An Efficient Software Radio Implementation of the UMTS Turbo Codec

Matthew C. Valenti
Lane Dept. of Comp. Sci. & Elect. Eng.
West Virginia University
Morgantown, WV 26506-6109
email: mvalenti@wvu.edu

Abstract— **This paper addresses some critical implementation issues involved in the development of a turbo decoder, using the UMTS specification as a concrete example. The assumption is that the decoder is to be implemented in software rather than hardware, and thus a variable number of decoder iterations is not only possible, but desirable. Three twists on the decoding algorithm are proposed: (1) A linear approximation of the correction function used by the max* operator which reduces complexity with only a negligible loss in BER performance; (2) A method for normalizing the backward recursion which yields a 12.5% savings in memory usage; and (3) A simple method for halting the decoder iterations based only on the log-likelihood ratios.**

Keywords— **Coding, WCDMA (UMTS), Software radio.**

I. Introduction

Turbo codes have received a considerable amount of attention since their introduction [1]. They are particularly attractive for cellular communication systems, and have been included in the specifications for both WCDMA (UMTS) and cdma2000. At this time, the reasons for the superior performance of turbo codes and the associated decoding algorithm are, for the most part, understood.

The purpose of this paper is neither to explain the phenomenal performance of turbo codes, nor to rigorously derive the decoding algorithm. Rather, the purpose is to clearly explain an efficient decoding algorithm suitable for immediate implementation in a software radio receiver. In order to provide a concrete example, the discussion will be limited to the turbo code used by the UMTS specification. The decoding algorithm is based on the log-MAP algorithm [2], although many parts of the algorithm have been simplified without any loss in performance. In particular, the branch metrics used in the proposed algorithm are much simpler to compute, and the amount of storage is reduced by 12.5% by an appropriate normalization process. Some critical implementation issues are discussed, in particular the computation of the max* operator and the dynamic halting of the decoder iterations. Simple, but effective solutions to both of these problems are proposed and illustrated through simulation.

This work was supported by the Office of Naval Research under grant N00014-00-0655.

II. The UMTS Turbo Code

This section presents a brief overview of the turbo code used in the UMTS specification, as published by the Third Generation Partnership Project (3GPP). A more detailed explanation can be found in [3].

The UMTS turbo encoder is comprised of two rate 1/2, constraint length 4 recursive systematic convolutional (RSC) codes concatenated in parallel. The feedforward generator is (15) and the feedback generator is (13), both in octal. The number of data bits at the input of the turbo encoder is K, where $40 \leq K \leq 5114$. Data is encoded by the first (i.e. upper) encoder in its natural order, and by the second (i.e. lower) after being interleaved (details of the interleaving are found in the specification). The data bits are transmitted together with the parity bits generated by the two encoders. Thus, the overall encoder code rate is $r = 1/3$, not including the tail bits (discussed below). The output of the encoder is in the form: $X_1, Z_1, Z'_1, X_2, Z_2, Z'_2$, where X_k is the kth systematic (i.e. data) bit, Z_k is the parity output from the upper (uninterleaved) encoder and Z'_k is the parity output from the lower (interleaved) encoder.

The trellises of both encoders are forced back to the all-zeros state by the proper selection of tail bits. Unlike conventional convolutional codes, which can be terminated with a tail of zeros, the RSC encoder is terminated with the bits that are fed back (which are determined by the state of the encoder and the feedback polynomial). The tail bits are then transmitted at the end of the encoded frame according to X_{K+1}, Z_{K+1}, X_{K+2}, Z_{K+2}, X_{K+3}, Z_{K+3}, X'_{K+1}, Z'_{K+1}, X'_{K+2}, Z'_{K+2}, X'_{K+3}, Z'_{K+3}, where X represents the tail bits of the upper encoder, Z represents the parity bits corresponding to the upper encoder's tail, X' represents the tail bits of the lower encoder, and Z' represents the parity bits corresponding to the lower encoder's tail. Thus the number of coded bits is $3K + 12$, and the code rate is $K/(3K + 12)$ when tail bits are taken into account.

III. Channel Model

BPSK modulation is assumed along with either an AWGN or flat-fading channel. The output of the matched filter is $Y_k = a_k S_k + n_k$, where $S_k = 2X_k - 1$ for the systematic bits and $S_k = 2Z_k - 1$ for the parity bits, n_k is Gaussian noise with variance $\sigma^2 = (3K +$

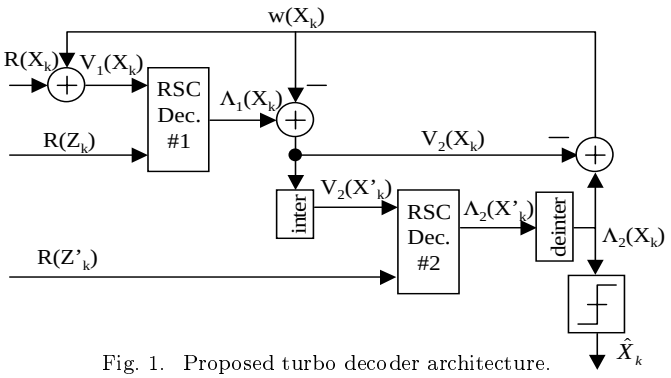

Fig. 1. Proposed turbo decoder architecture.

12)/(2$K(E_b/N_o)$), and a_k is the channel gain ($a_k = 1$ for AWGN and is a Rayleigh random variable for Rayleigh flat-fading).

The input to the decoder is assumed to be in log-likelihood ratio (LLR) form, which assures that the channel gain and noise variance have been properly taken into account. Specifically, the decoder input is $R_k = 2a_kY_k/\sigma^2$. For the remainder of the discussion, the notation $R(X_k)$ denotes the received LLR corresponding to systematic bit X_k, $R(Z_k)$ denotes the received LLR for the upper parity bit Z_k and $R(Z'_k)$ denotes the received LLR corresponding to the lower parity bit Z'_k.

IV. Decoder Architecture

The architecture of the decoder is as shown in Fig. 1. As indicated by the presence of a feedback path, the decoder operates in an iterative manner. Each full-iteration consists of two half-iterations, one for each constituent RSC code. The timing of the decoder is such that RSC decoder #1 operates during the first half-iteration, and RSC decoder #2 operates during the second half iteration. The operation of the RSC decoders is fully described in section VI.

The value $w(X_k)$ is the extrinsic information produced by decoder #2 and introduced to the input of decoder #1. Prior to the first iteration, $w(X_k)$ is initialized to all zeros. Because of the way that the branch metrics were derived, it is sufficient to simply add $w(X_k)$ to the systematic LLR input $R(X_k)$, which forms a new variable denoted $V_1(X_k)$. The input to RSC decoder #1 is both $V_1(X_k)$ and $R(Z_k)$, and the output is the LLR $\Lambda_1(X_k)$. By subtracting $w(X_k)$ from $\Lambda_1(X_k)$, a new variable $V_2(X_k)$ is formed. Similar to $V_1(X_k)$, $V_2(X_k)$ contains the sum of the systematic channel LLR and the extrinsic information produced by decoder #1 (note however that the extrinsic information for RSC decoder #1 never has to be explicitly computed). The input to decoder #2 is $V_2(X'_k)$, which is the interleaved version of $V_2(X_k)$, and $R(Z'_k)$, which is the channel LLR corresponding to the second encoder's parity bits. The output of RSC decoder #2 is the LLR $\Lambda_2(X'_k)$, which is deinterleaved to form $\Lambda_2(X_k)$. The extrinsic information $w(X_k)$ is then formed by subtracting $V_2(X_k)$ from $\Lambda_2(X_k)$ and is fed back to use during the next iteration. Once the iterations have been completed, a hard bit decision is taken using $\Lambda_2(X_k)$, where $\hat{X}_k = 1$ when $\Lambda_2(X_k) > 0$ and $\hat{X}_k = 0$ when $\Lambda_2(X_k) \leq 0$.

V. The Max* Operator

The RSC decoders in Fig. 1 are each executed using a version of the classic MAP algorithm implemented in the log-domain [2]. As will be discussed in Section VI, the algorithm is based on the Viterbi algorithm with two key modifications: First, the trellis must be swept through not only in the forward direction, but also in the reverse direction, and second, the add-compare-select (ACS) operation of the Viterbi algorithm is replaced with the Jacobi logarithm, also known as the max* operator [4]. Because the max* operator must be executed twice for each node in the trellis during each half-iteration, it constitutes a significant, and sometimes dominant, portion of the overall decoder complexity. The manner that max* is implemented is essential to the performance and complexity of the decoder, and several methods have been proposed for its computation. Below, we consider four variants of the algorithm: log-MAP, max-log-MAP, constant-log-MAP, and linear-log-MAP. The only difference between these algorithms is the manner in which the max* operation is performed.

A. Log-MAP Algorithm

With the log-MAP algorithm, the Jacobi logarithm is computed exactly using:

$$\begin{aligned} \max*(x,y) &= \ln\left(e^x + e^y\right) \\ &= \max(x,y) + ln\left(1 + e^{-|y-x|}\right) \\ &= \max(x,y) + f_c\left(|y-x|\right), \end{aligned} \tag{1}$$

where the correction function $f_c(|y-x|)$ can be implemented using either the log and exp functions in C or a look-up table. The log-MAP algorithm is the most complex of the four algorithms, but offers the best BER performance. The correction function used by log-MAP is illustrated in Fig. 2, along with the correction functions used by constant-log-MAP and linear-log-MAP.

B. Max-log-MAP algorithm

With the max-log-MAP algorithm, the Jacobi algorithm is loosely approximated using:

$$\max*(x,y) \approx \max(x,y), \tag{2}$$

i.e., the correction function of the log-MAP algorithm is not used. The max-log-MAP algorithm is the least complex of the four algorithms (it has twice the complexity

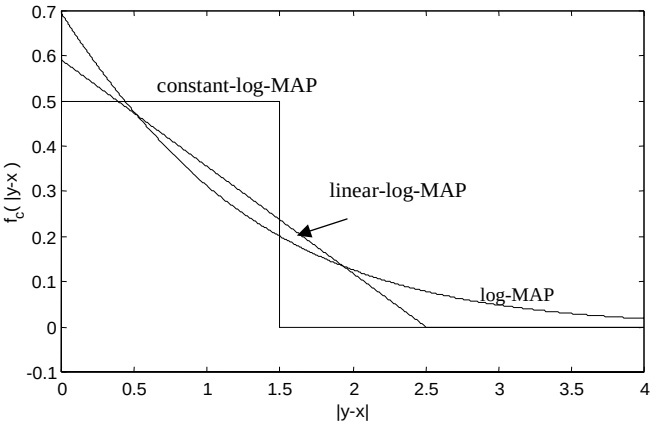

Fig. 2. Correction function used by log-MAP, linear-log-MAP, and constant-log-MAP algorithms.

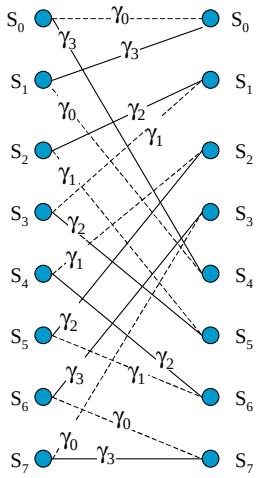

Fig. 3. Trellis section for the RSC code used by the UMTS turbo code. Solid lines indicate data 1 and dotted lines indicate data 0. Branch metrics are indicated.

of the Viterbi algorithm for each half-iteration), but offers the worst BER performance (about 0.4 dB worse than log-MAP). The max-log-MAP algorithm has the additional benefit of being almost completely tolerant of imperfect noise variance estimates when operating on an AWGN channel.

C. Constant-log-MAP algorithm

The constant-log-MAP algorithm, approximates the Jacobi algorithm using [5]:

$$\max*(x,y) \approx \max(x,y) + \begin{cases} 0 & \text{if } |y-x| > T \\ C & \text{if } |y-x| \le T \end{cases} \quad (3)$$

where it is shown in [6] that the best values for the UMTS turbo code are $C = 0.5$ and $T = 1.5$. This algorithm is equivalent to the log-MAP algorithm, with the correction function implemented by a 2-element lookup table. The performance and complexity is between that of the log-MAP and max-log-MAP algorithms, although the BER performance is only about 0.03 dB worse than log-MAP. However, a disadvantage of constant-log-MAP is that it is more susceptible to noise variance estimation errors than is log-MAP.

D. Linear-log-MAP algorithm

The linear-log-MAP algorithm, first introduced in [7], uses the following linear approximation to the Jacobi algorithm:

$$\max*(x,y) \approx \max(x,y) + \begin{cases} 0 & \text{if } |y-x| > T \\ a|y-x| + b & \text{if } |y-x| \le T \end{cases} \quad (4)$$

In [7], the values of the parameters a, b, and T were picked for convenient fixed point implementation. Since we are assuming a floating-point processor is available, a better solution would be to find these parameters by minimizing the total squared error between the exact correction function and its linear approximation. By performing this minimization, we have found that $a = -0.236$, $b = 0.592$, and $T = 2.508$. The linear-log-MAP offers performance and complexity between that of the log-MAP and constant-log-MAP algorithms. As will be shown in the simulation results, a key advantage of the linear-log-MAP algorithm is that it converges faster than constant-log-MAP.

VI. MAP Algorithm in the Log Domain

The RSC decoders operate by sweeping through the code trellis in both the forward and reverse directions using the modified Viterbi algorithm (the modification is that the ACS operations are replaced with max*). Then, LLR values can be computed for each stage of the trellis. Two key observations should be pointed out before going into the details of the algorithm: First, it does not matter whether the forward sweep or the reverse sweep is performed first; second, only the partial path metrics for the entire first sweep (forward or backward) must be stored in memory. The LLR values can be computed during the second sweep, and thus partial path metrics for only two stages of the trellis (the current and previous stages) must be maintained for the second sweep.

A. Trellis Structure and Branch Metrics

The trellis of the RSC encoder used by the UMTS turbo code is shown in Fig. 3. Solid lines indicate data $X_k = 1$ and dotted lines indicate data $X_k = 0$. The branch metric associated with the branch connecting states S_i (on the left) and S_j (on the right) is $\gamma_{ij} = V(X_k)X(i,j) + R(Z_k)Z(i,j)$, where $X(i,j)$ is the data bit associated with the branch and $Z(i,j)$ is the parity bit associated with the

branch. Because the RSC encoder is rate 1/2, there are only 4 distinct branch metrics:

$$\begin{aligned}\gamma_0 &= 0\\ \gamma_1 &= V(X_k)\\ \gamma_2 &= R(Z_k)\\ \gamma_3 &= V(X_k)+R(Z_k),\end{aligned} \tag{5}$$

where for decoder #1 $V(X_k) = V_1(X_k)$ and for decoder #2 $V(X_k) = V_2(X'_k)$ and $R(Z_k) = R(Z'_k)$.

B. Backward Recursion

The proposed decoder begins with the backward recursion, saving normalized partial path metrics at all the nodes in the trellis (with an exception noted below), which will later be used to calculate the LLRs during the forward recursion. The backward partial path metric for state S_i at trellis stage k is denoted $\beta_k(S_i)$ with $1 \le k \le K+3$ and $0 \le i \le 7$. The backward recursion is initialized with $\beta_{K+3}(S_0) = 0$ and $\beta_{K+3}(S_i) = -\infty\ \forall i > 0$.

Beginning with stage $k = K+2$ and proceeding through the trellis in the backward direction until stage $k = 1$, the partial path metrics are found according to:

$$\begin{aligned}\tilde{\beta}_k(S_i) &= max*\{(\,\beta_{k+1}(S_{j_1})+\gamma_{ij_1}\,),\\ &\qquad (\,\beta_{k+1}(S_{j_2})+\gamma_{ij_2}\,)\},\end{aligned} \tag{6}$$

where the tilde above $\beta_k(S_i)$ indicates that the metric has not yet been normalized, and S_{j_1} and S_{j_2} are the two states at stage $k+1$ in the trellis that are connected to state S_i at stage k. After the calculation of $\tilde{\beta}_k(S_0)$, the partial path metrics are normalized according to:

$$\beta_k(S_i) = \tilde{\beta}_k(S_i) - \tilde{\beta}_k(S_0). \tag{7}$$

Because after normalization $\beta_k(S_0) = 0\ \forall k$, only the other seven normalized partial path metrics $\beta_k(S_i)$, $1 \le i \le 7$, need to be stored. This constitutes a 12.5% savings in memory relative to either no normalization or other common normalization techniques.

C. Forward Recursion and LLR Calculation

During the forward recursion, the trellis is swept through in the forward direction in a manner similar to the Viterbi algorithm. Unlike the backward recursion, only the partial path metrics for two stages of the trellis must be maintained: The current stage k and the previous stage $k-1$. The forward partial path metric for state S_i at trellis stage k is denoted $\alpha_k(S_i)$ with $0 \le k \le K-1$ and $0 \le i \le 7$. The forward recursion is initialized by setting $\alpha_0(S_0) = 0$ and $\alpha_0(S_i) = -\infty\ \forall i > 0$.

Beginning with stage $k = 1$ and proceeding through the trellis in the forward direction until stage $k = K$[1], the unnormalized partial path metrics are found according to:

$$\begin{aligned}\tilde{\alpha}_k(S_j) &= max*\{(\,\alpha_{k-1}(S_{i_1})+\gamma_{i_1 j}\,),\\ &\qquad (\,\alpha_{k-1}(S_{i_2})+\gamma_{i_2 j}\,)\},\end{aligned} \tag{8}$$

where S_{i_1} and S_{i_2} are the two states at stage $k-1$ that are connected to state S_j at stage k. After the calculation of $\tilde{\alpha}_k(S_0)$, the partial path metrics are normalized using:

$$\alpha_k(S_i) = \tilde{\alpha}_k(S_i) - \tilde{\alpha}_k(S_0). \tag{9}$$

As the α's are computed for stage k, the algorithm can simultaneously obtain an LLR estimate for data bit X_k. This LLR is found by first noting that the likelihood of the branch connecting state S_i at time $k-1$ to state S_j at time k is:

$$\lambda_k(i,j) = \alpha_{k-1}(S_i) + \gamma_{ij} + \beta_k(S_j) \tag{10}$$

The likelihood of data 1 (or 0) is then the Jacobi logarithm of the likelihood of all branches corresponding to data 1 (or 0), and thus:

$$\begin{aligned}\Lambda(X_k) &= \underset{S_i \to S_j : X_i = 1}{max*}\{\lambda_k(i,j)\}\\ &\qquad - \underset{S_i \to S_j : X_i = 0}{max*}\{\lambda_k(i,j)\},\end{aligned} \tag{11}$$

where the max* operator is computed recursively over the likelihoods of all data 1 branches ($S_i \to S_j : X_i = 1$) or data 0 branches ($S_i \to S_j : X_i = 0$). Once $\Lambda(X_k)$ is calculated, $\alpha_{k-1}(S_i)$ is no longer needed and may be discarded.

VII. Simulation Results

Simulations were run to illustrate the performance of all four variants of the decoding algorithm. For the simulation, the frame/interleaver size of K=5114 bits was used, and up to 14 decoder iterations were performed. Results for both AWGN and fully interleaved Rayleigh flat-fading channels were produced. Fig. 4 shows the bit error rate (BER) and Fig. 5 shows the frame error rate (FER) in both AWGN and Rayleigh flat-fading, while Fig. 6 shows the average number of iterations required for the decoder to converge in AWGN (a value of 15 on this curve indicates that the decoder did not converge). In order to present a fair comparison, all four algorithms decoded the same received codewords, and thus the data, noise, and fading were the same for all four curves.

In all cases, the performance of max-log-MAP is noticeably worse than the other three algorithms: At BER = 10^{-5}, max-log-MAP is 0.412 dB worse than log-MAP in AWGN. The other three algorithms have roughly the same performance, although linear-log-MAP is always better than constant-log-MAP: In AWGN and at BER = 10^{-5}, constant-log-MAP is 0.028 dB worse than log-MAP, while linear-log-MAP is only 0.008 dB worse than log-MAP. At

[1]Note that α_k does not need to be computed when $k = K$ (it is never used), although the LLR $\Lambda(X_k)$ must still be found.

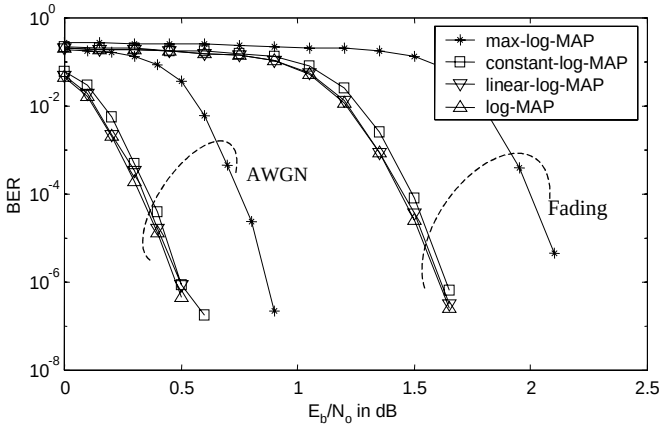

Fig. 4. BER of K=5114 UMTS turbo code after 14 decoder iterations.

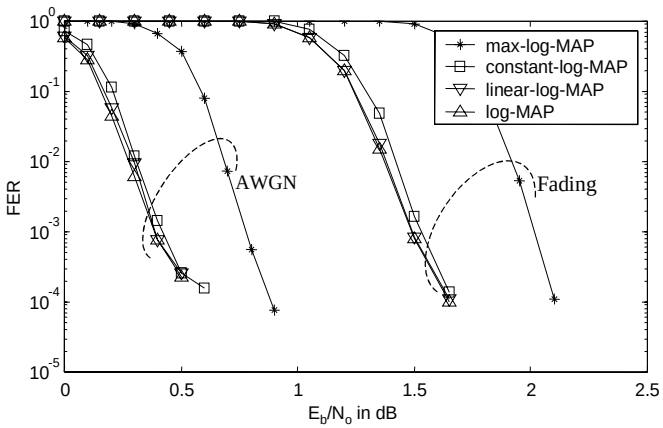

Fig. 5. FER of K=5114 UMTS turbo code after 14 decoder iterations.

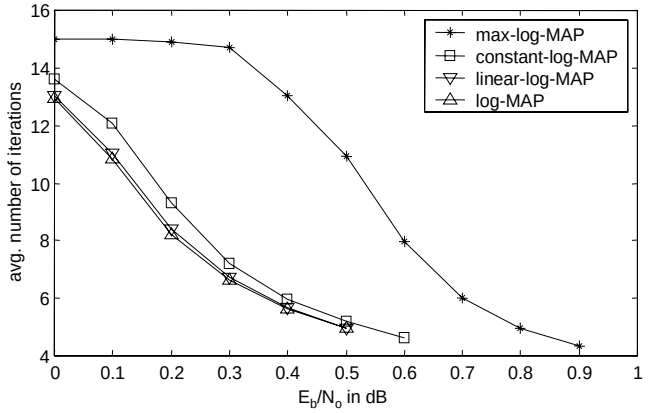

Fig. 6. Average number of decoder iterations required for the K=5114 to converge in AWGN.

$E_b/N_o = 0.4$ dB, max-log-MAP required about 13 iterations, log-MAP about 5.63, linear-log-MAP about 5.67, and constant-log-MAP about 5.98. Thus an additional benefit of linear-log-MAP is that it requires fewer decoder iterations than constant-log-MAP.

VIII. Dynamic Halting Condition

The simulation results from the previous section assumed that the decoder halted as soon as it converged, i.e. when the BER for the frame went to zero. This requires knowledge of the data, which is available when running a computer simulation. However, in practice, the decoder will not have knowledge of the data, and thus a blind method for halting the iterations must be employed. Because the decoder rarely requires the maximum number of iterations to converge, using an early stopping criterion will allow a much greater throughput in a software radio implementation.

Several other early stopping criteria have been proposed based on cross entropy between iterations or on the sign-difference ratio [8]. The decoder considered here uses a simpler, but effective, stopping criteria based only on the log-likelihood ratio. The decoder stops once the absolute value of all of the LLRs are above a threshold, Λ_T, i.e. the decoder halts once

$$\min_{1 \le k \le K} \{|\Lambda_2(X_k)|\} \quad > \quad \Lambda_T. \tag{12}$$

The performance of the stopping condition is highly dependent on the choice of Λ_T. If it is too small, then the decoder will tend to not perform enough iterations and BER performance will suffer. If, however, it is too large, then the decoder will tend to overiterate, and the throughput will suffer. Through simulation, a value of $\Lambda_T = 10$ was found to be acceptable.

The K=640 bit UMTS turbo code was simulated in AWGN using both ideal halting (i.e. halt once the decoder converges) and halting using various values for Λ_T. The decoder used a maximum of 10 iterations of the constant-log-MAP algorithm and each curve was generated using the same received code words. BER results are shown in Fig. 7, FER results are shown in Fig. 8, and the average number of decoder iterations is shown in Fig. 9. As can be seen, $\Lambda_T = 1$ and $\Lambda_T = 5$ are too small and raise the BER floors, while $\Lambda_T = 10$ raises the FER floor only slightly and has only a negligible effect on the BER floor. Using the threshold $\Lambda_T = 10$ requires, on average, less than one extra iteration compared to ideal halting.

It is interesting to note that the BER is sometimes lower with $\Lambda_T = 10$ than with ideal halting. The reason for this is as follows: The number of errors at the output of a turbo decoder will sometimes oscillate from one iteration to the next. If the received codeword is too corrupted to successfully decode, "ideal halting" will always run the full number of iterations; thus, the number of bit errors will be

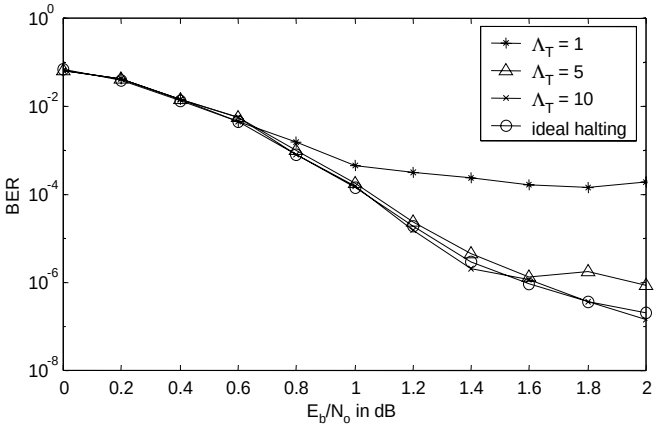

Fig. 7. BER of K=640 UMTS turbo code in AWGN after 10 iterations of constant-log-MAP decoding with various halting thresholds.

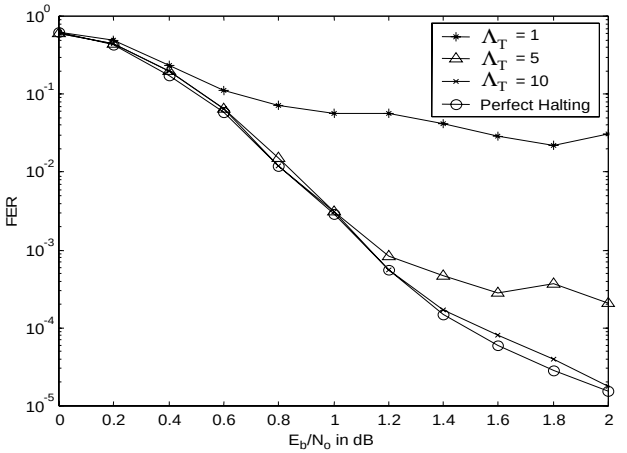

Fig. 8. FER of K=640 UMTS turbo code in AWGN after 10 iterations of constant-log-MAP decoding with various halting thresholds.

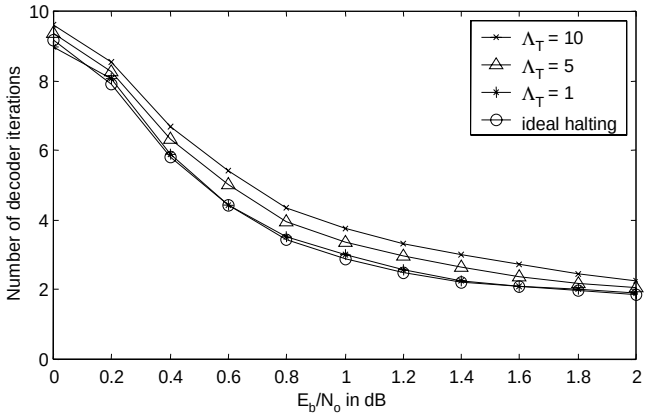

Fig. 9. Average number of decoder iterations required for the K=640 to converge in AWGN using constant-log-MAP decoding and various halting thresholds.

dictated by the performance at the last iteration, which due to the oscillatory nature of the decoder, could be quite high. On the other hand, the early halting decoder will stop the iterations when the LLRs are high, even if the BER is not identically zero. Thus, although early halting cannot lower the FER, it can lower the BER by having fewer bit errors when there is a frame error.

IX. Conclusions

Three aspects regarding the implementation of the UMTS turbo codec have been discussed in this paper. First, a simple, but effective, linear approximation to the Jacobi algorithm was proposed. This approximation offers better performance and faster convergence than the constant-log-MAP algorithm at the expense of only a modest increase in complexity. Second, a method for normalizing the partial path metrics was proposed that eliminates the need to store the metrics for state S_0. Finally, a method for halting the decoder iterations based only on the current value of the LLR's was proposed.

One weakness of the constant-log-MAP algorithm discussed in [6] is that it is more vulnerable to incorrect noise variance estimates than is the log-MAP algorithm. An additional benefit of the linear-log-MAP algorithm, which is not shown here, is that it is more tolerant of noise variance estimate mismatch than is constant-log-MAP. Future work should carefully examine how the linear-log-MAP algorithm behaves in the presence of imprecise noise variance estimates.

References

[1] C. Berrou, A. Glavieux, and P. Thitimasjshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes(1)," in *Proc., IEEE Int. Conf. on Commun.*, (Geneva, Switzerland), pp. 1064–1070, May 1993.

[2] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and suboptimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. on Telecommun.*, vol. 8, pp. 119–125, Mar./Apr. 1997.

[3] European Telecommunications Standards Institute, "Universal mobile telecommunications system (UMTS): Multiplexing and channel coding (FDD)," *3GPP TS 125.212 version 3.4.0*, pp. 14–20, Sept. 23, 2000.

[4] A. J. Viterbi, "An intuitive justification and a simplified implemetation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 260–264, Feb. 1998.

[5] W. J. Gross and P. G. Gulak, "Simplified map algorithm suitable for implementation of turbo decoders," *Electronics Letters*, vol. 34, pp. 1577–1578, Aug. 6, 1998.

[6] B. Classon, K. Blankenship, and V. Desai, "Turbo decoding with the constant-log-MAP algorithm," in *in Proc., Second Int. Symp. Turbo Codes and Related Appl.*, (Brest, France), pp. 467–470, Sept. 2000.

[7] J.-F. Cheng and T. Ottosson, "Linearly approximated log-MAP algorithms for turbo coding," in *Proc., IEEE Veh. Tech. Conf.*, (Houtson, TX), May 2000.

[8] Y. Wu, B. D. Woerner, and W. J. Ebel, "A simple stopping criterion for turbo decoding," *IEEE Commun. Letters*, vol. 4, pp. 258–260, Aug. 2000.