International Symposium on Information Theory and its Applications, ISITA2008
Auckland, New Zealand, 7-10, December, 2008

Priority-First Search Decoding for Convolutional Tail-biting Codes

Yunghsiang S. Han[†], Ting-Yi Wu[†], Hung-Ta Pai[†], Po-Ning Chen[‡], and Shin-Lin Shieh[‡††]

[†] Graduate Institute of
Communication Engineering
National Taipei University
Taipei, Taiwan, R.O.C.
E-mail: yshan@mail.ntpu.edu.tw

[‡] Dept. of Communications Engineering
National Chiao-Tung University
Hsin Chu City, Taiwan 30056, R.O.C.

[††] Sunplus mMobile Inc.
HsinChu Science Park
Taiwan 300, R.O.C.

Abstract

Due to rapid interest on the applications of convolutional tail-biting to communication systems, several suboptimal algorithms have been proposed to achieve near-optimal Word error rate (WER) performances with circular Viterbi decoding approach. Among them, the wrap-around Viterbi algorithm (WAVA) proposed in [1] is the one with least decoding complexity. Very recently, a maximum likelihood (ML) decoding algorithm has been proposed in [2]. The scheme has two phases. The Viterbi algorithm is applied to the trellis of the convolutional tail-biting code and the information obtained in the first phase is used by algorithm A*, which is performed to all subtrellises, in the second phase. In this work, a new two-phase ML decoding algorithm is proposed. From the simulation results for the $(2, 1, 12)$ convolutional tail-biting code, the proposed algorithm has 16 times less average decoding complexity in the second phase when compared to the one using algorithm A* and 15123 times less than that of the WAVA, respectively, when $\mathrm{SNR_b} = 4$ dB.

1. Introduction

Convolutional codes have been widely used to provide effective error control capability in digital communications. Usually, in a convolutional encoder, zeros are appended to the sequence of information bits to clear the contents of shift register. These extra zero tail-bits can also enhance the error protection capability of the codewords. For sufficiently long information sequence, the loss in code rates due to these zero tail-bits is negligible. However, when the information sequence is short, evident code rate loss may be induced by adding these extra zero tail-bits.

Several methods have been proposed to resolve the code rate lose of the aforementioned convolutional zero-tail codes, such as Direct Truncation, Puncturing [3], and Tail-biting [1, 4]. In particular, the convolutional tail-biting codes can directly overcome the loss on the code rate, and induce less performance degradation. In comparison, the convolutional zero-tail encoder always starts from and ends at the same all-zero state, while the convolutional tail-biting one only ensures that the initial state and the final state are the same but may vary with the status of the input data. As a result that all possible states can be the initial state for the convolutional tail-biting codes, the decoding complexity drastically increases.

The decoding of a convolutional tail-biting code is performed on its trellis, over which a path always ends at the same state as the initial one, and is called a *tail-biting path*. There is a one-to-one correspondence between a tail-biting path and a codeword. For this reason, "tail-biting paths" and "codewords" will be used interchangeably in this work.

If the number of initial states (equivalently, final states) of the convolutional tail-biting code is N_s, the trellis is composed of N_s subtrellises with the same initial and final state. By following similar naming convention, these subtrellises are called the *tail-biting subtrellises*, or simply subtrellises, and will be denoted by T_i for the ith subtrellis.

In literature, several suboptimal decoding algorithms for the convolutional tail-biting codes have been proposed [1, 4–6]. Among them, the wrap-around

This work was supported by the *National Science Council* of Taiwan, R.O.C., under the projects of NSC 96-2628-E-305-001 and Aiming for the Top University and Elite Research Center Development Plan.

Viterbi algorithm (WAVA) is the one with the least decoding complexity [1]. Conceptually, the WAVA applies the Viterbi algorithm (VA) onto the wrap-around "super" trellis of the convolutional tail-biting code. This wrap-around super trellis is constituted by several trellises that are connected one after another in a tandem fashion. Usually, at most four trellises are sufficient to obtain a near-optimal performance.

A straightforward optimal decoding algorithm for the convolutional tail-biting codes is to perform the Viterbi algorithm on all of the tail-biting subtrellises; however, such approach may be impractical due to its high computational complexity. Very recently, a maximum likelihood (ML) decoding algorithm of practical decoding complexity has been proposed [2]. The scheme has two phases. In the first phase, the Viterbi algorithm is applied to the trellis of the convolutional tail-biting code to obtain the trellis information. Based upon the trellis information, the algorithm A* is then performed on all subtrellises in the second phase to yield the ML decision. It has been shown that the decoding complexity has been reduced from N_s VA trials to equivalently 1.3 VA trials without sacrificing the optimality in performance.

In this work, a refinement on the two-phase decoding of the convolutional tail-biting code in [2] is proposed. Specifically, a new ML decoding metric and a new evaluation function are designed and used in the first and second phases, respectively. Moreover, the backward Viterbi algorithm that performs the Viterbi search backwardly on the trellis, and the priority-first search algorithm that is a simplified version of the Algorithm A* are respectively applied instead for phases one and two. Simulations for the $(2, 1, 12)$ convolutional tail-biting code showed that at $\mathrm{SNR_b} = 4$ dB, our refined two-phase decoding algorithm has 16 times less average decoding complexity at phase two than the one in [2], and is even 15123 times less complex than the WAVA.

2. Priority-First Search ML Decoding of Convolutional Tail-Biting Codes

Let $\mathscr{C}$ be an $(n, 1, m)$ convolutional tail-biting code of L information bits, where n is the number of output bits per information bit, and m is the memory order. Hence, the trellis of $\mathscr{C}$ has $N_s = 2^m$ states at each level, and is of $L + 1$ levels. As aforementioned, the corresponding tail-biting paths for codewords of $\mathscr{C}$ should constrain on the same initial and final state. By relaxing such constraint, we denote by $\mathscr{C}_s$ the super code of $\mathscr{C}$, which consists of all paths that may end at a final state different from the initial state.

Denote by $\boldsymbol{v} \triangleq (v_0, v_1, \ldots, v_{N-1}) \in \{0,1\}^N$ the binary codeword of $\mathscr{C}$, where $N = nL$. Define the hard-decision sequence $\boldsymbol{y} = (y_0, y_1, \ldots, y_{N-1})$ corresponding to the received vector $\boldsymbol{r} = (r_0, r_1, \ldots, r_{N-1})$ as

$$y_j \triangleq \begin{cases} 1, & \text{if } \phi_j < 0; \\ 0, & \text{otherwise,} \end{cases}$$

where $\phi_j \triangleq \ln[\Pr(r_j|0)/\Pr(r_j|1)]$. Then, it can be derived by the Wagner rule that the ML decoding output $\boldsymbol{v}^*$ for received vector $\boldsymbol{r}$ satisfies

$$\sum_{j=0}^{N-1} (v_j^* \oplus y_j)|\phi_j| \le \sum_{j=0}^{N-1} (v_j \oplus y_j)|\phi_j| \quad \text{for all } \boldsymbol{v} \in \mathscr{C},$$

where "$\oplus$" is the exclusive-or operation. We thereby define a new metric for the path in a subtrellis as follows.

Definition 1 *For a path with zero-one labels $\boldsymbol{x}_{(\ell n-1)}^{(i)} = (x_0^{(i)}, x_1^{(i)}, \ldots, x_{\ell n-1}^{(i)})$, which ends at level ℓ in subtrellis T_i, define the metric associated with it as*

$$M\left(\boldsymbol{x}_{(\ell n-1)}^{(i)}\right) \triangleq \sum_{j=0}^{\ell n-1} M(x_j^{(i)}),$$

where $M(x_j^{(i)}) \triangleq (y_j \oplus x_j^{(i)})|\phi_j|$ is the bit metric.

The metrics for those paths not belonging to any subtrellis T_i, where $1 \le i \le N_s$, can be similarly defined.

Our proposed algorithm also has two decoding phases as similar the scheme given in [2]. Yet, different from their scheme, the VA in our first phase is applied to the trellis of the convolutional tail-biting code in a backward fashion using the metric just defined. Thus, we will have a set of N_s survivors ending at the initial states rather than the final states after phase one. Notably, these survivor paths correspond to codewords in $\mathscr{C}_s$, but may not be codewords in $\mathscr{C}$. We also retain the metric of the survivor ending at state s_ℓ of level ℓ, obtained in the first phase, and will denote it by $c(s_\ell)$. We summarize the backward VA as follows.

⟨Phase One: The backward Viterbi Algorithm⟩

Step 1. Associate zero initial metric with every zero-length path containing only the final state s_L at level L in the trellis, and let $c(s_L) = 0$. Set $\ell = L$.

Step 2. Decrease ℓ by one. Compute the path metric for all paths entering a state at level ℓ by

adding the bit metrics of the branch[1] *entering that state to the metric of the connecting survivor at level* $\ell+1$. *For each state* s_ℓ *at level* ℓ, *keep the entering path with the least metric, and delete the remaining, and let* $c(s_\ell)$ *equal the least metric.*

Step 3. If $\ell = 0$, *the algorithm stops; otherwise, go to Step 2.*

In the second phase, instead of operating on the entire trellis with respect to the super code $\mathscr{C}_s$, the decoding of the priority-first search decoding only operates on tail-biting subtrellises. Thus, the output of the second phase will always be a codeword in $\mathscr{C}$. In addition, the priority-first search is applied forwardly as in [2].[2] For each path with zero-one labels $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ over subtrellis T_i, a new evaluation function f is associated with it as follows:

$$f(\boldsymbol{x}^{(i)}_{(\ell n-1)}) = g(\boldsymbol{x}^{(i)}_{(\ell n-1)}) + h(\boldsymbol{x}^{(i)}_{(\ell n-1)}), \qquad (1)$$

where

$$g(\boldsymbol{x}^{(i)}_{(\ell n-1)}) = g(\boldsymbol{x}^{(i)}_{((\ell-1)n-1)}) + \sum_{j=(\ell-1)n}^{\ell n-1} M(x^{(i)}_j) \qquad (2)$$

with initial value $g(\boldsymbol{x}^{(i)}_{(-1)}) = 0$,

$$h(\boldsymbol{x}^{(i)}_{(\ell n-1)}) = c(s_\ell)$$

and s_ℓ is the state that path $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ ends at.

It is easy to see that $f(\boldsymbol{x}^{(i)}_{N-1}) = g(\boldsymbol{x}^{(i)}_{N-1})$ since $h(\boldsymbol{x}^{(i)}_{N-1}) = c(s_L) = 0$, where s_L is the state that path $\boldsymbol{x}^{(i)}_{(N-1)}$ ends at. Hence, the tail-biting path with the minimum f-function value is exactly the one with the minimum ML metric.

Then, equipped with an *Open Stack* for paths visited thus far by the priority-first search algorithm, and a *Close Table* for starting and ending states and ending level of the paths that have ever been on top of the Open Stack, we summarize the priority-first search algorithm on subtrellises in the following.

⟨Phase Two: The Priority-First Search Algorithm⟩[3]

Step 1. Sort all survivors obtained after phase one according to ascending order of their metrics. If the survivor with the least metric is also a tail-biting path (that starts and ends at the same state), then output it as the final ML decision, and the algorithm stops.

Step 2. Set ρ *equal to the least metric among all survivors that are also tail-biting paths, if such exists; otherwise, set* $\rho = \infty$. *Delete all survivors whose metrics are no less than* ρ.

Step 3. Load into the Open Stack every zero-length path whose initial state s_0 *at level* 0 *coincides with any of the ending states of the remaining survivors. Sort these zero-length paths in the Open Stack according to ascending order of their* f*-function values.*

Step 4. If the Open Stack is empty, output the survivor with metric ρ *as the final ML decision, and the algorithm stops.*

Step 5. If the top path in the Open Stack reaches level L *in its subtrellis, output the path as the final ML decision, and the algorithm stops.*

Step 6. If the information of the starting and ending states and ending level of the top path has been recorded in the Close Table, discard the top path from the Open Stack, and go to Step 4; otherwise, record the paired information of the starting and ending states and ending level of the top path in the Close Table.

Step 7. Compute the f*-function values of the successors of the top path, and delete the top path from the Open Stack. If the* f*-function value of any successor is equal to or greater than* ρ, *just delete it.*

Step 8. Insert the remaining successor paths into the Open Stack, and re-order the Open Stack according to ascending f*-function values. Go to Step 4.*

3. Optimality of the Priority-First Search Algorithm Guided by f in (1)

In this section, we provide the proof that the priority-first search algorithm proposed in the previous section is optimal. We begin with the lemma that will be essential in the proof of the main result.

Lemma 1 f *is a non-decreasing function along any tail-biting path on each subtrellis, that is,*

$$f\left(\boldsymbol{x}^{(i)}_{(\ell n-1)}\right) \le f\left(\boldsymbol{x}^{(i)}_{((\ell+1)n-1)}\right),$$

[1] Each branch metric is the sum of n bit metrics (cf. (2)).

[2] Our decoding algorithm can also employ the conventional forward VA in the first phase, and perform the priority-first search backwardly in the second phase.

[3] There is an index for each path what subtrellis it belongs to.

where path $\boldsymbol{x}^{(i)}_{((\ell+1)n-1)}$ *is the immediate successor of path* $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ *over subtrellis* T_i.

Proof: Since, according to the backward VA in the first phase, $c(s_\ell)$ is the minimum metric among all paths that start from any final state, but end at state s_ℓ of level ℓ, we have:

$$c(s_\ell) \leq c(s_{\ell+1}) + \sum_{j=\ell n}^{(\ell+1)n-1} M(x_j^{(i)}),$$

where $s_{\ell+1}$ and s_ℓ are respectively the states that paths $\boldsymbol{x}^{(i)}_{((\ell+1)n-1)}$ and $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ end at. Hence,

$$\begin{aligned} f\left(\boldsymbol{x}^{(i)}_{((\ell+1)n-1)}\right) &= g\left(\boldsymbol{x}^{(i)}_{((\ell+1)n-1)}\right) + h\left(\boldsymbol{x}^{(i)}_{((\ell+1)n-1)}\right) \\ &= g\left(\boldsymbol{x}^{(i)}_{(\ell n-1)}\right) + \sum_{j=\ell n}^{(\ell+1)n-1} M(x_j^{(i)}) \\ &\quad + c(s_{\ell+1}) \\ &\geq g\left(\boldsymbol{x}^{(i)}_{(\ell n-1)}\right) + c(s_\ell) \\ &= f\left(\boldsymbol{x}^{(i)}_{(\ell n-1)}\right). \end{aligned}$$

With the availability of Lemma 1, the optimality of our proposed algorithm can be proved as follows.

Step 1 follows from the fact that the best path in $\mathscr{C}_s$, when it also belongs to $\mathscr{C}$, is the best path in its subcode $\mathscr{C}$; hence, the tail-biting survivor with the least metric shall be the ML decision.

Step 2 is based on two observations. The first one is that ρ is an upper bound on the metric of the final ML path. Secondly, from Lemma 1, the initial f-function value of a subtrellis is no larger than the f-function value of any later-found tail-biting path. Therefore, as the f-function values of the tail-biting paths are exactly their metrics, all tail-biting paths whose initial f-function value is no less than ρ cannot be the final ML path, and can be deleted.

It remains to show that if the Open stack is empty or the top path reaches level L, the algorithm will output the ML path. It suffices to prove that Steps 6 and 7 will not delete the ML path.

Suppose that at Step 6, the paired information of the starting and ending states and ending level of the new top path $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ has been recorded in the Close Table at some previous time due to path $\hat{\boldsymbol{x}}^{(i)}_{(\ell n-1)}$. Since path $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ must be the offspring of a path $\boldsymbol{x}^{(i)}_{(\bar{\ell} n-1)}$ that once coexisted with $\hat{\boldsymbol{x}}^{(i)}_{(\ell n-1)}$ in the Open Stack at the time $\hat{\boldsymbol{x}}^{(i)}_{(\ell n-1)}$ was on top of the Open Stack, where $\bar{\ell} < \ell$, we have

$$f\left(\boldsymbol{x}^{(i)}_{(\ell n-1)}\right) \geq f\left(\boldsymbol{x}^{(i)}_{(\bar{\ell} n-1)}\right) \geq f\left(\hat{\boldsymbol{x}}^{(i)}_{(\ell n-1)}\right).$$

Notably, the first inequality follows from Lemma 1, and the second inequality is valid because the top path in the Open Stack always carries the minimum f-function value. As a result, the minimum-metric tail-biting path generated from path $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ will always have an equal or larger metric than the minimum-metric tail-biting path generated from path $\hat{\boldsymbol{x}}^{(i)}_{(\ell n-1)}$. The deletion of path $\boldsymbol{x}^{(i)}_{(\ell n-1)}$ accordingly will not eliminate the ML tail-biting path.

For Step 7, we argue that the f-function value associated with a path is no greater than that of any path generated from it. Hence, extending any path with f-function value already larger than or equal to ρ will only lead to a tail-biting path with metric no smaller than ρ. Since we have already kept one tail-biting path with metric ρ in Step 2, the deletion of a path with f-function value $\geq \rho$ will not affect the optimality of our algorithm.

Finally, when a top path reaches level L, its f-function value shall be the smallest among the offsprings of all paths coexisted in the Open Stack. It is therefore the ML tail-biting path.

We close this section by remarking that since the convolutional tail-biting codes are "circular" in nature, their decoding can in fact start from any position that aligns with the branch margins in the received vector. In other words, our two-phase decoding algorithm applies by simply re-numbering the branch-aligned position that the decoding process starts from as the new level 0. The selection of the position corresponding to the new level 0 is as follows. Find a branch-aligned position in the received vector such that the sum of the reliabilities of the immediate subsequent λ branches is the largest, where the reliability at position j is $|\phi_j| = |\ln[\Pr(r_j|0)/\Pr(r_j|1)]|$ for $0 \leq j \leq N-1$. This step is performed before the first phase of decoding, and its complexity is almost negligible in comparison with the decoding complexity of the two-phase algorithm.

4. Simulations over AWGN Channels

In this section, we investigate by simulations the computational effort and the word error rate (WER) of the proposed decoding algorithm over the additive white Gaussian noise (AWGN) channels. We assume that the codeword is BPSK-modulated. Hence, the received vector is given by

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

Table 1: Comparison of average (ave) and maximum (max) numbers of branch metric computations in the second phase

SNR_b	1 dB		2 dB		3 dB		4 dB	
algorithm	ave	max	ave	max	ave	max	ave	max
WAVA	196608	196608	196608	196608	196608	196608	196608	196608
A*	40441	2338532	8513	1871687	931	1301153	211	371576
PFSA(12)	5853	393021	1228	257729	114	245329	13	57262

Table 2: Comparison of average (ave) and maximum (max) numbers of memory required in the second phase

SNR_b	1 dB		2 dB		3 dB		4 dB	
algorithm	ave	max	ave	max	ave	max	ave	max
WAVA	4096	4096	4096	4096	4096	4096	4096	4096
A*	19994	1864371	4842	1641087	1357	1138133	879	339133
PFSA(12)	4350	339807	1830	237298	946	223638	598	20735

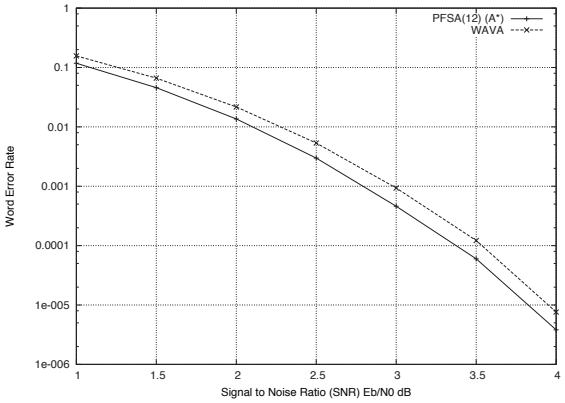

Figure 1: The word error rates (WERs) of PFSA(12), A*, and WAVA.

for $0 \leq j \leq N-1$, where $\mathcal{E}$ 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 signal-to-noise ratio (SNR) for the channel is therefore $\mathrm{SNR} \triangleq \mathcal{E}/N_0$. In order to account for the code redundancy for different code rates, we will use the SNR per information bit, i.e.,

$$\mathrm{SNR_b} = \frac{N\mathcal{E}/L}{N_0} = n\left(\frac{\mathcal{E}}{N_0}\right),$$

in the following discussions.

We now turn to the empirical examination of the average decoding complexity. The $(2, 1, 12)$ binary convolutional tail-biting code with generator $5135, 14477$ (octal) is considered. The length of the information bits used in our simulations is 48. For ease of referring them, we will respectively abbreviate the proposed algorithm in phase two with $\lambda = 12$ and the scheme given in [2] as the PFSA(12) and the A* in the sequel. For all simulations, at least 30 word errors have been reported to ensure that there is no bias on the simulation results.

In Figure 1, we compare the WERs of the PFSA(12) with those obtained by the A*, as well as the WAVA given in [1] with two iterations. Since both the PFSA(12) and the A* are ML decoders, it is reasonable that they yield the same WER. Also noted from Figure 1 is that the WAVA has about 0.2 dB coding gain loss on WER performance.

In Table 1, we compare the average and maximum computational efforts of the PFSA(12) with those of the A* and the WAVA in the second phase. For a fair comparison, we count only the samples for which the computations of branch metrics, i.e., the second term in (2), are necessary.[4] Since the WAVA searches for the entire trellis in phase two, its number of branch metrics computed is a constant for all SNR_b. It can then be noted that when $\mathrm{SNR_b} = 4$ dB, the maximum number of computations that the PFSA(12) requires is 2/7 of that of the WAVA. Much more saving (about 15000 times less) can be observed when the average number of computations is concerned.

Likewise, the PFSA(12) has much smaller average and maximum computational complexities than the A* for all SNR_b. For example, the average and maximum numbers of computational efforts of the PFSA(12) are respectively 16 times and 6 times less than those of the A* at $\mathrm{SNR_b} = 4$ dB. A more striking result is that

[4]The proposed recursive implementation of the Algorithm A* in [2] has a merit that no branch metric computation is required when the successor path has the same f-function value as its predecessor. By this reason, the complexities of the PFSA(12) in Table 1 also excludes the computations of those branch metrics that equate the f-function values of the successor and its immediate predecessor.

when $\mathrm{SNR_b} \geq 4$ dB, the average number of metric values evaluated by the PFSA(12) reduces to only two digits.

In Table 2, we compare the average and maximum memory required by the PFSA(12) with those by the A* and the WAVA in the second phase. According to the algorithm, the memory required by the WAVA equals the number of paths stored at each level of the trellis, and hence, is a constant value of 4096 for the $(2, 1, 12)$ code. The memories demanded by the A* and the PFSA(12), however, depend on the SNR, and are the numbers of paths stored in stacks. It can then be noted that when $\mathrm{SNR_b} = 4$ dB, the average number of memory that the PFSA(12) requires is only 1/7 of that required by the WAVA. As anticipated, the WAVA outperforms the PFSA(12) at the required maximum memory index.

Table 2 also indicates that the PFSA(12) has much smaller average and maximum memory complexities than the A* for all SNR_b. As an example, the maximum number of paths stored in a stack for the PFSA(12) is ten times less than that for the A* at $\mathrm{SNR_b} = 4$ dB.

5. Concluding Remarks and Future Work

With the help of a new evaluation function, the ML tail-biting path can be located in the proposed two-phase decoding procedure with a much less decoding complexity. Analysis of the decoding complexity of the proposed algorithm will be an interesting future work. A fixed decoding complexity variant of the PFSA, as contrary to the varying complexity of the current version, will also be of interest from the practical standpoint.

References

[1] R. Y. Shao, S. Lin, and M. P. C. Fossorier, "Two decoding algorithm for tailbiting codes," *IEEE Trans. Commun.*, vol. COM-51, no. 10, pp. 1658 – 1665, October 2003.

[2] P. Shankar, P. N. A. Kumar, K. Sasidharan, B. S. Rajan, and A. S. Madhu, "Efficient convergent maximum likelihood decoding on tail-biting," available at http://arxiv.org/abs/cs.IT/0601023.

[3] Y. P. E. Wang and R. Ramesh, "To bite or not to bite - a study of tail bits versus tail-biting," *in Proceeding of IEEE Personal, Indor and Mobile Radio Communications*, vol. 2, pp. 317 – 321, October 1996.

[4] Q. Wang and V. K. Bhargava, "An efficient maximum likelihood decoding algorithm for generalized tail biting," *IEEE Trans. Commun.*, vol. COM-37, no. 8, pp. 875 – 879, 1989.

[5] H. H. MA and J. K. WOLF, "On tail biting convolutional codes," *IEEE Trans. Commun.*, vol. COM-34, no. 2, pp. 104 – 111, February 1986.

[6] R. V. Cox and C. E. W. Sundberg, "An efficient adaptive circular viterbi algorithm for decoding generalized tailbiting convolutional codes," *IEEE Trans. Veh. Techol*, vol. 43, no. 11, pp. 57 – 68, February 1994.