Journal of the Chinese Institute of Engineers
Vol. 35, No. 4, June 2012, 431–437

Taylor & Francis
Taylor & Francis Group

The modified wrap-around Viterbi algorithm for convolutional tail-biting codes

Yunghsiang S. Han[a]*, Ting-Yi Wu[b], Hung-Ta Pai[c] and Po-Ning Chen[b]

[a]*Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei City, Taiwan, R.O.C.;*
[b]*Department of Electrical Engineering, National Chiao-Tung University, Hsinchu City, Taiwan, R.O.C.;*
[c]*Graduate Institute of Communication Engineering, National Taipei University, Taipei, Taiwan, R.O.C.*

(*Received 9 September 2010; final version received 8 February 2011*)

Recently, two near-optimal decoding algorithms [Shao, R.Y., Lin, S., and Fossorier, M.P.C., 2003. Two decoding algorithms for tailbiting codes. *IEEE transactions on communications*, 51 (10), 1658–1665; Krishnan, K.M. and Shankar, P., 2006. Approximate linear time ML decoding on tail-biting trellises in two rounds. *In IEEE international symposium on information theory*, Seattle, WA, USA, pp. 2245–2249] have been proposed for convolutional tail-biting codes. Both algorithms iterate the Viterbi algorithm twice, but use different metrics in the second iteration. Simulations showed that the latter algorithm (Krishnan and Shankar 2006) improved on the earlier one (Shao *et al*. 2003) in word error rates at the price of additional storage consumption. In this work, we prove that with a proper modification to the earlier one, the two algorithms can be made to have exactly the same survivor path at each state in the trellis, and hence are equivalent in error performance. One can consequently adopt the modified algorithm to alleviate the need for extra storage consumption of the later algorithm and, at the same time, achieve equally good performance.

Keywords: Viterbi algorithm; tail-biting codes; tail-biting trellises

1. Introduction

Nowadays, convolutional tail-biting codes (CTBC) are popular because they can neutralize the loss on the code rate and result in less-performance degradation than the zero-tail convolutional codes (ZTCC) (Wang and Bhargava 1989, Shao *et al*. 2003). Unlike the ZTCCs for which all code paths on the trellis are required to start from and end at the zero state, the code paths of the CTBCs can start from a state other than the zero one, as long as they end at the same state. For convenience, we will refer to the code paths on the CTBC trellis as the *tail-biting paths*. Since the decoding of the CTBC requires additional identification of the initial (equivalently, the final) state corresponding to the transmitted codeword, and since the number of states grows exponentially with the memory order, the maximum-likelihood (ML) decoding of the CTBC becomes much more involved than the ML decoding of the ZTCC.

In the literature, several suboptimal decoding algorithms for the CTBC have been proposed (Cox and Sundberg 1994, Shao *et al*. 2003, Krishnan and Shankar 2006, Chen and Tsai 2008). Among them, the wrap-around Viterbi algorithm (WAVA) (Shao *et al*. 2003) and the approximate linear time ML decoding algorithm (ALTMLA) (Krishnan and Shankar 2006) are the two that can achieve the least decoding complexity. In comparison with the WAVA, the ALTMLA has better word error rate (WER) performance but requires more memory storage. Further improvement on the average number of iterations and memory consumption of the WAVA is later conducted in Chen and Tsai (2008). Comparison of the WAVA with an improvement over the decoding algorithm given in Cox and Sundberg (1994) and the bidirectional Viterbi algorithm (VA) over the fading channel can be found in Zhang *et al*. (2009). Recently, an ML-decoding algorithm for the CTBC has been proposed (Pai *et al*. 2008). Although optimal in performance, the ML decoding algorithm, of which the decoding complexity varies with the signal-to-noise ratio (SNR) as contrary to the constant-decoding complexity of the WAVA with fixed number of iterations, may only be suitable for a software implementation.

In this work, we prove that even though the WAVA looks different from the ALTMLA, it can be made completely equivalent to the ALTMLA by a proper modification. The modified WAVA (which will be termed the MWAVA in the sequel) and the ALTMLA then give exactly the same survivor path

*Corresponding author. Email: yshan@mail.ntust.edu.tw

ISSN 0253–3839 print/ISSN 2158–7299 online

http://dx.doi.org/10.1080/02533839.2012.655905
http://www.tandfonline.com

at each state in the decoding trellis, and hence have equally good error performance. One can accordingly adopt the MWAVA to alleviate the need for the extra storage required by the ALTMLA with no sacrifice in performance. The establishment of this equivalence also facilitates the use of the ALTMLA to provide an explanation on why the MWAVA (as well as the WAVA) performs well with only two iterations.

2. Background and refinement of the WAVA

Denote by $\mathcal{C}$ the (n, l, m) CTBC with information bits of length L, where n is the number of output bits per information bit, and m is the memory order. The trellis of $\mathcal{C}$ then has 2^m states at each level, and is of $L+1$ levels. As mentioned in the previous section, the corresponding tail-biting paths for the codewords in $\mathcal{C}$ should constrain on the same initial and final state. For convenience in later discussion, we denote by $\mathcal{C}_s$ the *super code* of $\mathcal{C}$, which additionally takes in all paths that end at a final state different from the initial state.

Let $v \stackrel{\Delta}{=} (v_0, v_1, \ldots, v_{N-1})$ be a binary codeword of $\mathcal{C}$, where $N = nL$ and each $v_j \in \{0, 1\}$. It can be verified (Han *et al.* 1993) that the ML decoding output v^* for the received vector r satisfies

$$\sum_{j=0}^{N-1} \left(\phi_j - (-1)^{v_j^*}\right)^2 \leq \sum_{j=0}^{N-1} \left(\phi_j - (-1)^{v_j}\right)^2 \quad \text{for all } v \in \mathcal{C},$$

where $\phi_j \stackrel{\Delta}{=} \text{In}[\Pr(r_j|0)/\Pr(r_j|1)]$ and $\text{pr}(\cdot|\cdot)$ is the channel transition probability. A metric for a path in the trellis can then be defined as follows.

Definition 1: For a path with label $x_{(ln-1)}^{(i,s)} = (x_0^{(i,s)}, x_1^{(i,s)}, \ldots, x_{(ln-1)}^{(i,s)}) \in \{0, 1\}^{ln}$, which starts from state i at level 0 and ends at state s at level l, the M-metric associated with it is defined as

$$M\left(\boldsymbol{x}_{(ln-1)}^{(i,s)}\right) \stackrel{\Delta}{=} \sum_{j=0}^{ln-1} M\left(x_j^{(i,s)}\right),$$

where

$$M\left(x_j^{(i,s)}\right) \stackrel{\Delta}{=} \left(\phi_j - (-1)^{x_j^{(i,s)}}\right)^2.$$

Based on the above definition, the target of the ML decoding is to find the tail-biting path with the smallest M-metric in the trellis.

When directly applying the VA onto the CTBC trellis, the resultant survivor paths may not end at the state from which they start. Hence, the path found by the VA is only guaranteed to be contained in the super code $\mathcal{C}_s$ but does not necessarily correspond to a codeword in $\mathcal{C}$. In order to solve this problem, WAVA was proposed in Shao *et al.* (2003), which tried to locate the best tail-biting path by iterating the VA by the way of wrapping the final states of the CTBC trellis around the respective initial states.

According to the simulation results given in Shao *et al.* (2003), the WAVA can achieve near ML performance with only two iterations. The two-iteration WAVA is outlined in the following:

The first iteration (Iter1)

Iter1-Step 1: Associate zero initial metric with every zero-length path containing only the initial state at level 0. Record $\pi(i)$ as the initial metric associated with state i at level 0.

Iter1-Step 2: Apply the VA using M-metric with initial metric π. At the end of the first iteration, if the best survivor path is also a tail-biting path, output it as the decoding result, and stop the algorithm; otherwise, proceed with the next iteration.

The second iteration (Iter2)

Iter2-Step 1: Initialize every zero-length path containing only the initial state at level 0 with the metric of the first-iteration survivor path ending at the same state at level L. Record $\pi(i)$ as the initial metric associated with state i at level 0.

Iter2-Step 2: Apply the VA again using M-metric with initial metric π. At the end of the 2nd iteration, output the best tail-biting path if it exists; otherwise, output the best survivor path ending at level L (in which case, a WER event may occur).

In the above algorithm, a set of 2^m survivors results after the completion of the first iteration. Notably, these survivor paths do not necessarily correspond to codewords in $\mathcal{C}$, and they may end at some final states different from the ones from which they start. In such a case, the algorithm proceeds with the next iteration. However, if none of the survivor paths are tail-biting paths after two iterations of the VA, the algorithm simply outputs the best survivor path (which is of course not a codeword) and may admit a failure in error correcting.

Another suboptimal algorithm, named ALTMLA, was proposed recently (Krishnan and Shankar 2006). The ALTMLA needs to record the metric of every intermediate survivor ever explored during the execution of the first iteration. These recorded metrics will then be incorporated into the new metric used in the second VA iteration, which is defined below.

Definition 2: Denote by $c_l(s)$ the path metric of the survivor ending at state s at level l, which is recorded during the execution of the first VA iteration.

Let $\pi(i) = c_L(i)$ be the initial metric at state i at level 0. Then for a path with label $x_{(ln-1)}^{(i,s)} = (x_0^{(i,s)}, x_1^{(i,s)}, \ldots, x_{ln-1}^{(i,s)}) \in \{0,1\}^{ln}$, which starts from state i at level 0 and ends at state s at level l, the accumulative path metric employed by the ALTMLA in the second VA iteration is defined as

$$\bar{M}\left(\boldsymbol{x}_{(ln-1)}^{(i,s)}\right) \triangleq \pi(i) + M\left(\boldsymbol{x}_{(ln-1)}^{(i,s)}\right) - c_l(s).$$

Due to the compensation for $c_l(s)$ in the new metric, the ALTMLA improves the performance of the WAVA. Krishnan and Shankar (2006) also observed that when the metric of a survivor path obtained from the first iteration is no less than the metric of the best first-iteration survivor tail-biting path, the best tail-biting path obtained from the second iteration can never start from the ending state of this survivor path. They accordingly exclude those paths starting from such state by setting their initial metrics to infinity.

Although the presentation of the ALTMLA looks dissimilar to the WAVA, we found that the latter algorithm can be made equivalent to the former by a proper modification.

Modified two-iteration WAVA:

The first iteration (Iter1): The same as the WAVA.
The second iteration (Iter2)

Iter2-Step 1″: Initialize every zero-length path containing only the initial state at level 0 with the metric of the first-iteration survivor path ending at the same state at level L. Replace those initial metrics with infinity if they are not less than the minimum metric among all survivor tail-biting paths obtained in the first iteration. Record $\pi(i)$ as the initial metric associated with state i at level 0.

Iter2-Step 2″: Apply the VA again using M-metric with initial metric π. At the end of the second iteration, subtract $\pi(i)$ from the final metric at state i at level L. Output the best tail-biting path if it exists; otherwise, output the best survivor path ending at level L (in which case, a WER event may occur).

Since the ALTMLA and the MWAVA employ the same metric in their first VA iteration, their equivalence can be sufficiently substantiated by showing that both algorithms yield the same survivor path at each state at their second iterations. It can be proved by induction on levels of the trellis. When $l = 0$, it is clear from Definition 2 that

$$\bar{M}\left(\boldsymbol{x}_{(-1)}^{(i,i)}\right) \triangleq \pi(i) + M\left(\boldsymbol{x}_{(-1)}^{(i,i)}\right) - c_0(i) = \pi(i)$$

since $M(\boldsymbol{x}_{(-1)}^{(i,i)}) = 0$ and $c_0(i) = 0$. The equivalence of the two algorithms at the first level is thus confirmed.

Suppose that the ALTMLA and the MWAVA have the same survivor path at each state at level $(k-1)$. Assume that the two paths that enter state s at level k are, respectively, labeled with $\boldsymbol{x}_{(kn-1)}^{(i_1,s)}$ and $\boldsymbol{x}_{(kn-1)}^{(i_2,s)}$, where i_1 and i_2 are, respectively, the initial states of the two entering paths. Then, for the MWAVA, the accumulated metrics of the two entering paths are, respectively,

$$\pi(i_1) + M\left(\boldsymbol{x}_{(kn-1)}^{(i_1,s)}\right) \quad \text{and} \quad \pi(i_2) + M\left(\boldsymbol{x}_{(kn-1)}^{(i_2,s)}\right).$$

By the assumption that the ALTMLA and the MWAVA have the same survivor path at each state up to level $(k-1)$, the entering paths into state s at level k for the ALTMLA should be the same (as the MWAVA), and their ALTMLA metrics are, respectively,

$$\bar{M}\left(\boldsymbol{x}_{(kn-1)}^{(i_1,s)}\right) = \pi(i_1) + M\left(\boldsymbol{x}_{(kn-1)}^{(i_1,s)}\right) - c_k(s)$$

and

$$\bar{M}\left(\boldsymbol{x}_{(kn-1)}^{(i_2,s)}\right) = \pi(i_2) + M\left(\boldsymbol{x}_{(kn-1)}^{(i_2,s)}\right) - c_k(s).$$

Therefore, both algorithms have the same survivor at state s at level k because subtraction of the same quantity $c_k(s)$ does not alter the comparison result between the two.

Finally, we can derive from Iter2-Step 2″ of the MWAVA that the survivor path ending at state s at level L has accumulative metric

$$\begin{aligned}
&\pi(i) + M\left(\boldsymbol{x}_{(Ln-1)}^{(i,s)}\right) - \pi(s) \\
&\quad = \pi(i) + M\left(\boldsymbol{x}_{(Ln-1)}^{(i,s)}\right) - c_L(s) \\
&\quad = \bar{M}\left(\boldsymbol{x}_{(Ln-1)}^{(i,s)}\right)
\end{aligned}$$

for both the MWAVA and the ALTMLA because the MWAVA will subtract $\pi(s)$ from the final metric at state s at level L, and $c_L(s) = \pi(s)$. Since the above argument holds true for every final state, s, the proof is then completed.

As the MWAVA does not need to record the metric of the survivor path at each state during the first iteration, it can save the storage space of $2^m(L+1)$ required by the ALTMLA. The $\bar{M}$-metric employed by the ALTMLA nonetheless provides an explanation on why the MWAVA (as well as the WAVA) performs well with only two iterations. According to Shankar *et al.* (2006), the $\bar{M}$-metric is indeed an estimate about the M-metric of a tail-biting path passing through a particular state. Hence, what the ALTMLA (or the MWAVA) intends to do is to retain the prospective best tail-biting path at each particular state.

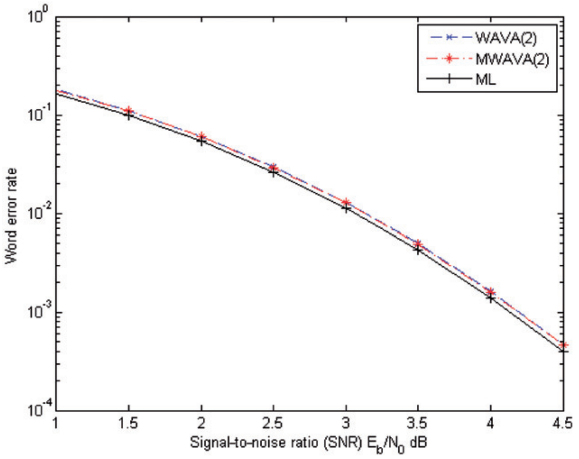

Figure 1. The WERs of the two-iteration WAVA (WAVA(2)), the two-iteration MWAVA (MWAVA(2)), and the ML decoders for the (2,1,4) CTBC with generator 72,62.

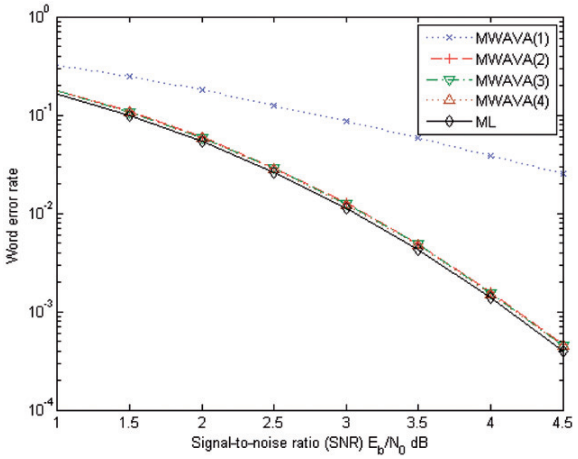

Figure 2. The WERs of the t-iteration MWAVA (MWAVA(t)) and the ML decoder for the (2,1,4) CTBC with generator 72,62.

3. Simulation results over AWGN channels

In this section, we examine the WERs of the WAVA and the MWAVA (equivalently, the ALTMLA) by simulations over the additive white Gaussian noise (AWGN) channels. We assume that the codeword is antipodally modulated. Hence, the j-th received scalar is given by

$$r_j = (-1)^{v_j}\sqrt{E} + \lambda_j,$$

for $0 \le j \le N-1$, where E is the signal energy per channel bit, and $\{\lambda_j\}_{j=0}^{N-1}$ are independent noise samples of a white Gaussian process with single-sided noise power per hertz N_0. The SNR for the channel is therefore E/N_0. In order to account for the code redundancy for different code rates, we will use the SNR per information bit as

$$\frac{E_b}{N_0} = \frac{NE/L}{N_0} = n\left(\frac{E}{N_0}\right).$$

For all simulations below, at least 30 word errors have been reported to ensure that there is no bias on the simulation results.

We first consider the (2,1,4) binary CTBC with generator 72,62 (octal). The length of the information bits simulated is $L = 20$.

Figure 1 compares the WER of the two-iteration WAVA with that of the two-iteration MWAVA for the (2,1,4) CTBC. As shown in the figure, the two-iteration MWAVA (equivalently, the ALTMLA) has slightly better performance than the two-iteration WAVA. It is also noted from Figure 1 that the two-iteration WAVA and the two-iteration MWAVA are only slightly inferior to the optimal performance for all E_b/N_0 simulated.

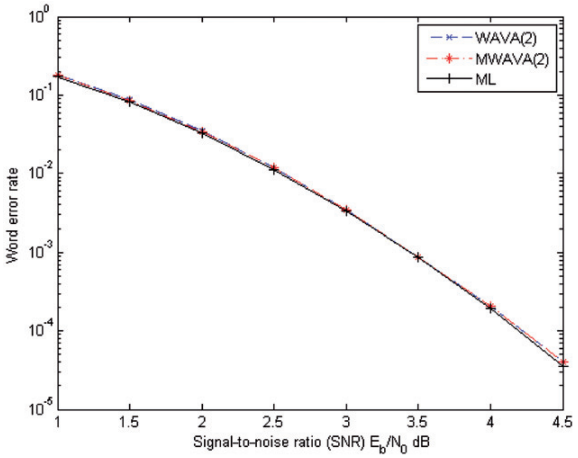

Figure 3. The WERs of the two-iteration WAVA (WAVA(2)), the two-iteration MWAVA (MWAVA(2)), and the ML decoders for the (2,1,6) CTBC with generator 554,744.

In Figure 2, we investigate the effect of iterations on the MWAVA (equivalently, the ALTMLA) for the (2,1,4) CTBC. According to Figure 2, the second iteration produces the largest gain on WER performance. In addition, with four iterations, a comparable performance to the ML decoder can be achieved.

We next consider the (2,1,6) binary CTBC with generator 554,744 (octal). The length of the information bits simulated is $L = 48$.

In Figure 3, the WERs of the two-iteration WAVA and the two-iteration MWAVA for the (2,1,6) CTBC are presented. The two-iteration MWAVA

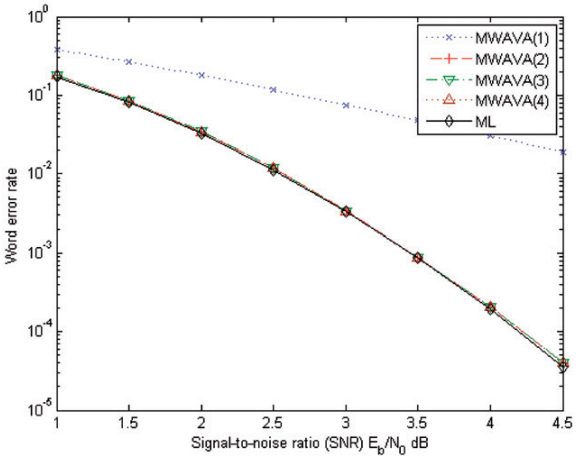

Figure 4. The WERs of the t-iteration MWAVA (MWAVA(t)) and the ML decoder for the (2,1,6) CTBC with generator 554,744.

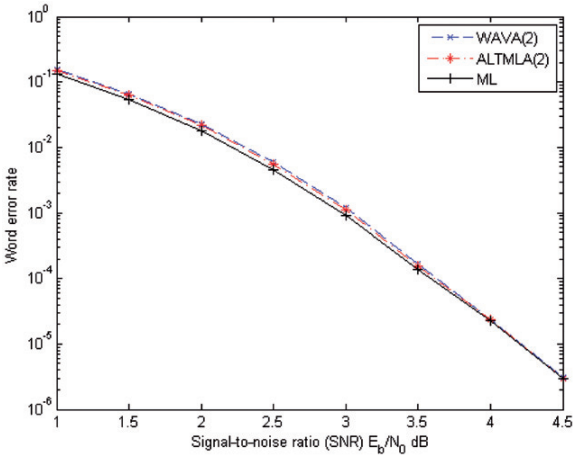

Figure 5. The WERs of the two-iteration WAVA (WAVA(2)), the two-iteration MWAVA (MWAVA(2)), and the ML decoders for the (2,1,8) CTBC with generator 515,677.

(equivalently, the ALTMLA) again has slightly better performance than the two-iteration WAVA. However, the two-iteration WAVA and the two-iteration MWAVA now yield almost identical performance to the ML decoder for all E_b/N_0 simulated, and hence are near optimal as having been addressed in Shao *et al.* (2003).

Figure 4 investigates the effect of iterations on the MWAVA (equivalently, the ALTMLA) for the (2,1,6) CTBC. The same as the results obtained from the (2,1,4) CTBC, the second iteration produces the largest gain on WER performance, now only with two

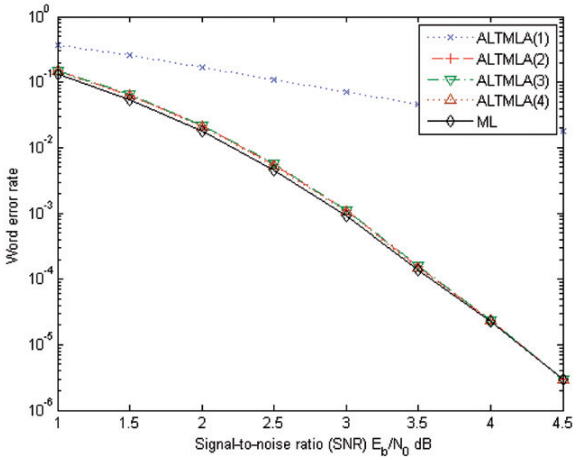

Figure 6. The WERs of the t-iteration MWAVA (MWAVA(t)) and the ML decoder for the (2,1,8) CTBC with generator 515,677.

iterations. An almost identical performance to the ML decoder can be achieved.

We subsequently turn to the (2,1,8) binary CTBC with generator 515,677 (octal). The length of the information bits simulated is $L = 48$.

In Figure 5, we summarize the WERs of the two-iteration WAVA and the two-iteration MWAVA for the (2,1,8) CTBC. As expected, the two-iteration MWAVA (equivalently, the ALTMLA) has slightly better performance than the two-iteration WAVA. Also, the two-iteration WAVA and the two-iteration MWAVA are only slightly inferior to the optimal performance for all E_b/N_0 simulated.

In Figure 6, we investigate the effect of iterations on the MWAVA (equivalently, the ALTMLA) for the (2,1,8) CTBC. This figure then shows that the second iteration again produces the largest gain on WER performance. Yet, it requires four iterations to result in a comparable performance to the ML decoder.

The final code to be examined is the (2,1,12) binary CTBC with generator 5135,14477 (octal). The length of the information bits simulated is $L = 48$.

Figure 7 concludes similarly to the previous simulations that the two-iteration MWAVA slightly improves the two-iteration WAVA in performance. However, for the (2,1,12) CTBC, the WERs of the two-iteration WAVA and the two-iteration MWAVA are not as close to the ML performance as those obtained from the previous CTBCs of smaller constraint length.

In Figure 8, we again investigate the effect of iterations on the MWAVA (equivalently, the ALTMLA) for the (2,1,12) CTBC code. Analogously, we observe from this figure that the second iteration

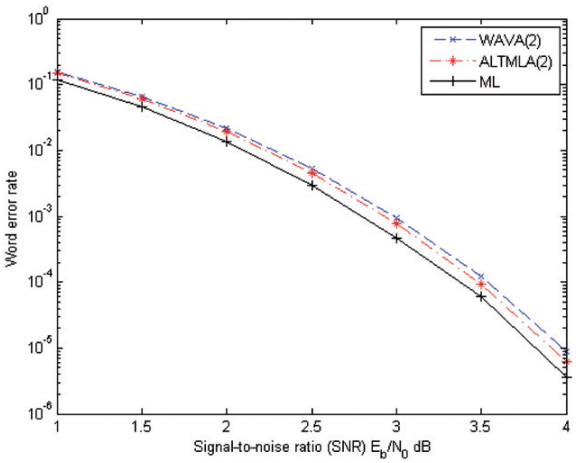

Figure 7. The WERs of the two-iteration WAVA (WAVA(2)), the two-iteration MWAVA (MWAVA(2)), and the ML decoders for the (2,1,12) code with generator 5135,14477.

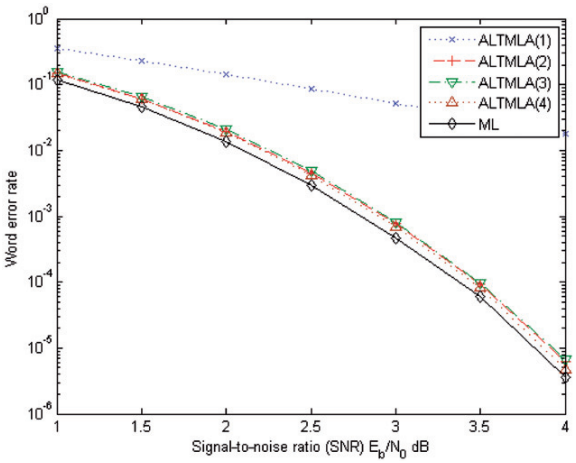

Figure 8. The WERs of the t-iteration MWAVA (MWAVA(t)) and the ML decoders for the (2,1,12) CTBC with generator 5135,14477.

produces the largest gain on WER performance. However, a comparable performance to the ML decoder cannot be achieved by multiple iterations for the (2,1,12) CTBC code.

4. Conclusions

Even though the ALTMLA proposed in Krishnan and Shankar (2006) is seemingly different from the WAVA given in Shao *et al*. (2003), we have proved that the latter can be made equivalent in performance to the former with a proper modification. As a consequence, the new modification can retain the better performance of the ALTMLA without the drawback of its large memory consumption.

Acknowledgments

This work was supported by the National Science Council of Taiwan, R.O.C., under grants NSC 96-2628-E-305-001-MY3.

Nomenclature

$\mathcal{C}$	(n, l, m) convolutional tail-biting code
$\mathcal{C}_s$	super code of $\mathcal{C}$
E	signal energy per channel bit
E_b	signal energy per information bit
L	length of information bits
m	memory order of CTBC
n	number of output bits per information bit of CTBC
N	$n \cdot L$
N_0	signal-sided noise power per hertz
$\Pr(\cdot\|\cdot)$	channel transition probability
r	N-tuple received vector
v	N-tuple binary codeword of $\mathcal{C}$
ϕ_i	log-likelihood ratio at position i ($\phi_j \triangleq \ln[\Pr(r_j\|0)/\Pr(r_j\|1)]$)

References

Chen, T.T. and Tsai, S.H., 2008. Reduced-complexity wrap-around Viterbi algorithm for decoding tail-biting convolutional codes. *In: 14th European wireless conference*, Prague, Czech Republic, 1–6.

Cox, R.V. and Sundberg, C.E.W., 1994. An efficient adaptive circular Viterbi algorithm for decoding generalized tailbiting convolutional codes. *IEEE transactions on vehicular technology*, 43 (11), 57–68.

Han, Y.S., Hartmann, C.R.P., and Chen, C.C., 1993. Efficient priority-first search maximum-likelihood soft-decision decoding of linear block codes. *IEEE transactions on information theory*, 39 (5), 1514–1523.

Krishnan, K.M. and Shankar, P., 2006. Approximate linear time ML decoding on tail-biting trellises in two rounds. *IEEE international symposium on information theory*. Seattle, WA, USA, 2245–2249.

Pai, H.T., *et al*., 2008. Low-complexity ML decoding for convolutional tail-biting codes. *IEEE communications letters*, 13 (12), 883–885.

Shankar, P., *et al*., 2006. *Efficient convergent maximum likelihood decoding on tail-biting*. Available from: http://arxiv.org/abs/cs.IT/0601023 [Accessed 10 February 2006].

Shao, R.Y., Lin, S., and Fossorier, M.P.C., 2003. Two decoding algorithms for tailbiting codes. *IEEE transactions on communications*, 51 (10), 1658–1665.

Wang, Q. and Bhargava, V.K., 1989. An efficient maximum likelihood decoding algorithm for generalized tail biting. *IEEE transactions on communications*, 37 (8), 875–879.

Zhang, M., *et al*., 2009. Research on an-based decode of tail-biting convolutional codes and their performance analyses used in LTE system. *International forum on information technology and applications*. China: Chengdu, 303–306.