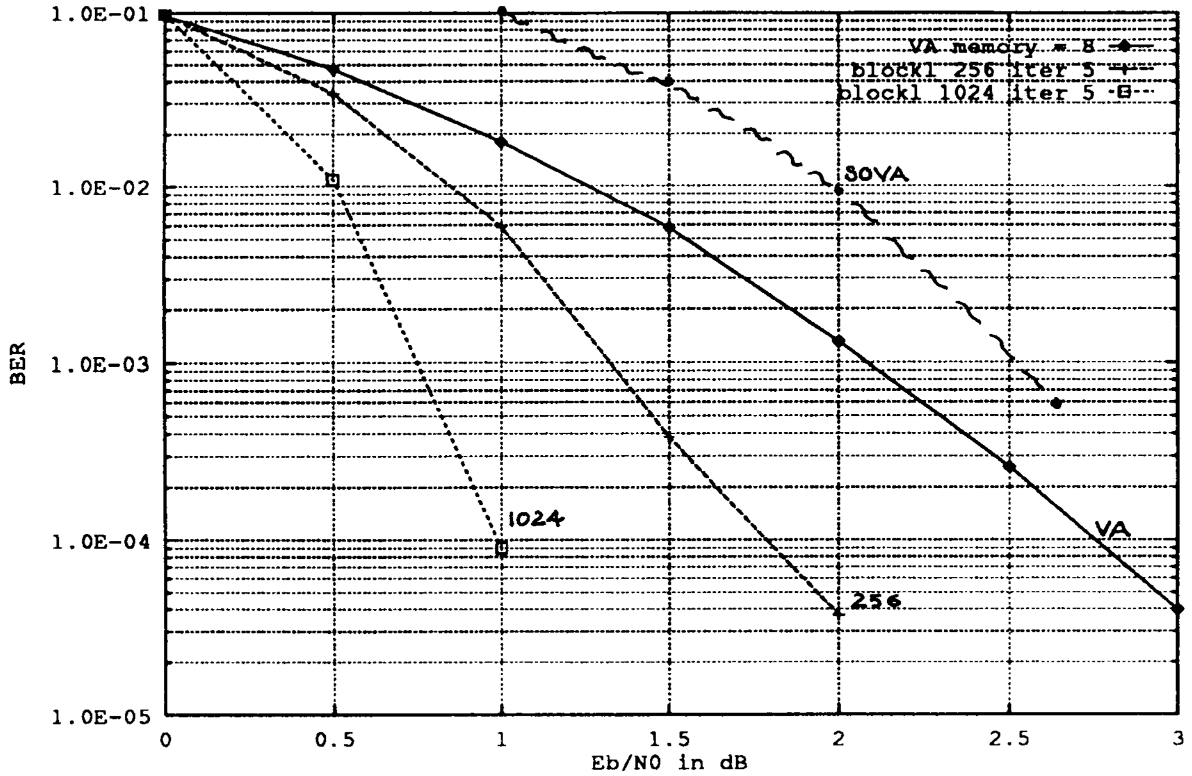


Fig. 7. Turbo decoding BER against E_b/N_0 for the basic BCJR decoder; frame size 256 and 1024, five iterations, and $m = 4$ encoders. Comparison is made to SOVA turbo decoder and to $m = 8$ encoder with ordinary VA decoding. E_b is total energy per data bit.

log-likelihood ratio

$$\Lambda_{\text{extr1}}(u_t) = \Lambda_{\text{app1}}(u_t) - \Lambda_{\text{apri1}}(u_t). \tag{10}$$

Decoder 2 accepts $\Lambda_{\text{extr1}}(u_t)$ as its *a priori* information about u_t, denoted in the figure as Λ_{apri2}, and from this together with the streams $\{x_t\}$ and $\{y_{t,2}\}$, it calculates a new *a posteriori* ratio for u_t, denoted $\Lambda_{\text{app2}}(u_t)$. Technically, $\Lambda_{\text{app1}}(u_t)$ is the *a priori* ratio at the input to Decoder 2, but investigations by us and others have shown that this input gives too much emphasis to the original *a priori* ratio Λ_{apri1}, and so Λ_{extr1} is used instead.

The calculation of Λ_{app2} by MAP Decoder 2 completes one iteration of the turbo decoder, and the output data symbols are available as the sign of the ratio. If further iterations are desired, an *a priori* input for Decoder 1 is formed from

$$\Lambda_{\text{extr2}}(u_t) = \Lambda_{\text{app2}}(u_t) - \Lambda_{\text{apri2}}(u_t) \tag{11}$$

and fed to Decoder 1. Various interleavers and deinterleavers are needed as shown in the figure.

As a performance benchmark for this decoder, we give Fig. 7, which plots the decoder BER against E_b/N_0 for frame lengths 256 and 1024 data bits. The encoders are the $m = 4$ one in Fig. 6(a). Each frame transmission is terminated by m

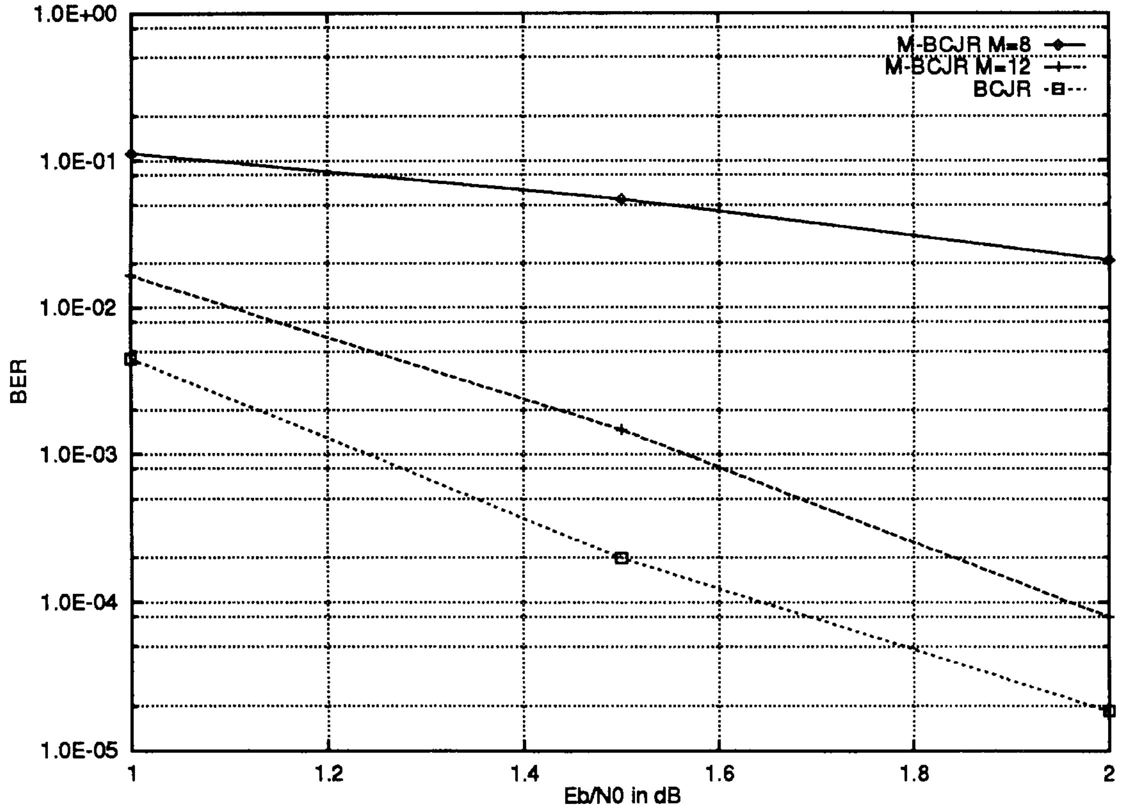


Fig. 8. Turbo decoding and the M-BCJR. BER against E_b/N_0 for the BCJR and the M-BCJR with $M = 8, 12$. Frame size 256 and $m = 4$ encoders.

bits, and the decoders pass through five iterations. The figure also shows some BER's observed for a SOVA implementation of the decoder, with frame size 256. SOVA is about 1 dB worse that the full BCJR implementation. For comparison to these, we have plotted the BER observed with simple VA decoding of the $(16)^2$-state rate-1/3 code with best free distance. It performs 2 dB worse at low BER, compared to the 1024-frame turbo system. Further details are available in [4].

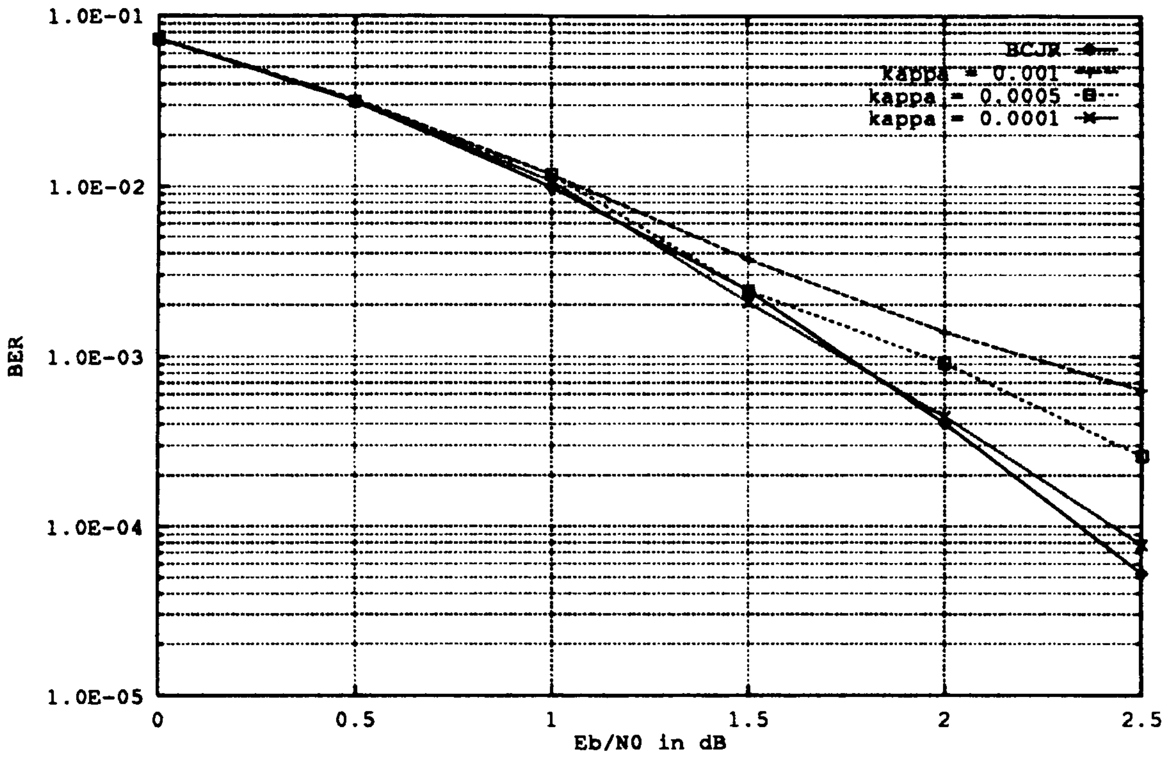
Fig. 8 shows the BER against E_b/N_0 when the BCJR in Decoders 1 and 2 is replaced by an M-BCJR. The frame size is 256, and the decoder works through five iterations. As in the serial decoder, the performance improves when more state paths are kept, but seems to reach the full BCJR performance only when the full trellis of paths is computed.

Fig. 9(a) (BER) and (b) (frame-error rate) shows the performance when the threshold BCJR is used instead. There are again five iterations, and as always, the decoder ran until at least ten frame errors occurred. The encoders in this test series were a (75, 62) RSC, with free distance 6. It appears that threshold 0.0001 gives near-optimal performance, and that 0.0005 gives quite good performance. But the real story is told by Fig. 10, which plots the average number of live states per iteration for several thresholds as a function of E_b/N_0, and by Fig. 11, which shows how this average declines with the iteration number at threshold 0.0005. At any reasonable E_b/N_0, the T-BCJR yields a major reduction in the recursion calculation already on the first decoder iteration. After the initial one or two iterations, the T-BCJR reduces the computation to the extension of virtually a single path, when the channel E_b/N_0 exceeds 1.5 dB. This means that later decoder iterations contribute little cost compared to the initial iteration.

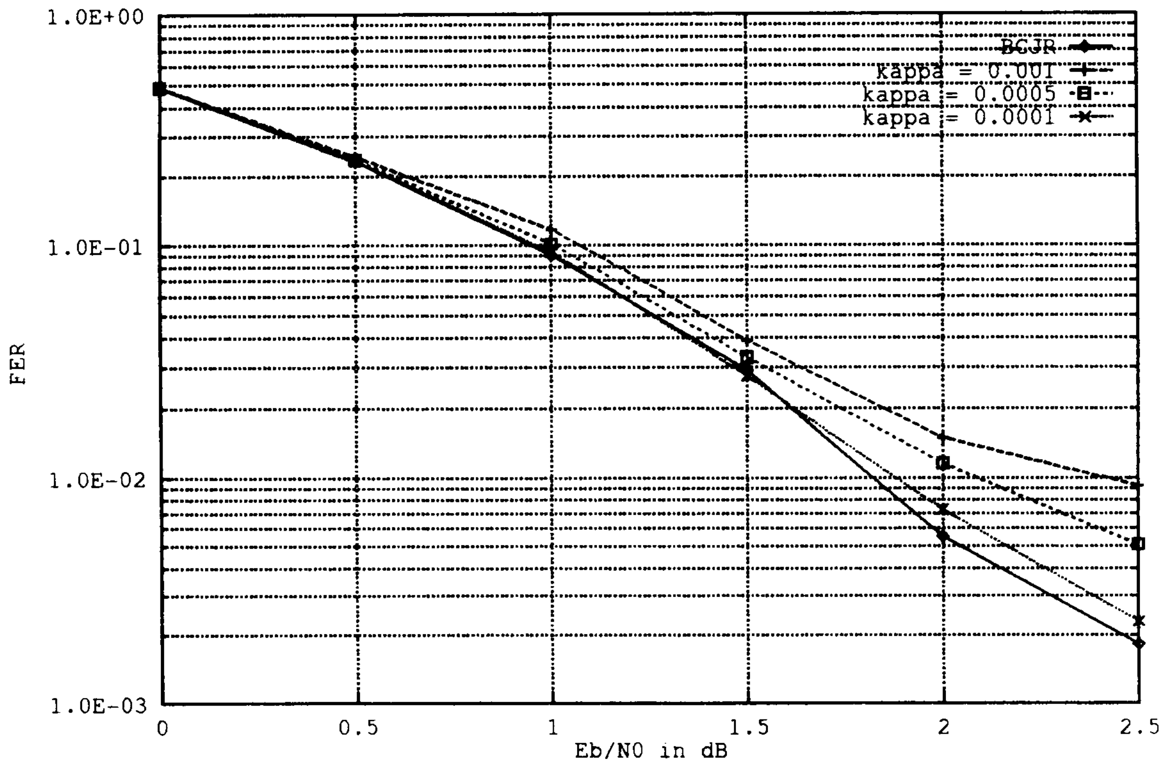
V. Conclusions

We have developed several ways to reduce the trellis calculation of the BCJR algorithm, and have tested them with serial and parallel concatenated coding schemes at rate 1/3. By far, the most successful strategy of reduction is to suppress calculation after trellis nodes whose values fall below a threshold. There is apparently a huge variation in the values of normalized α and β during the BCJR recursions, and the dropping of values that fall below a certain reasonable limit seems to have virtually no effect on the utility of succeeding values. We suspect that there is wide scope for further and more subtle extensions of the threshold idea.

The reduction of the recursions to a constant M computation paths through the trellis was not successful. An explanation of this is possible. We have observed that erroneous decoding is caused by disturbances that cause many values of α or β at a few levels to have roughly similar values, instead of the usual wide spread of values. To deal with this situation requires all of the resources of the BCJR procedure, and when this is restricted *a priori* in the manner of the M-BCJR, certain noise bursts cannot be cleaned up. The T-BCJR, on the other hand, is guided by the size of the α and β values, and when there are many values of significant size, the T-BCJR expands the computation to include all of them. During the decoder iterations, the two parts of the turbo



(a)



(b)

Fig. 9. (a) Turbo decoding and the T-BCJR. Bit-error rate against E_b/N_0 for several thresholds. Five iterations. (b) Frame-error rate for the case of (a). Five iterations.

decoder trade information, and it may happen that one part finds that some state transition in the other has very low probability. The consequence is that some α and β values in the other forthwith fall below threshold, and no further computation occurs out of this transition. This kind of behavior probably causes the reduction in T-BCJR computation to a single path that occurs in later iterations.

This paper and most previous work with turbo codes have been restricted to codes with small state space, generally 16 or less. The reduced algorithms here make practical for the

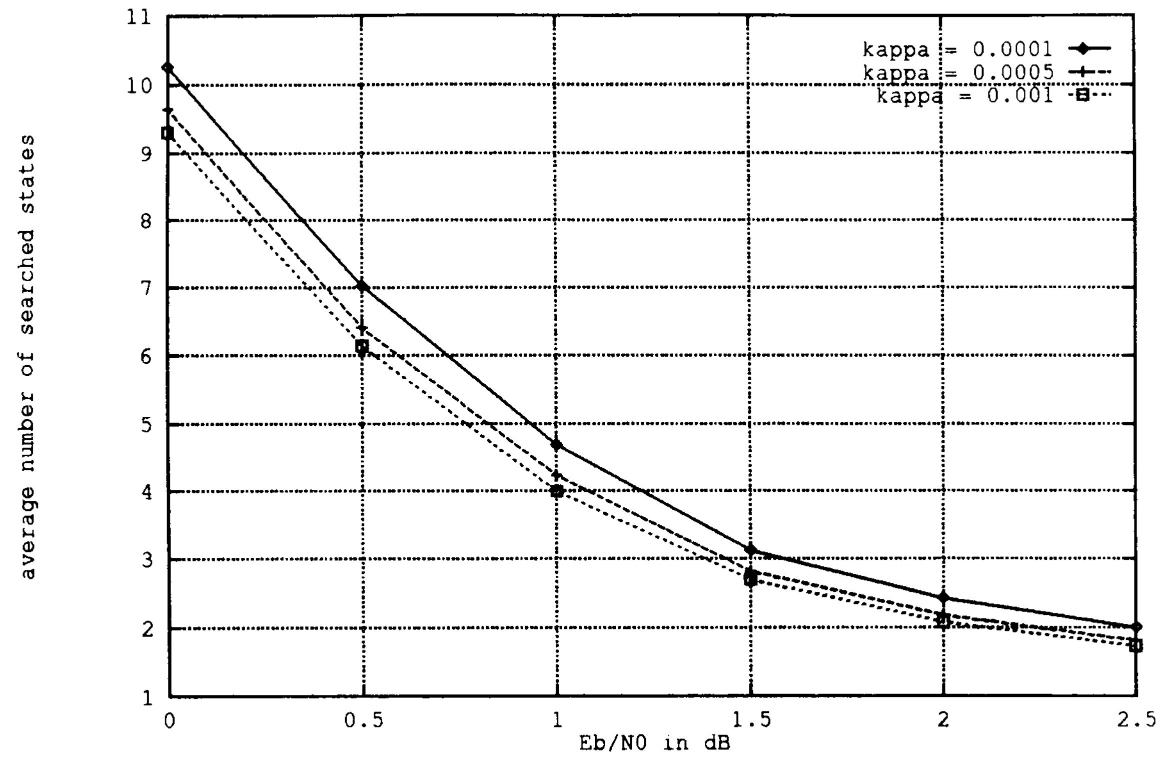


Fig. 10. Turbo decoding and the T-BCJR. Average number of live states per iteration against E_b/N_0 for several thresholds. System of Fig. 9. Eight iterations.

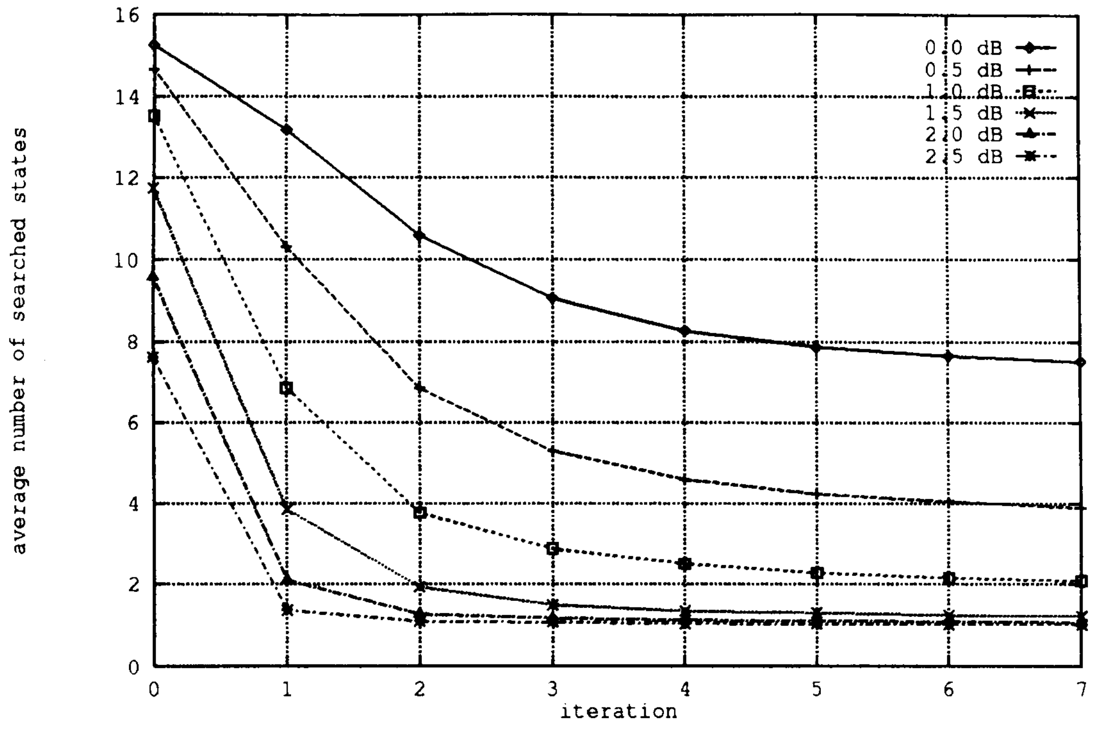


Fig. 11. Turbo decoding and the T-BCJR. Average number of live states observed during each iteration, with threshold 0.0005 and E_b/N_0 = 0–2.5 dB. System of Fig. 9. Eight iterations.

first time the use of codes with many states. We have, in fact, tested encoders with as many as 64 states, and their overall computation average was not very different from the 16 state results here. This is because the M-BCJR remains M always, and the T-BCJR only expands toward 64 states during very unlikely noise bursts.

It may be that the M-BCJR is relatively more attractive with large codes than it is with small ones; historically, this was the case with the M algorithm and ordinary convolutional decoding. However, we suspect and conjecture that large codes are useful only at higher E_b/N_0; small component codes are big enough at rate 1/3 in the range 0–2 dB. Above this range, the T-BCJR is unbeatable because the average computation drops to a single path after the first iteration. Thus, we suspect that the T-BCJR is generally better than the M-BCJR.

References

[1] C. Berrou *et al.*, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *Proc. IEEE Int. Conf. Commun.*, Geneva, Switzerland, May 1993, pp. 1064–1070.

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

[3] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft decision outputs and its applications," in *Proc. IEEE GLOBECOM*, Dallas, TX, Nov. 1989, pp. 47.1.1–47.1.6.

[4] V. Franz, "The soft-output M-algorithm," Dipl. Ing. thesis, Dep. Commun. Eng., Tech. Univ. Munich, Munich, Germany, July 1996.

[5] P. Robertson *et al.*, "A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain," in *Conf. Rec., IEEE Int. Conf. Commun.*, June 1995, pp. 1009–1013.

[6] L. Papke and P. Robertson, "Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme," in *Conf. Rec., IEEE Int. Conf. Commun.*, June 1996, pp. 102–106.

[7] S. Benedetto *et al.*, "Serial concatenation of interleaved codes: Performance analysis, design and iterative decoding," *IEEE Trans. Inform. Theory*, accepted for publication.

Volker Franz was born in Munich, Germany, in 1972. He received the Dipl.-Ing. degree in electrical engineering from the Technical University of Munich in 1996.

In 1996, he was a Guest Researcher at Rensselaer Polytechnic Institute, Troy, NY, where he performed research on turbo channel decoding. In August 1996, he joined the Advanced Development Mobile Networks Group at Siemens AG, Munich. He is also a doctoral candidate part time with the Department Communication Engineering, Technical University of Munich. His current research interests are adaptive equalization and channel coding for mobile networks.

John B. Anderson (M'72–SM'82–F'87) was born in New York in 1945. He received the B.S., M.S., and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1967, 1969, and 1972, respectively.

From 1972 to 1980, he was a faculty member of the Department of Electrical and Computer Engineering, McMaster University, Hamilton, Canada. He was a founding member there of the Communications Research Laboratory, the leading university laboratory in that field in Canada. Since 1981, he has been with the Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, where he is a Professor. He has held Visiting Professorships at the University of California (Berkeley), Chalmers and Lund University in Sweden, Queen's University, Canada, Deutsche Luft- und Raumfahrt, Germany, and the Technical University of Munich. His research work is in coding and communication algorithms, bandwidth-efficient coding, and the practical application of these to speech and data transmission. He has served as a consultant in these fields in a number of countries. He is an author of the books *Digital Phase Modulation* (Plenum, 1986), *Modern Electrical Communication*, 2nd ed. (Prentice-Hall, 1988), *Source and Channel Coding: An Algorithmic Approach* (Kluwer, 1991), and the forthcoming *Digital Transmission Engineering* (IEEE Press).

Dr. Anderson was a member of the IEEE Information Theory Society Board of Governors during 1980–1987, serving as the Society's Vice President (1983–1984) and President (1985). He served on the Publications Board of IEEE during 1989–1991 and 1994–1996, and was Editor-in-Chief of IEEE Press during 1994–1996. In 1983, he was Co-Chairman of the IEEE International Symposium on Information Theory held in St. Jovite, Canada. He served as Associate Editor of the IEEE TRANSACTIONS ON INFORMATION THEORY (1980–1984) and as a Guest Editor of the IEEE TRANSACTIONS ON COMMUNICATIONS on several occasions. He received the Humboldt Research Prize (Germany) in 1991. In 1996, he was elected to the Swedish National Visiting Chair in Information Technology.