# TAIL BITING TRELLIS REPRESENTATION OF CODES: DECODING AND CONSTRUCTION

Technical Report

to

NASA

Goddard Space Flight Center
Greenbelt, Maryland 20771

Grant Numbers: NAG 5-8414

Report Number: 99-003

Principal Investigator: Shu Lin

Department of Electrical Engineering
University of Hawaii at Manoa
2540 Dole Street, Holmes Hall 483
Honolulu, Hawaii 96822

December 16, 1999

# TAIL BITING TRELLIS REPRESENTATION OF CODES: DECODING AND CONSTRUCTION

**Rose Y. Shao, Shu Lin**
**and**
**Marc Fossorier**

**December 16, 1999**

# Tail Biting Trellis Representation of Codes: Decoding and Construction *

Rose Y. Shao, Shu Lin and Marc Fossorier
Department of Electrical Engineering
University of Hawaii at Manoa
Honolulu, Hawaii 96822, U.S.A.
Email:yshao,slin@spectra.eng.hawaii.edu

**Abstract**

This paper presents two new iterative algorithms for decoding linear codes based on their tail biting trellises, one is unidirectional and the other is bidirectional. Both algorithms are computationally efficient and achieves virtually optimum error performance with a small number of decoding iterations. They outperform all the previous suboptimal decoding algorithms. The bidirectional algorithm also reduces decoding delay. Also presented in the paper is a method for constructing tail biting trellises for linear block codes.

*This research was supported by NSF under Grants NCR 94-15374, CCR 97-32959, CCR 98-14054 and NASA under Grant NAG 5-8414.

## 1 Introduction

There are two types of trellis representation of codes, conventional trellis representation and tail biting trellis representation. In conventional trellis representation of a code [1-3], the code trellis has one starting state and one ending state, and the starting state and the ending state are the same state. The paths connecting the starting state and the ending state give all the codewords (or code sequences) of the code. In tail biting trellis representation of a code, the code trellis has multiple starting states and multiple ending states [4-6]. Each starting state has a unique corresponding ending state, and they are the same state. A path in a tail biting trellis is a valid codeword if and only if it starts from a state and ends at the same state. Such a path is called a **tail biting path**.

Tail biting trellis representation was first proposed by Solomon and van Tilborg [4] in 1979 for terminating the code trellis of a convolutional code without code rate loss. This tail biting trellis representation of a convolutional code was later generalized by Ma and Wolf [5] in 1986. Tail biting of a convolutional code results in a block code. Figure 1 depicts an 8-section tail biting trellis with 4 starting states and 4 ending states for a rate-1/2 convolutional code of memory order 2 with generator sequences $\mathbf{g}_1 = (1, 0, 1)$ and $\mathbf{g}_2 = (1, 1, 1)$ [1]. This tail biting convolutional code is a (16,8) linear block code. In [4], a link between quasi-cyclic codes and tail biting convolutional codes was also established. This link led to the most recent generalization of tail biting trellis representations of linear block codes by Calderbank, Forney and Vardy [6].

Tail biting trellis representation of a linear block code can significantly reduce the overall state and branch complexities of its conventional trellis representation, especially for long codes. Figures 2 and 3 depict a minimal 8-section conventional trellis and a minimal 8-section tail biting trellis for the (8,4,4) Reed-Muller (RM) code, respectively. The conventional trellis for this code has a total of 34 states and a total of 44 branches. The maximum state complexity(maximum number of states) is 8 which occurs at section boundary locations 3 and 5. The tail biting trellis for this code has two starting states, two ending states, a total of 30 states and a total of 40 branches. The maximum state complexity is 4. For this short code, the reduction in trellis state and branch complexities is small. Consider the (24,12,8) Golay code. This code has a minimal 12-section conventional trellis with a total number of 1,066 states and a total number of 1,960 2-bit branches. The maximum state complexity is 256 [7, 8]. However, this code has a 12-section regular tail biting trellis with 16 starting states and 16 ending states [6]. The number of states at each section boundary location is 16. Figure 6 shows the first 4 sections of the tail biting trellis for the code. It has a total of 208 states and a total of 384 2-bit branches. The maximum state complexity is 16. We see that tail biting trellis representation of the (24,12,8) Golay code results in a significant reduction in both state and branch complexities compared with the conventional trellis representation. The reduction in state and branch complexities of a code trellis results in a reduction of decoding complexity of a trellis-based decoding algorithm. General structure and construction of tail biting trellises for linear block codes have been further investigated lately [9-12].

As pointed out earlier, a path in a tail biting trellis is a valid codeword if and only if its starting and ending states are the same. Since there are multiple starting states, a transmitted codeword can start from any of its starting states which is unknown to the receiver. Therefore, any tail biting trellis-based decoding algorithm must have a mechanism to estimate the starting state for each transmission of a codeword. A simple minded maximum likelihood decoding (MLD) algorithm is to treat a tail biting trellis with $M$ starting states and $M$ ending states as a union of $M$ subtrellises, each subtrellis consists of the paths that connect a starting state and its corresponding ending state [4]. Each subtrellis is then a conventional trellis. Process each subtrellis with the conventional Viterbi algorithm and determine its survivor. This results in $M$ survivors, one for each subtrellis. Then compare these $M$ survivors. The one with the best metric (the largest correlation metric or the smallest Euclidean distance metric) is then the most likely (ML) codeword and the decoded codeword.

This simple minded MLD algorithm requires to process $M$ subtrellises independently and then compares $M$ survivors to make a decoding decision. If $M$ is large, the computational complexity can be very large. If a single decoder is used, it also results in long decoding delay. If $M$ decoders are used to process the $M$ subtrellises in parallel, this reduces the decoding delay but increases the hardware complexity. To reduce the decoding complexity, several suboptimal iterative Viterbi-type algorithms for decoding codes based on their tail biting trellises have been proposed [5,13-18].

In this paper, two new iterative algorithms for decoding codes based on their tail biting trellises are presented, one is an unidirectional wrap-around algorithm and the other is a bidirectional wrap-around algorithm. In these algorithms, both cumulative state metric and path metric are used to determine surviving paths and the final survivor. During the decoding iteration process, the decoder continues updating the best surviving path until certain termination condition is met. Both algorithms are computationally efficient and achieves virtually optimum error performance with a small number of iterations, say 2 to 4, for almost the entire range of signal-to-noise ratios (SNR) for many codes being simulated. They outperform all the previous suboptimal decoding algorithms. The bidirectional algorithm also reduces decoding delay. Also presented in this paper is a simple method for constructing tail biting trellises for linear block codes.

## 2 Preliminary and Review

This section first provides some needed background information regarding tail biting trellises for linear codes and then gives a brief review of existing algorithms for decoding codes based on their tail biting trellises.

Let $T$ be an $L$-section tail biting trellis for a binary linear code $C$ with section boundary locations indexed by $0, 1, 2, \cdots, L$. These boundary locations also serve as time indices [2, 3, 7, 8]. For $0 \leq t \leq L$, let

$$\Sigma_t(C) = \{s_{t,1}, s_{t,2}, \cdots, s_{t,q_t}\} \tag{1}$$

denote the state space of the tail biting trellis $T$ at boundary location-$t$ (or time-$t$), where $q_t = |\Sigma_t(C)|$. Then $\Sigma_0(C)$ and $\Sigma_L(C)$ are the starting and ending (or final) state spaces of $T$, respectively, and

$$\Sigma_0(C) = \Sigma_L(C), \tag{2}$$

i.e., $\Sigma_0(C)$ and $\Sigma_L(C)$ consists of the same set of states. Hence, $q_0 = q_L$. For $1 \leq i \leq q_0$ (or $q_L$), $s_{0,i}$ and $s_{L,i}$ are the same state. A tail biting trellis for a convolutional code is time-invariant and the state spaces at all the boundary locations are the same, i.e.,

$$\Sigma_0(C) = \Sigma_1(C) = \cdots = \Sigma_L(C). \tag{3}$$

Figure 1 displays the time-invariant property of a tail biting trellis for a 4-state convolutional code. However, a tail biting trellis for a linear block code is in general time-varying, i.e., for $0 \leq i, j \leq L$ and $i \neq j, \Sigma_i(C)$ and $\Sigma_j(C)$ may not be the same. For example, the minimal 8-section tail biting trellis for the (8,4,4) RM code shown in Figure 3 is a time-varying tail biting trellis.

The tail biting trellis $T$ may be viewed as a union of $q_0$ subtrellises [4, 12]. Each subtrellis consists of those tail biting paths in $T$ that connect a state $s_{0,i}$ at boundary location-0 to the same state $s_{L,i}$ at boundary location-$L$, i.e., the starting and ending states of each subtrellis are the same. For example, the tail biting trellis shown in Figure 1 consists of 4 subtrellises, and the tail biting trellis for the (8,4,4) RM code shown in Figure 3 consists of two subtrellises. The tail biting paths of the subtrellis $T_0$ of $T$ which contains the all-zero path actually form a linear subcode $C_0$ of $C$ [2, 11, 12]. As a result, the other subtrellises of $T$ are the trellises for the cosets of $C_0$ in $C$. Therefore, the $q_0$ subtrellises of $T$ are structurally identical (or isomorphic). All these subtrellises share a common part from certain boundary location-$t_1$ to certain boundary location-$t_2$ with $t_1 \leq t_2$ [2, 12]. For example, the 4 subtrellises of the tail biting trellis shown in Figure 1 share a common part from boundary location-2 to boundary location-6, i.e., they share 4 common trellis sections. The two subtrellises of the tail biting trellis for the (8,4,4) RM code shown in Figure 3 share 4 common trellis sections from boundary location-2 to boundary location-6.

Representing a linear code $C$ by a tail biting trellis $T$ with $q_0$ starting states and $q_L$ ending states with $q_0 = q_L$, a transmitted codeword may start from any of the $q_0$ starting states. If a tail biting trellis based algorithm is used for decoding $C$, the decoding algorithm must first estimate the starting and ending states of the transmitted codeword and then determines the most likely codeword that connects the estimated starting and ending states. The simple minded tail biting trellis based MLD described in Section 1 requires to process $q_0$ subtrellises with the Viterbi algorithm independently. For simplicity, we call this Viterbi-type tail biting trellis based MLD as the VTMLD algorithm. Define the Viterbi processing of a trellis of $L$ section as a **Viterbi trial** (VT) and the processing of one trellis section (including all the addition, comparison and selection operations) as a **Viterbi update** (VU) [16]. Then the VTMLD algorithm requires a total of $q_0$ VT's and a total of $q_0 L$ VU's to make a decoding decision. For large $q_0$ and $L$, the VTMLD algorithm results in a large computational complexity.

To overcome the computational complexity problem of the VTMLD algorithm, several Viterbi-type suboptimal decoding algorithms have been proposed [13-18]. All these decoding algorithms are iterative in nature and they process the tail biting trellis $T$ of a code with the conventional Viterbi algorithm repeatedly until certain stopping conditions are met or a preset maximum number of decoding iterations is reached. The differences between these suboptimal decoding algorithms are their starting conditions, termination conditions, methods for estimating the starting state, and methods for selecting the decoded codeword (or sequences). All these suboptimal decoding algorithms require much less VU's (or VT's) than the VTMLD algorithm at the expense of some performance degradation.

The first and the simplest suboptimal algorithm for decoding a code based on its tail biting trellis $T$ is the Bar-David algorithm [5] which estimates the starting state based on a probabilistic approach. It consists of the following steps:

1) Choose an arbitrary starting state from $\Sigma_0(C)$.

2) Process the tail biting trellis $T$ with the Viterbi algorithm to find all the survivors originated from the selected starting state to all the ending states in $\Sigma_L(C)$.

3) Select the survivor with the best metric among all the survivors as the **winning path**.

4) Check if the winning path is a tail biting path. If so, stop the decoding process and output the winning path as the decoded codeword. Otherwise, go to the next step.

5) Use the ending state of the winning path as a new starting state.

6) Check if this starting state has been used before. If so, go to Step 1), otherwise, go to step 2).

The above process continues until a winning tail biting path is found or the number of iterations reaches $q_0$. For the later case, the decoder simply outputs the best winning path at that time. If the winning path is not a tail biting path, it is not a valid codeword. The Bar-David algorithm is indeed very simple but has a significant performance degradation compared to the VTMLD algorithm, especially for low to medium SNR's. Even though, for large SNR, the average number of decoding iterations (or VT's) required is small. However, for small SNR, the number of decoding iterations required approaches $q_0$, which results in a large number of computations.

An improvement of Bar-David algorithm is the two-step algorithm devised by Ma and Wolf [5]. The first step of this algorithm is to obtain an ordered list of the $q_0$ starting states using an algebraic method called "continued fractions." The second step of this algorithm is to perform Viterbi trials using each entry in the list as the starting state. At the end of each Viterbi trial, check whether the winning path is a tail biting path. If so, stop the decoding process and output the winning path as the decoded codeword; otherwise, continue the iteration process with the next state on the list as the starting state. This two-step algorithm provides some improvement in performance and decoding complexity over the Bar-David algorithm, but it still requires a large number of Viterbi trials for small SNR.

To further improve the performance of Bar-David and Ma-Wolf's two-step algorithms and reduce their decoding complexities, Wang and Bhargava [13] proposed another iterative decoding algorithm. This algorithm starts processing the tail biting trellis $T$ with the Viterbi algorithm from all the states in $\Sigma_0(C)$ with the same initial state metrics. Every state in $\Sigma_0(C)$ is equally likely regarded as the starting state of the transmitted codeword. When the processing reaches the end of the tail biting trellis $T$, tests are performed. Based on the results of these tests, the decoder either terminates the decoding process and outputs the best tail biting path that has been found or reduces the starting state space by eliminating those unlikely starting states and updates the candidate tail biting path (the candidate for the decoded codeword) that is stored in a buffer. For the latter case, the decoder processes the tail biting trellis $T$ again with the states in the reduced starting state space as the starting states with the same state metrics. This iterative process continues until either the most likely tail biting path is found or the reduced starting state space is empty. Wang-Bhargava algorithm is asymptotically optimal, which improves the error performance of Bar-David and Ma-Wolf algorithms with reduced decoding complexity. However, this algorithm is quite complex with a number of test conditions and a variable workload. Even though, it is asymptotically optimum for high channel SNR, however, it performs relatively poorly compared to the VTMLD algorithm for low to medium SNR's. One reason for its poor performance for low to medium SNR's is that the algorithm does not transfer all available soft information from trial to trial (or from one iteration to the next iteration). For example, at each decoding iteration, all the starting states are assigned the same starting metrics. However, the metrics of the states in the reduced starting state space computed at the end of the previous Viterbi trial can be used as the starting state metrics for the current trial.

This would allow the real starting state to gain momentum faster than the other states in the reduced state space. As a result, the iteration decoding process may converge faster.

To improve the performance of the Wang-Bhargava algorithm, several algorithms were devised based on continuous Viterbi processing of a trellis which is a multifold repetition (or concatenation) of the tail biting trellis $T$ [14-16]. All the available soft information is transferred from one Viterbi trial to the next Viterbi trial. The state metrics at the end of one trial are used as the starting state metrics for the next trial. Among these algorithms, the most efficient one is the circular Viterbi (CV) algorithm devised by Cox and Sundberg [16]. Let $T^* = T \circ T \circ \cdots$ denote the multifold repetition of $T$ where $\circ$ denotes the concatenation operation. The CV algorithm simply processes the extended trellis continuously with the Viterbi algorithm. Let $t$ and $t_0$ be two nonnegative integers such that $0 \leq t_0 \leq L$ and

$$t = t_0 \ (modulo\ L).$$

Then

$$\Sigma_t(C) = \Sigma_{t_0}(C). \tag{4}$$

At boundary location-$t$ of $T^*$, let $M(t,i)$ denote the metric of state $s_{t,i}$ (this is simply the cumulative metric of the survivor that terminates at state $s_{t,i}$). The CV algorithm consists of the following steps:

1) Start from all the states in $\Sigma_0(C)$ with the same initial state metrics.

2) Apply the Viterbi algorithm continuously to process $T^*$ section by section. At level-$t$ (or boundary location-$t$), record the metric of the surviving path entering each state in $\Sigma_t(C)$ and save the information labeling bits of the surviving branches in a word, called the decision word for section-$t$.

3) After processing the first $L$ sections of $T^*$, the decoder starts making decoding decision.

4) For some chosen small $\epsilon$ and some constant $\alpha$, if

$$|M(t,i) - M(t-L,i) - \alpha| < \epsilon \tag{5}$$

for $1 \leq i \leq q_t$, stop the decoding process and output the winning path between level-$(t-L)$ and level-$t$ (a reordering of the decoded information bits may be needed). Otherwise, go to Step 2).

5) Repeat Steps 2 and 4 until the stopping condition of (5) is met or a maximum number of iterations $I_{max}$ is reached.

With this algorithm, a large storage is required to store all the state metrics over a span of $L+1$ sections. To simplify this algorithm, the following stopping rule was suggested [16]:

4$^*$) Check the decision words separated by $L$ sections. If $m$ consecutive decision words are the same as their predecessors, the decoding process stops. Otherwise, go to Step 2).

Simulation results show that the two CV algorithms with stopping conditions 4 and 4$^*$, respectively, perform almost equally well in terms of error performance and computational complexity. Simulation results also show that the CV algorithm performs better than the Wang-Bhargava algorithm for small to medium SNR's with significant reduction in decoding computational complexity. However, for large SNR, the Wang-Bhargava algorithm outperforms the CV algorithm. Both algorithms still have a significant performance degradation compared to the VTMLD algorithm.

The other two continuous Viterbi decoding algorithms [14, 15] based on processing $T^*$ have similar features as the CV algorithm.

## 3 A Wrap-Around Viterbi Algorithm

This section presents a new iterative algorithm for decoding linear codes based on their tail biting trellises. This algorithm is also devised based on processing a tail biting trellis $T$

repeatedly in a continuous manner with the Viterbi algorithm. Each Viterbi processing (or Viterbi trial) of $T$ is called an iteration. The algorithm consists of a sequence of decoding iterations. The available soft information is transferred from one iteration to the next iteration. The algorithm starts the decoding process from all the states in $\Sigma_0(C)$ with the same initial state metrics at the first iteration. At the end of each iteration, if no decoding decision is made, the metrics of the ending states in $\Sigma_L(C)$ are used as the starting state metrics of the next iteration. This is called **wrap-around**. The wrap-around process results in a continuous Viterbi decoding over the tail biting trellis $T$. The wrap-around processing of $T$ continues until certain stopping conditions are met. The algorithm uses cumulative path metric during the continuous wrap-around process to select the survivor into each state. However, at the end of each iteration, it uses the metric difference between a starting state in $\Sigma_0(C)$ and an ending state in $\Sigma_L(C)$ (if they are connected by a survivor) to determine the $L$-branch winning path for decoding decision. The decoding process stops if the winning path is a tail biting path in $T$ (a codeword in $C$) or a maximum number of iterations is reached. Decoding decision is made at the end of each iteration. If no decoding decision in made at the end of an iteration, a candidate for the final decoded codeword is updated and stored. For simplicity, this iterative decoding algorithm is referred to as a **wrap-around Viterbi** (WA-V) algorithm.

## 3.1 The algorithm

Assume that the correlation metric is used as the decoding metric. Again let $T$ be an $L$-section tail biting trellis for a linear code $C$ with starting state space $\Sigma_0(C)$, ending state space $\Sigma_L(C)$ and $\Sigma_0(C) = \Sigma_L(C)$. For $0 \leq t \leq L$, let $\Sigma_t(C) = (s_{t,1}, s_{t,2}, \cdots, s_{t,q_t})$ denote state space of $T$ at section boundary location-$t$. Since the WA-V algorithm processes $T$ repeatedly with the Viterbi algorithm in a continuous manner, the metric of a state in $T$ is updated during each iteration. A survivor terminated at a state in $T$ at boundary location-$t$

during the $i$-th decoding iteration is a surviving path originated from a starting state in $\Sigma_0(C)$ at the beginning of the first decoding iteration. For $i \geq 1$, $0 \leq t \leq L$ and $1 \leq k \leq q_t$, let $C_{t,k}^{(i)}$ denote the metric of state $s_{t,k}$ at the $i$-th decoding iteration which is defined as the cumulative path metric of the survivor of the $i$-th iteration terminated at state $s_{t,k}$. At the end of the $i$-th iteration, let $\mathbf{p}^{(i)}$ denote an $L$-branch surviving path that connects the starting state $s_{0,h}$ in $\Sigma_0(C)$ and the ending state $s_{L,j}$ in $\Sigma_L(C)$. The path metric of $\mathbf{p}^{(i)}$ denoted $\Delta_{\mathbf{p}^{(i)}}$, is defined as the following difference between the metric $C_{L,j}^{(i)}$ of state $s_{L,j}$ and the metric $C_{0,h}^{(i)}$ of state $s_{0,h}$:

$$\Delta_{\mathbf{p}^{(i)}} \triangleq C_{L,j}^{(i)} - C_{0,h}^{(i)}. \tag{6}$$

The $L$-branch path with the largest path metric among all the $q_L(= q_0)$ surviving $L$-branch paths in $T$ at the end of the $i$-th decoding iteration is called the **best path**, denoted $\mathbf{p}_{best}^{(i)}$. Note that the best surviving path is not necessarily a tail biting path in $T$ (or a codeword in $C$). If $\mathbf{p}_{best}^{(i)}$ is a tail biting path, decoding stops and the decoder outputs $\mathbf{p}_{best}^{(i)}$ as the decoded codeword. If $\mathbf{p}_{best}^{(i)}$ is not a tail biting path, the decoder finds the best surviving tail biting path, denoted $\mathbf{p}_{T,best}^{(i)}$, among all the surviving tail biting paths (if any) at the end of the $i$-th iteration. Use $\mathbf{p}_{best}^{(i)}$ and $\mathbf{p}_{T,best}^{(i)}$ to update the currently stored best path $\mathbf{p}_{best}$ and best tail biting path $\mathbf{p}_{T,best}$. Whenever $\mathbf{p}_{T,best}^{(i)} = \mathbf{p}_{best}^{(i)}$, decoding stops. When a maximum number of decoding iterations is reached, the decoder outputs $\mathbf{p}_{T,best}$ if it exists, otherwise, the decoder outputs $\mathbf{p}_{best}$. Let $\Delta_{best}$ and $\Delta_{T,best}$ denote the metrics of $\mathbf{p}_{best}$ and $\mathbf{p}_{T,best}$, respectively. Then the decoder needs to update and store $(\mathbf{p}_{best}, \Delta_{best})$ and $(\mathbf{p}_{T,best}, \Delta_{T,best})$ from iteration to iteration.

At the end of the $i$-th decoding iteration, if no decoding decision is made, the state metric vector

$$\mathbf{C}_L^{(i)} \triangleq (C_{L,1}^{(i)}, C_{L,2}^{(i)}, \cdots, C_{L,q_L}^{(i)}) \tag{7}$$

is used as the initial state metric vector for the $(i+1)$-th iteration, i.e., $\mathbf{C}_0^{(i+1)} = \mathbf{C}_L^{(i)}$. The WA-V algorithm can be formulated as follows:

1) Initialization - start from all the states in $\Sigma_0(C)$ with the same initial state metric.

2) For $1 \leq i \leq I_{max}$, execute the $i$-th decoding iteration with $\mathbf{C}_0^{(i)} = \mathbf{C}_L^{(i-1)}$ ($\mathbf{C}_L^{(0)}$ is the intial state metric vector). At section boundary location-$L$ of $T$, compare the metrics of all the $L$-branch surviving paths that terminate at the ending states in $\Sigma_L(C)$. Select the best path $\mathbf{p}_{best}^{(i)}$ as the winning path of the $i$-th iteration. If $\mathbf{p}_{best}^{(i)}$ is a tail biting path, go to Step 4). Otherwise, update $(\mathbf{p}_{best}, \Delta_{best})$ and go to Step 3).

3) Find the best surviving tail biting path $\mathbf{p}_{T,best}^{(i)}$ (if any). Compare the metrics of $\mathbf{p}_{T,best}^{(i)}$ and $\mathbf{p}_{T,best}$ and update $(\mathbf{p}_{T,best}, \Delta_{\mathbf{p}_{T,best}})$. Check whether $i < I_{max}$. If so, set $i = i + 1$ and go to Step 2). Otherwise, go to Step 5).

4) Output $\mathbf{p}_{best}^{(i)}$ as the decoded codeword.

5) Output $\mathbf{p}_{T,best}$ as the decoded codeword if it exists. Otherwise, output $\mathbf{p}_{best}$ as the decoded word.

Figure 7 depicts the flowchart of the above WA-V algorithm.

At the beginning of the WA-V decoding, each initial state is treated equally with the same starting metric due to our ignorance of the real starting state of the transmitted codeword. After several iterations, different initial states may have different state metrics. A path stemming from an initial state with larger metric has a better chance to become the winning path than the ones stemming from the initial states with smaller metrics. The larger metric an initial state has, the more likely it is the real starting state of the transmitted codeword.

## 3.2 Analysis

Let $C_s$ denote the set of all paths in $T$ that connect the states in $\Sigma_0(C)$ and the states in $\Sigma_L(C)$ (i.e., from any state in $\Sigma_0(C)$ to any state in $\Sigma_L(C)$). Then $C_s$ is a super code of $C$. The WA-V algorithm is characterized by a number of theorems given below.

**Theorem 3.1** *If all the starting states in $\Sigma_0(C)$ at the beginning of the i-th iteration of the WA-V algorithm have the same initial state metrics, then the best L-branch surviving path $\mathbf{p}_{best}^{(i)}$ at the end of the i-th iteration has the largest metric among all the codewords in $C_s$ for a given received sequence $\mathbf{r}$. If $\mathbf{p}_{best}^{(i)}$ is a tail biting path in $T$, $\mathbf{p}_{best}^{(i)}$ is the most likely codeword in $C$ with respect to $\mathbf{r}$.*

*Proof:* First we note that at the end of the $i$-th iteration, each survivor terminated at a state in $\Sigma_L(C)$ has a larger cumulative path metric than its competitors at any state boundary location-$t$ of $T$ for $0 \leq t \leq L$. If all the states in $\Sigma_0(C)$ have the same initial state metric at the beginning of the $i$-th iteration, it follows from the definition of path metric of an $L$-branch path $\mathbf{p}^{(i)}$ given by (6) that the best $L$-branch surviving path $\mathbf{p}_{best}^{(i)}$ must have the largest metric among all the paths in $T$ (or all the codewords in $C_s$) with respect to the received sequence $\mathbf{r}$. If $\mathbf{p}_{best}^{(i)}$ is a tail biting path, it is a codeword in $C$ and hence the most likely codeword in $C$ for the given received sequence $\mathbf{r}$. This proves the theorem. $\Delta\Delta$.

Since all the states in $\Sigma_0(C)$ have the same initial state metrics at the beginning of the first iteration, a direct consequence of Theorem 3.1 is Corollary 3.1.

**Corollary 3.1** *At the first iteration of the WA-V algorithm, the best L-branch surviving path $\mathbf{p}_{best}^{(1)}$ is the most likely codeword in $C_s$. If $\mathbf{p}_{best}^{(1)}$ is a tail biting path, it is the most likely codeword in $C$.*

From Corollary 3.1, we see that if decoding is done at the end of the first iteration of WA-V algorithm, the decoder output is the most likely codeword in $C$.

**Theorem 3.2** *For $i > 1$, if the states in $\Sigma_0(C)$ do no have the same initial metrics at the beginning of the i-th iteration, the best surviving L-branch path $\mathbf{p}_{best}^{(i)}$ may not be the most likely codeword in $C_s$ and hence it may not be the most likely codeword in $C$ even if it is a tail biting path in $T$.*

*Proof:* Recall that the selection of the survivor at a state is based on the cumulative path metric up to that state. From (6), we have

$$C_{L,j}^{(i)} = C_{0,h}^{(i)} + \Delta_{\mathbf{p}^{(i)}}. \tag{8}$$

It follows from (8) that for $i > 1$, a path with a larger path metric $\Delta_{\mathbf{p}^{(i)}}$ can be discarded by the WA-V algorithm in favor of a competitor which originates from a state in $\Sigma_0(C)$ with a larger state metric, if the difference between their path metrics is not enough to compensate for the difference between their initial state metrics. In this case, the best surviving $L$-branch path $\mathbf{p}_{best}^{(i)}$ is not the most likely codeword in $C_s$ and hence it may not be the most likely codeword in $C$ even if it is a tail biting path. $\Delta\Delta$.

Theorem 3.2 says that for $i > 1$, if the decoding is made at the end of the $i$-th iteration, the decoder output may not be the most likely codeword in $C$. This implies that the WA-V algorithm is not an optimal MLD algorithm. However, simulation results show that this algorithm achieves near optimum error performance with only 2 to 4 iterations for many codes (block or convolutional). It outperforms all the existing iterative algorithms described in Section 2 with smaller computational complexity.

## 3.3 Performance and complexity

The WA-V algorithm has been applied to decode various block and convolutional codes based on their tail biting trellises. Simulations results for the AWGN channel show that this algorithm achieves near optimum MLD error performance for all the codes being decoded with a maximum number $I_{max}$ of 2 to 4 iterations. For convolutional codes with $L \geq 6m$ where $m$ is the memory order, $I_{max} = 2$ is enough for the WA-V algorithm to achieve near optimum error performance. The WA-V algorithm requires much less computational complexity than the optimum VTMLD algorithm. Simulation results for several codes are given in the following to demonstrate the effectiveness of the WA-V algorithm.

Figure 8 compares the error performance of the WA-V algorithm with that of the other existing algorithms described in Section 2 for the (64,32) tail biting code obtained by truncating the rate-1/2 (2,1,7) convolutional code of memory order $m = 7$ with generator polynomials (712,476)(or generator sequences $\mathbf{g}_1 = (1,1,1,0,0,1,0,1)$ and $\mathbf{g}_2 = (1,0,0,1,1,1,1,1)$) [1]. The tail biting trellis for this code has $L = 32$ sections and 128 states at each section boundary location. Since for the other existing algorithms, simulations were performed over a binary symmetric channel (BSC), we compare the WA-V algorthm with these algorithms based on a BSC. From Figure 8, we see that the WA-V algorithm outperforms all the existing algorithms. It achieves virtually optimum error performance with a maximum number of iterations set to 4 and outperforms the Wang-Bhargava and CV algorithms by 0.2 dB at BER's $10^{-3}$ and $10^{-4}$, respectively. It outperforms the Bar-David and Ma-Wolf algorithms by 1.0 dB and 0.8 dB at BER=$10^{-3}$, respectively. Table 1 gives the decoding complexities of various algorithms in terms of average number of Viterbi trials required for various SNR's. It shows that the WA-V algorithm requires the least number of Viterbi trials, 1.3 on average.

The rest of simulation results for the WA-V algorithm given in the following are obtained based on an AWGN channel. Figure 9 shows the bit error performance of the WA-V algorithm for decoding the (128,64) tail biting code obtained by truncating the rate-1/2 (2,1,6) convolutional code of memory order $m = 6$ generated by generator polynomials (554,744) [1]. The tail biting trellis of this code has $L = 64$ sections and 64 states. We see that the WA-V algorithm achieves virtually optimum error performance with the maximum number $I_{max}$ of iterations set to 2. The average number of iterations required is 1.33 for SNR's over the range 1.0 to 3.0 dB. The figure also shows that the algorithm converges very fast.

In [6], a 16-state tail biting trellis with 12 sections for the (24,12) Golay code has been constructed (see Figure 6). Figures 10 and 11 show the bit and frame error performance of the WA-V algorithm for decoding the (24,12) Golay code based on its 16-state tail biting trellis. Again, both figures show that the WA-V algorithm achieves virtually optimum error performance with the maximum number $I_{max}$ of iterations set to 4. With $I_{max}$ set to 2, there is only 0.2 dB performance degradation compared with MLD at BER=$10^{-4}$. Figure

12 shows the average number of Viterbi updates versus SNR's. For $I_{max} = 4$, it requires an average of 17 Viterbi updates to complete the decoding process at SNR=3 dB.

A 64-state 12-section tail biting trellis for the (24,12) Golay code can be constructed by truncating the 64-state trellis of the rate-1/2 $(2,1,6)$ convolutional code generated by generator polynomials (414,730) with $L = 12$ [17, 18]. Figure 13 shows the bit error performance of the (24,12) Golay code decoded with the WA-V algorithm based on this 64-state tail biting trellis. Also included in this figure is the bit error performance of a modified CV algorithm devised by Anderson and Hladik [17, 18], called the optimal circular Viterbi algorithm (O-CVA) (This algorithm has not been publically reported and the simulation results are provided by the authors.) We see that the WA-V algorithm with both $I_{max} = 2$ and $I_{max} = 4$ outperforms the O-CVA. The O-CVA requires 112 Viterbi updates to complete the decoding for each SNR. However, the average numbers of Viterbi updates required by the WA-V algorithm with $I_{max} = 4$ are 31.67 at 1 dB SNR, 25.65 at 2 dB SNR, 20.45 at 3 dB SNR and 16.2 at 4 dB SNR, respectively. For $I_{max} = 4$, the maximum number of Viterbi updates required is 48. Therefore, the WA-V algorithm outperforms the O-CVA with less computational complexity.

## 4 An Iterative Bidirectional Viterbi Decoding Algorithm

All the algorithms, including the WA-V algorithm presented in the last section, for decoding linear codes based on their tail biting trellises are unidirectional decoding algorithms which process a tail biting trellis in one direction. It is possible to devise a bidirectional decoding algorithm which processes a tail biting trellis from both ends simultaneously using two decoders. If the bidirectional process is carried out properly, not only optimum or near optimum error performance can be achieved but also the decoding delay can be shortened. In this section, such a bidirectional algorithm for decoding linear codes based on their tail biting trellises is presented and is called an **iterative bidirectional Viterbi** (IBD-V) algorithm.

Consider an $(n, k)$ code $C$ with an $L$-section tail biting trellis $T$. Let $\mathbf{v} = (v_1, v_2, \cdots, v_n)$ be the codeword to be transmitted. Before its transmission, it is permuted into $\mathbf{v}' = (v_1, v_n, v_2, v_{n-1}, \cdots)$. Let $\mathbf{r} = (r_1, r_n, r_2, r_{n-1}, \cdots)$ be the received sequence. Before decoding, $\mathbf{r}$ is decomposed into two sequences, $\mathbf{r}^{(1)} = (r_1, r_2, \cdots, r_n)$ and $\mathbf{r}^{(2)} = (r_n, r_{n-1}, \cdots, r_1)$. The IBD-V algorithm processes the tail biting trellis $T$ simultaneously from both ends (right and left) with two Viterbi decoders based on the received sequences $\mathbf{r}^{(1)}$ and $\mathbf{r}^{(2)}$, respectively. The decoders that process the tail biting trellis $T$ from the left and right ends are called the left and right Viterbi decoders, respectively. The state space $\Sigma_0(C)$ at boundary location-0 is the starting state space for the left Viterbi decoder and the state space $\Sigma_L(C)$ at boundary location-$L$ is the starting state space for the right Viterbi decoder. The decoding process that the two decoders start from opposite ends, work through the trellis and reach to the other ends is called a decoding iteration. The IBD-V algorithm consists of a sequence of iterations. At the end of each iteration, the two decoders wrap around the tail biting trellis T and then continue the decoding process. The metrics of states at each end of $T$ are used as the starting state metrics for the next iteration. At the beginning of the first iteration, all the starting states in $\Sigma_0(C)$ and $\Sigma_L(C)$ have the same initial state metrics. During each decoding iteration, the two decoders start to make decoding decision jointly as soon as they meet. Iteration process continues until the most likely tail biting path is found or a preset maximum number $I_{max}$ of iterations is reached.

Suppose the two decoders are executing the $i$-th iteration. For $0 \leq t \leq L$ and $1 \leq k \leq q_t$, let $C_{t,k}^{l,(i)}$ and $C_{t,k}^{r,(i)}$ denote the cumulative metrics of the state $s_{t,k}$ at the boundary location-$t$ computed by the left and right Viterbi decoders at the $i$-th iteration, respectively. There are two surviving paths terminating at the state $s_{t,k}$, one from the left end of the trellis $T$ and the other from the right end of $T$, denoted $\mathbf{p}_{t,k}^{l,(i)}$ and $\mathbf{p}_{t,k}^{r,(i)}$, and are called left and right surviving paths, respectively. The $L$-branch path obtained by concatenating the left surviving path $\mathbf{p}_{t,k}^{l,(i)}$ and the right surviving path $\mathbf{p}_{t,k}^{r,(i)}$ is called a **composite path** (CP) through $s_{t,k}$, denoted $\mathbf{p}_{t,k}^{c,(i)}$. A CP that has the same state at both ends is called a **composite tail biting path** (CTP). Let $s_{0,h}$ and $s_{L,j}$ denote the starting states of the left surviving path $\mathbf{p}_{t,k}^{l,(i)}$ and the right surviving path $\mathbf{p}_{t,k}^{r,(i)}$, respectively. Define the metric differences

$$\Delta_{t,k}^{l,(i)} \triangleq C_{t,k}^{l,(i)} - C_{0,h}^{l,(i)}, \tag{9}$$

$$\Delta_{t,k}^{r,(i)} \triangleq C_{t,k}^{r,(i)} - C_{L,j}^{r,(i)}, \tag{10}$$

as the **path metric gains** of the left and right surviving paths, $\mathbf{p}_{t,k}^{l,(i)}$ and $\mathbf{p}_{t,k}^{r,(i)}$, respectively. Then the path metric of the CP through the state $s_{t,k}$ is defined as

$$\Delta_{t,k}^{c,(i)} \triangleq \Delta_{t,k}^{l,(i)} + \Delta_{t,k}^{r,(i)}. \tag{11}$$

As soon as a state $s_{t,k}$ has been visited by the two Viterbi decoders from opposite directions during the $i$-th iteration, the metric of the CP through this state is computed. The CP at boundary location-$t$ that has the largest metric is called the **best** CP of the $i$-th iteration at the boundary location-$t$, denoted $\mathbf{p}_{t,best}^{c,(i)}$. Let $\Delta_{t,best}^{c,(i)}$ denote its metric. Let $\mathbf{p}_{best}^{c}$ and $\Delta_{best}^{c}$ denote the best composite path and its metric that have been found and stored up to the moment that $\mathbf{p}_{t,best}^{c,(i)}$ is found. If decoding decision is not made at this moment, $(\mathbf{p}_{t,best}^{c,(i)}, \Delta_{t,best}^{c,(i)})$ is then used to update $(\mathbf{p}_{best}^{c}, \Delta_{best}^{c})$. The pair $(\mathbf{p}_{best}^{c}, \Delta_{best}^{c})$ are updaded each time when each of the two decoders has completed processing one section of the tail biting trellis $T$ in opposite directions. If there are CTP's at the boundary location-$t$, the CTP that has the largest metric is called the **best** CTP of the $i$-th iteration at boundary location-$t$, denoted $\mathbf{p}_{t,best}^{T,(i)}$. Let $\Delta_{t,best}^{T,(i)}$ denote its metric. Let $\mathbf{p}_{best}^{T}$ and $\Delta_{best}^{T}$ denote the best CTP and its metric that have been found and stored up to the current moment. If decoding decision is not made at this moment, the pair $(\mathbf{p}_{t,best}^{T,(i)}, \Delta_{t,best}^{T,(i)})$ is used to update $(\mathbf{p}_{best}^{T}, \Delta_{best}^{T})$. The pair $(\mathbf{p}_{best}^{T}, \Delta_{best}^{T})$ is updated continuously as the two decoders move in opposite directions one section at a time.

The two decoders process the tail biting trellis $T$ continuously section by section. When they reach the opposite ends of $T$, they wrap around and start the next iteration. The two decoders collaborate to make a decoding decision. A decoding decision is made when the best CP found by the two decoders is a tail biting path in $T$ or when a preset maximum number of iterations, $I_{max}$, is reached. For the latter case, the decoder outputs $\mathbf{p}^T_{best}$ (if any), otherwise outputs $\mathbf{p}^c_{best}$ (this is not a codeword in $C$).

## 4.1 The algorithm

During each decoding iteration, the two decoders execute a procedure to find the best CP and the best CPT at each boundary location of $T$ to make a decoding decision. The procedure to be executed is called the **Find-Best($t$) procedure**. We use $flag = 1$ to indicate that the best CP is found to be a tail biting path, and $flag = 0$ otherwise. The **Find-Best($t$) procedure** at the $i$-th iteration consists of the following steps:

a) Compute $\Delta^{c,(i)}_{t,k}$ for all the states $s_{t,k}$ in $\Sigma_t(C)$.

b) Find the best composite path $\mathbf{p}^{c,(i)}_{t,best}$ at location-$t$.

c) If $\mathbf{p}^{c,(i)}_{t,best}$ is a tail biting path, set $flag = 1$ and output $\mathbf{p}^{c,(i)}_{t,best}$ as the decoded codeword. Otherwise, update $(\mathbf{p}^c_{best}, \Delta^c_{best})$ that are stored in the memory and go to step d.

d) Find the best composite tail biting path, $\mathbf{p}^{T,(i)}_{t,best}$ (if any), update the pair $(\mathbf{p}^T_{best}, \Delta^T_{best})$ that are stored in the memory.

Set the maximum number of iteration to $I_{max}$. The IBD-V algorithm consists of the following steps:

A. Set $i = 1.flag = 0$, $\mathbf{C}^{l,(1)}_0 = \mathbf{C}^{r,(1)}_L = (-\infty, -\infty, \cdots, -\infty)$ (initial state metrics at both ends of $T$), and $\Delta^c_{best} = \Delta^T_{best} = -\infty$.

B. If $i < I_{max}$ and $flag = 0$, execute the following steps:

1) Perform the Viterbi decoding process from both ends of the trellis.

2) When the two decoders meet at boundary location-$t_0$, set $t_l = t_r = t_0$.

3) Call and execute the **Find-Best($t_0$) procedure**. If $flag = 1$, stop the decoding process. Otherwise, go to Steps 4 and 5.

4) Set $t_l = t_l + 1$. If $t_l \leq L$, call and execute the **Find-Best($t_l$) procedure**.

5) Set $t_r = t_r + 1$. If $t_r \geq 0$, call and execute the **Find-Best($t_r$) procedure**.

6) Repeat Steps 4 and 5 simultaneously. If $flag$ is set to 1 by either decoder or by both, decoding stops. For the latter case, the two best composite paths found at boundary location-$t_l$ and -$t_r$ are both tail biting paths. If they are not the same, the one with larger metric is chosen as the decoded codeword. If $flag = 0$, update $(\mathbf{p}^c_{best}, \Delta^c_{best})$ and $(\mathbf{p}^T_{best}, \Delta^T_{best})$ and repeat Steps 4 and 5.

7) When the two decoders reach both ends of the trellis, check if $i < I_{max}$. If so, set $i = i + 1, \mathbf{C}_0^{l,(i)} = \mathbf{C}_L^{l,(i-1)}$, $\mathbf{C}_L^{r,(i)} = \mathbf{C}_0^{r,(i-1)}$, go to Step B and start the next iteration. Otherwise, go to Step C.

C. If $\Delta^T_{best} > -\infty$, output $\mathbf{p}^T_{best}$ as the decoded codeword; Otherwise output $\mathbf{p}^c_{best}$.

## 4.2 Analysis

Again, let $C_s$ be the super code of $C$ which consists of the label sequences of all the paths in $T$ as codewords. Assume that transmission is over an AWGN channel. The path in $T$ that has the largest correlation metric with the received sequence $\mathbf{r}$ is the most likely codeword in $C_s$ and is called the **optimum path**, denoted $\mathbf{p}_{opt}$. If $\mathbf{p}_{opt}$ is a tail biting path, it is the most likely codeword in $C$. In the following, we prove several theorems which characterize the IBD-V algorithm.

**Theorem 4.1** *If at the beginning of a decoding iteration of the IBD-V algorithm, all the starting states in $\Sigma_0(C)$ for the left decoder have the same initial state metrics and all the* *starting states in $\Sigma_L(C)$ for the right decoder have the same initial state metrics, the optimum path $\mathbf{p}_{opt}$ will not be discarded by the decoders during the decoding process.*

*Proof:* Since all the starting states at each end of $T$ have the same initial state metrics, it follows from (6) that the path terminating at a state $s_{t,k}$ at boundary location-$t$ which has the largest path metric also has the largest cumulative path metric. A path $\mathbf{p}$ in $T$ can be discarded by either the left decoder or the right decoder or by both. Suppose the optimum path $\mathbf{p}_{opt}$ for a given received sequence is discarded at a state $s_{t,k}$ at boundary location-$t$. If it is discarded by the left decoder, the left survivor into state $s_{t,k}$ from the left has larger path metric than the part of the optimum path $\mathbf{p}_{opt}$ from its starting state in $\Sigma_0(C)$ to state $s_{t,k}$. If it is discarded by the right decoder, the the right survivor into state $s_{t,k}$ from the right has larger path metric than the part of the optimum path $\mathbf{p}_{opt}$ from its starting state in $\Sigma_L(C)$ to state $s_{t,k}$. Either case results in a composite path through state $s_{t,k}$ that has a metric larger than the metric of $\mathbf{p}_{opt}$. This is not possible since $\mathbf{p}_{opt}$ has the largest metric with the received sequence. Therefore, $\mathbf{p}_{opt}$ will not be discarded by either decoder. $\Delta\Delta$.

A direct consequence of Theorem 4.1 is Corollary 4.1.

**Corollary 4.1** *If at the beginning of a decoding iteration of the IBD-V algorithm, all the starting states in $\Sigma_0(C)$ have the same initial state metrics and all the starting states in $\Sigma_L(C)$ have the same initial state metrics, the best composite path at each boundary location of $T$ is the optimum path. If the best composite path is a tail biting path, it is the most likely codeword in $C$ with respect to the received sequence $\mathbf{r}$.*

Note that at the beginning of the first iteration of the IBD-V algorithm, all the starting states in both $\Sigma_0(C)$ and $\Sigma_L(C)$ are set with the same initial state metrics. It follows from Corollary 4.1 that at the first iteration, the best composite path is found as soon as the two decoders meet at the center of the trellis $T$ and this best composite path is the optimal path in $T$. If this optimum path is a tail biting path, then it is the most likely codeword in $C$ and the decoding is optimal.

The next theorem simply says that if the condition of Theorem 4.1 does not hold, the best composite path found at any section boundary is not necessarily the optimum path in $T$ for a given received sequence $\mathbf{r}$. The proof of this theorem is similar to the proof of Theorem 3.2.

**Theorem 4.2** *If at the beginning of a decoding iteration, not all the starting states in $\Sigma_0(C)$ or in $\Sigma_L(C)$ have the same initial state metrics, the best composite path found at any boundary location of $T$ is not necessarily the optimum path $\mathbf{p}_{opt}$ in $T$ for a given received sequence $\mathbf{r}$, and hence it is not necessarily the most likely codeword in $C$ even if it is a tail biting path.*

Theorem 4.2 implies that for $i > 1$, if decoding decision is made during the $i$-th iteration, the decoded codeword is not necessarily the most likely codeword in $C$. Therefore, IBD-V algorithm is not an optimal MLD algorithm. However, simulation results of this algorithm for decoding many codes show that this algorithm achieves virtually optimum error performance with a maximum of only two iterations.

After the first decoding iteration, in general, the starting states at either end of the trellis T do not have the same initial state metrics. As a result, the sets of composite paths at different boundary locations may be different. This fact can be proved readily and is given in the following theorem.

**Theorem 4.3** *At the i-th decoding iteration with $i > 1$, the sets of composite paths at different boundary locations may be different.*

This theorem implies that after the first iteration, the best CP's at different boundary locations of the trellis $T$ may be different and the best CTP's at different boundary locations of $T$ may be different. Therefore, to achieve good error performance, we must keep updating $\mathbf{p}_{best}$ and $\mathbf{p}^T_{best}$ from section to section, because they are candidates for the decoded codeword when the decoding process is terminated at the end of $I_{max}$-th iteration.

Since the trellis is processed by two decoders from both directions and the two decoders start to make decoding decision as soon as they meet at the center of the trellis, the IBD-V algorithm has a shorter decoding delay than an unidirectional iterative algorithm, such as the WA-V algorithm. Furthermore, IBD-V algorithm updates its candidates for the decoded codeword continuously section by section rather than waiting until the end of an iteration. As a result, $\mathbf{p}_{best}^{T}$ is chosen from a larger subset of tail biting paths of $T$ than the WA-V algorithm and hence has better chance to be the most likely codeword in $C$. Therefore, the IBD-V algorithm achieves better error performance than the WA-V algorithm with the same number of iterations and converges faster to the MLD performance.

## 4.3 Performance and complexity

The IBD-V algorithm has been applied to decode several convolutional and linear block codes based on their tail biting trellises. Simulation results show that for all the codes being decoded, the IBD-V algorithm virtually achieves optimum error performance with a maximum of 2 iterations. Results for some of these codes are given in the following to show the effectiveness of the IBD-V algorithm.

Figures 14 and 15 show the bit and frame error performance of the IBD-V algorithm for the (64,32) tail biting code obtained by truncating the rate-1/2 (2,1,7) convolutional code generated by polynomials (712,476). We see that the IBD-V algorithm virtually achieves optimum error performance with a maximum of 2 iterations ($I_{max} = 2$). In the same figures, we see that the WA-V algorithm virtually achieves optimum error performance with a maximum of 4 iterations ($I_{max}$=4). Table 2 gives the percentages of transmitted codewords that are decoded into the most likely codewords with IBD-V and WA-V algorithms for various SNR's. We see that even with SNR=1 dB, the IBD-V algorithm decodes 93% of the transmitted codewords into most likely codewords with $I_{max} = 1$ and 99% with $I_{max} = 2$. The WA-V algorithm achieves almost the same percentages with $I_{max} = 2$ and $I_{max} = 4$, respectively. Table 3 gives the average numbers of Viterbi updates required for the two decoding algorithms with various SNR's. We see that both algorithms requires relatively small number of Viterbi updates to complete the decoding even for small SNR's.

Figure 9 shows the bit error performance of the IBD-V algorithm for the (128,64) tail biting codes obtained by truncating the rate-1/2 (2,1,6) convolutional code generated by polynomials (554, 744). Again, the IBD-V algorithm virtually achieves optimum error performance with a maximum of 2 iterations. In fact, for this long code, WA-V is just as effective as the IBD-V algorithm and it achieves optimum error performance with $I_{max} = 2$. Table 4 gives the average numbers of Viterbi updates required for both algorithms with various SNR's.

Figures 10 and 11 gives the bit and frame error performance of the IBD-V algorithm for the (24,12) Golay code based on the minimal 16-state tail biting trellis constructed in [6]. Again, we see that the IBD-V algorithm achieves optimum error performance with a maximum of 2 iterations. Table 5 gives the average number of Viterbi updates for both IBD-V and WA-V algorithms. We see that the number of Viterbi updates required for decoding this code is small even for low SNR's.

Finally, Table 6 gives the computational complexities for decoding the above codes with the VTMLD, WA-V and IBD-V algorithms, respectively. Also included in this table is a recursive MLD (RMLD) algorithm for decoding codes based on their tail biting trellises. This RMLD is simply a generalization of the RMLD devised in [19] for decoding linear block codes based on their conventional trellises. We see that both WA-V and IBD-V algorithms significantly reduce the computational complexity for these codes. The RMLD algorithm also reduces the computational complexity compared with the VTMLD algorithm.

# 5 General Structure and Construction of Tail Biting Trellises for Linear Block Codes

## 5.1 General structure

The general structure of an $L$-section tail biting trellis $T$ for an $(n, k)$ linear block code $C$ is depicted in Figure 5. Let $\{0, 1, ..., L\}$ denote the set of section boundary locations. Suppose $T$ consists of $2^m$ starting states and $2^m$ ending states. We may view $T$ as a union of $2^m$ isomorphic (or structurally identical) subtrellises which share a common part from boundary location-$t_1$ to boundary location-$t_2$, where $t_1 < t_2$. Each subtrellis consists of those paths in $T$ that connect a state at boundary location-0 to the same state at boundary location-$L$, and it has three parts, the header, the center span and the tail. The center span is shared by every subtrellis. A subtrellis starts from a specific starting state at boundary location-0 and grows until it reaches to the boundary location-$t_1$, it then transverses through the center span until it reaches the boundary location-$t_2$, and finally it starts to collapse until it terminates at a state at boundary location-$L$ that is identical to the starting state. For $0 \leq i < 2^m$, let $T_i$ denote the subtrellis whose starting and ending states are $s_{0,i}$ and $s_{L,i}$, respectively. Assume that $T_0$ contains the all-zero path. Then the paths in $T_0$ form an $(n, k - m)$ linear subcode of $C$, denoted $C_0$, and the paths in any other subtrellis form a coset of $C_0$ in $C$. Let $\mathbf{v} = (v_1, v_2, \cdots, v_n)$ be a codeword in $C$. Since all the subtrellises have the same common span from boundary location-$t_1$ to boundary location-$t_2$, there must be a codeword $\mathbf{w} = (w_1, w_2, ..., w_n)$ in each subtrellis whose components from location-$(t_1 + 1)$ to location-$t_2$ are zeros, i.e., $w_{t_1+1} = w_{t_1+2} = \cdots = w_{t_2} = 0$. For convenience, we call the part of first $t_1$ components of $\mathbf{w}$ the header, the part of last $n - t_2$ components of $\mathbf{w}$ the tail. Adding $\mathbf{w}$ to each path in $T_0$ results in a subtrellis which is isomorphic to $T_0$ and is identical to $T_0$ from boundary location-$t_1$ to boundary location-$t_2$. The header and the tail of this subtrellis are obtained by adding the header and the tail of $\mathbf{w}$ to the header and the tail of

$T_0$, respectively. This subtrellis is the trellis for the coset $\mathbf{w} + C_0$ of $C_0$ and $\mathbf{w}$ is the coset representative.

Although all the subtrellises share a common span from boundary location-$t_1$ to boundary location-$t_2$. Two individual subtrellises may share a longer span starting from boundary location-$i$ to boundary location-$j$ with $0 < i \leq t_1$ and $t_2 \leq j < L$. For $0 < i < j < L$, let $[i, j]$ denote the interval $\{i, i+1, \cdots, j\}$. The zero-span of an $n$-tuple $\mathbf{v} = (v_1, v_2, \cdots, v_n)$ is defined as the largest interval $[i, j]$ such that $v_{i+1} = v_{i+2} = \cdots = v_j = 0$. This definition implies that $v_i = v_{j+1} = 1$. Let $\mathbf{v}$ be a codeword in $C$ but not in $C_0$ whose zero-span is $[i, j]$ with $0 < i \leq t_1$ and $t_2 \leq j < L$. It is clear that $[t_1, t_2] \subseteq [i, j]$. Let $T_0(\mathbf{v})$ denote the subtrellis for the coset $\mathbf{v} + C_0$ obtained by adding $\mathbf{v}$ to every path in $T_0$. Then $T_0(\mathbf{v})$ and $T_0$ have a common span from boundary location-$i$ to boundary location-$j$. Let $\mathbf{v}$ and $\mathbf{w}$ be codewords in two different cosets of the partition $C/C_0$. Let $[i_1, j_1]$ and $[i_2, j_2]$ be the zero-spans of $\mathbf{v}$ and $\mathbf{w}$, respectively. Let $[i_3, j_3] = [i_1, j_1] \cap [i_2, j_2]$. Then the co-subtrellises, $T_0(\mathbf{v})$ and $T_0(\mathbf{w})$, are isomorphic and have a common span from boundary location-$i_3$ to boundary location-$j_3$.

## 5.2 Construction

Based on structure analysis given above, a simple method for constructing tail biting trellises for linear block code can be devised. For $0 < t_1 < t_2 < L$, let $C(t_1, t_2)$ denote the set of codewords in $C$ which satisfy the following conditions: (1) each nonzero codeword $\mathbf{v}$ in $C(t_1, t_2)$ has zero components from location-$(t_1 + 1)$ to location-$t_2$, i.e., $v_{t_1+1} = v_{t_1+2} = \cdots = v_{t_2} = 0$, and (2) the part of first $t_1$ components of $\mathbf{v}$ contains at least one nonzero component and the part of last $n - t_2$ components of $\mathbf{v}$ contains at least one nonzero component. Then $C(t_1, t_2)$ is a linear subcode of $C$. The zero-span of each codeword in $C(t_1, t_2)$ contains $[t_1, t_2]$ as a subinterval. Let $m$ be the dimension of $C(t_1, t_2)$. There exists an $(n, k-m)$ linear subcode $C_0$ in $C$ such that $C$ is the direct sum of $C_0$ and $C(t_1, t_2)$. Let $C/C_0$ denote the partition of $C$ modulo $C_0$. Then the vectors in $C(t_1, t_2)$ can be used as the coset representatives of the coset in $C/C_0$. $C(t_1, t_2)$ is called the coset space of $C/C_0$.

Let $T_0$ be the minimal conventional (one starting state and one ending state) bit-level trellis for $C_0$ [2]. Form all the co-trellises $T_0(\mathbf{v})$ of $T_0$ with $\mathbf{v} \in C(t_1, t_2)$. All these co-trellises have a common span from boundary location-$t_1$ to boundary location-$t_2$. Putting all these co-trellises together and sharing maximum common spans between them, we obtain a tail biting trellis with $2^m$ starting states and $2^m$ ending states. The overall (state and branch) complexity of this tail biting trellis depends on the length of common span of the co-trellises, the choice of the boundary locations, $t_1$ and $t_2$, of the common span. These parameters should be chosen to minimize the trellis complexity. If the minimum distance of $C$ is $d$, then the condition $n - t_2 + t_1 \geq d$ must be hold.

To facilitate the construction of a tail biting trellis for a linear $(n, k)$ block code $C$, we put its generator matrix G in **tail biting trellis oriented form (TBTOF)**, called the TBTO generator (TBTOG) matrix. The span of an $n$-tuple is defined as the smallest interval $[a, b]$ which contains all the nonzero components. The active span of $\mathbf{v}$ is defined as the interval $[a, b-1]$. For a vector $\mathbf{v}$ with zero-span $[i, j]$, define its circular span as $[j+1, i]$. In this case, the leading '1' is at location-$(j+1)$ and trailing '1' is at location-$i$. The TBTOG matrix of $C$ is of the following form:

$$G_{TB} = \begin{bmatrix} G_0 \\ G_c \end{bmatrix}, \tag{12}$$

where (1) $G_0$ is the generator matrix for the subcode $C_0$, (2) $G_c$ is the generator matrix for the coset space $C/C_0$, (3) no two rows of $G$ have their leading "ones" in the same columns, (4) no two rows have their trailing "ones" in the same column, and (5) the leading '1' of each row in $G_0$ appears in a column before the leading '1' of any row below it. This TBTOG matrix can be obtain by performing elementary operations of the rows of a given generator matrix of the code. $G_0$ is in the conventional trellis oriented form and generates the minimal conventional trellis $T_0$ for the subcode $C_0$ [2, 3, 20]. From the TBTOG matrix $G_{TB}$, we can determine the state space dimension profile easily. At the state boundary location-$i$, the dimension $\rho_i$ of state space $\Sigma_i$ is simply equal to the number of rows in $G_{TB}$ which are active at boundary location-$i$. Since the zero-spans of the rows in $G_c$ contain the interval $[t_1, t_2]$ as a subinterval and their circular spans are contained in the circular interval $[t_2+1, t_1]$, these rows are not active from boundary location-$t_1$ to boundary location-$t_2$.

As an example, consider the (8,4,4) RM code generated by

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

By elementary row operations, we obtain the following TBTOG matrix:

$$G_{TB} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \\ \mathbf{g}_4 \end{bmatrix} = \left[\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & \multicolumn{1}{c|}{1} & 0 & 0 & 0 & \multicolumn{1}{c|}{0} & 1 & 1 \end{array}\right]. \qquad (13)$$

The first three rows of $G_{TB}$ span the subcode $C_0$ and the fourth row $\mathbf{g}_4$ spans the coset space $C/C_0$ which has dimension one. The length of the zero-span of $\mathbf{g}_4$ is 4 and its circular span is [7,2]. Based on this TBTOG matrix, an 8-section tail biting trellis with two starting states and two ending states can be constructed as shown in Figure 3. Note that at boundary location-0 and boundary location-$L$, only the last row of $G_{TB}$ is active. The state space dimension profile of the tail biting trellis is (1,2,2,2,1,2,2,2,1) and its state space complexity profile is (2,4,4,4,2,4,4,4,2). To construct an 8-section bit-level tail biting trellis for this code, we first construct the minimal conventional 8-section bit-level trellis $T_0$ for $C_0$. Adding the fourth row $\mathbf{g}_4$ to the paths in $T_0$, we obtain the co-subtrellis $T_0(\mathbf{g}_4)$ for the only coset of $C_0$. The two co-trellises share a common span from boundary location-2 to boundary location-6. Putting these two co-subtrellises together and sharing the common span, we obtain the tail biting trellis as shown in Figure 3. This is a tail biting trellis for the (8,4,4) RM code and it has different structure from the one constructed in [6]. It has mirror symmetry, i.e., the right-half of the trellis is the mirror image of the left-half of the trellis. If it is sectionalized at boundary locations in $\{0,2,4,6,8\}$, a 4-section tail biting trellis is obtained as shown in Figure 4.

In [6], Calderbank, Forney and Vardy constructed the following generator matrix for the (24,12) Golay code,

$$G_{TB} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_{12} \end{bmatrix} = \left[\begin{array}{cccccccccccc}
11 & 01 & 11 & 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\
00 & 11 & 11 & 10 & 01 & 11 & 00 & 00 & 00 & 00 & 00 & 00 \\
00 & 00 & 11 & 01 & 10 & 11 & 11 & 00 & 00 & 00 & 00 & 00 \\
00 & 00 & 00 & 11 & 01 & 11 & 01 & 11 & 00 & 00 & 00 & 00 \\
00 & 00 & 00 & 00 & 11 & 01 & 11 & 01 & 11 & 00 & 00 & 00 \\
00 & 00 & 00 & 00 & 00 & 11 & 11 & 10 & 01 & 11 & 00 & 00 \\
00 & 00 & 00 & 00 & 00 & 00 & 11 & 01 & 10 & 11 & 11 & 00 \\
00 & 00 & 00 & 00 & 00 & 00 & 00 & 11 & 01 & 11 & 01 & 11 \\
\hline
11 & 00 & 00 & \multicolumn{1}{c|}{00} & 00 & 00 & 00 & \multicolumn{1}{c|}{00} & 11 & 01 & 11 & 01 \\
01 & 11 & 00 & \multicolumn{1}{c|}{00} & 00 & 00 & 00 & \multicolumn{1}{c|}{00} & 00 & 11 & 11 & 10 \\
10 & 11 & 11 & \multicolumn{1}{c|}{00} & 00 & 00 & 00 & \multicolumn{1}{c|}{00} & 00 & 00 & 11 & 01 \\
01 & 11 & 01 & \multicolumn{1}{c|}{11} & 00 & 00 & 00 & \multicolumn{1}{c|}{00} & 00 & 00 & 00 & 11
\end{array}\right]. \qquad (14)$$

This matrix is already in the TBTOF given by (12). The first 8 rows of $G_{TB}$ form the submatrix $G_0$ which generates the subcode $C_0$. $G_0$ is in the conventional trellis oriented form [2, 3, 20]. By inspecting the active spans of the rows of $G_0$, we readily find that the state space dimension profile of the minimal 24-section bit-level trellis $T_0$ for the subcode $C_0$ is

$$(0,1,1,2,2,3,3,4,4,4,4,4,4,4,4,4,4,3,3,2,2,1,1,0).$$

The maximum state space dimension of this trellis is 4 and hence its maximum state space has 16 states. If this trellis is sectionalized at the boundary locations in

$$\{0,2,4,6,8,10,12,14,16,18,20,22,24\}, \qquad (15)$$

we obtained a minimal 12-section trellis for the subcode $C_0$ whose state space dimension profile is (0,1,2,3,4,4,4,4,4,3,2,1,0). The last four rows of the TBTOG matrix $G_{TB}$ form the submatrix $G_c$ which spans the coset space $C/C_0$. The zero spans of these last four rows are [2,16], [4,18], [6,20] and [8,22], respectively, and they contain the interval [8,16] as the common zero span. Therefore, $T_0$ and its fifteen co-trellises share a common center span from boundary location-8 to boundary location-16. In fact, by inspecting the zero spans of these four rows, we can easily determine the common center span of any two co-trellises of $T_0$. Putting $T_0$ and its co-trellises together and sharing maximum common spans between them, we obtain the 16-state 12-section regular (or uniform) tail biting trellis for the (24,12) Golay code constructed by Calderbank, Forney and Vardy [6] as shown in Figure 6. By determining the number of rows of the TBTOG matrix $G_{TB}$ that are active at the boundary locations, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, we readily find the state space dimension profile of the 12-section tail biting trellis for the (24,12) Golay code is (4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4).

Construction of the bit level tail biting trellis for $C$ based on $G_{TB}$ can be accomplished easily based on the state defining information set at each boundary location, and the change of the state defining information set from one boundary location to the next boundary location as described in [2]. The information bits in the state defining set $A_i^s$ at the boundary location-$i$ defines the state space $\Sigma_i$ at boundary location-$i$ and the change of the state defining information set from $A_i^s$ at boundary locatio-$i$ to the state defining information set $A_{i+1}^s$ at boundary location-$(i+1)$ defines the state transitions. Labeling the states helps the construction.

Let $C'(t_1, t_2)$ be a linear subcode of $C(t_1, t_2)$ with dimension $m' < m$. A tail biting trellis for $C$ can be constructed based on this subcode $C'(t_1, t_2)$. Then the resultant tail biting trellis has $2^{m'}$ starting states and $2^{m'}$ ending states. Based on linear subcodes of $C(t_1, t_2)$, we can construct various tail biting trellises.

## 5.3 Tail biting trellises for RM codes

RM codes have relatively simple conventional trellis structure [2, 3, 7, 22] and can be effectively decoded with trellis-based decoding algorithms, such as the Viterbi algorithm and the MAP algorithm [23, 24]. In the following, we show that these codes also have simple tail biting trellis structure and hence can be decoded effectively with the two new iterative decoding algorithms proposed in the last two sections.

For any integers $m$ and $r$ with $0 < r < m$, there exists a binary $r$-th order RM code, decoded RM$(r, m)$, with the following parameters:

$$\text{Code length}: \quad n = 2^m;$$

$$\text{Dimension}: \quad k(r,m) = 1 + \binom{m}{1} + \cdots + \binom{m}{r};$$

$$\text{Minimum distance}: \quad d_{min} = 2^{m-r}.$$

For $1 \leq i \leq m$, let $\mathbf{v}_i$ be a $2^m$-tuple over GF(2) of the following form:

$$\mathbf{v}_i = (\underbrace{0\ldots0}_{2^{i-1}}, \underbrace{1\ldots1}_{2^{i-1}}, \underbrace{0\ldots0}_{2^{i-1}}, \cdots, \underbrace{1\ldots1}_{2^{i-1}}). \tag{16}$$

which consists of $2^{m-i+1}$ alternate all-zero and all-one $2^{i-1}$-tuples. Let $\mathbf{a} = (a_1, a_2, \cdots, a_n)$ and $\mathbf{b} = (b_1, b_2, \cdots, b_n)$ be two binary $n$-tuples. Define the following logic (boolean) product of $\mathbf{a}$ and $\mathbf{b}$,

$$\mathbf{a} \cdot \mathbf{b} \triangleq (a_1 \cdot b_1, a_2 \cdot b_2, \cdots, a_n \cdot b_n),$$

where "." denotes the logic product (or AND operation), i.e., $a_i \cdot b_i = 1$ if and only if $a_i = b_i = 1$. For simplicity, we use $\mathbf{ab}$ for $\mathbf{a} \cdot \mathbf{b}$. Let $\mathbf{v}_0$ denotes the all-one $2^m$-tuple, $\mathbf{v}_0 = (1, 1, \cdots, 1)$. For $1 \leq i_1 < i_2 < \cdots < i_l \leq m$, the product vector $\mathbf{v}_{i_1}\mathbf{v}_{i_2}\cdots\mathbf{v}_{i_l}$ is said to have degree $l$. The $r$-th order RM code, RM$(r, m)$, of length $2^m$ is generated (or spanned) by the following set of independent vectors:

$$\{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_m, \mathbf{v}_1\mathbf{v}_2, \mathbf{v}_1\mathbf{v}_3, \cdots, \mathbf{v}_{m-1}\mathbf{v}_m, \text{up to products of degree } r\}.$$

If the vectors in this set are arranged as rows of a matrix, we obtain a generator matrix for the $r$-th order RM code, denoted $G_{r,m}$. It is possible to construct a tail biting trellis for the $r$-th order RM code with a center common span of length $2^{m-1}$ from boundary location-$2^{m-2}$ to boundary location-$3 \cdot 2^{m-2}$. The lengths of the header and tail of each subtrellis are both $2^{m-2}$. The code vector $\mathbf{v}_m^* = \mathbf{v}_0 + \mathbf{v}_m + \mathbf{v}_{m-1}$, has the following form:

$$(\underbrace{1,1,\cdots,1}_{2^{m-2}},\underbrace{0,0,\ldots,0,0}_{2^{m-1}},\underbrace{1,1,\cdots,1}_{2^{m-2}}).$$

If we replace $\mathbf{v}_m$ by $\mathbf{v}_m^*$ and form the following basis for the subcode $C(2^{m-2}, 3 \cdot 2^{m-2})$ of the $r$-th order RM code, RM$(r,m)$:

$$\begin{aligned} B(2^{m-2}, 3 \cdot 2^{m-2}) &= \{\mathbf{v}_m^*, \mathbf{v}_m^*\mathbf{v}_1, \cdots, \mathbf{v}_m^*\mathbf{v}_{m-2}, \ldots, \text{up to products of degree} \\ &\qquad r \text{ with } \mathbf{v}_m^* \text{ as a product component}\} \\ &= \{\mathbf{v}_m^*\mathbf{v}_{i_1}\cdots\mathbf{v}_{i_l} : 1 \le i_1 < i_2 \cdots < i_l \le m-2 \\ &\qquad \text{and } 0 \le l \le r-1\}, \end{aligned} \tag{17}$$

where for $l = 0$, $\mathbf{v}_m^*\mathbf{v}_{i_1}\cdots\mathbf{v}_{i_l} \triangleq \mathbf{v}_m^*$. Then the subcode $C(2^{m-2}, 3 \cdot 2^{m-2})$ is spanned by

$$q_c = \binom{m-2}{0} + \binom{m-2}{1} + \cdots + \binom{m-2}{r-1} \tag{18}$$

linear independent codewords, called coset generators in $B(2^{m-2}, 3 \cdot 2^{m-2})$. Each generator in $B(2^{m-2}, 3 \cdot 2^{m-2})$ has a sequence of $2^{m-1}$ zeros from position-$(2^{m-2}+1)$ to position-$3 \cdot 2^{m-2}$, both the header and the tail of each codeword contain at least one nonzero component. In constructing a tail biting trellis for the RM$(r,m)$ code with a common center span of length $2^{m-1}$, we can use any subset of $B(2^{m-2}, 3 \cdot 2^{m-2})$ to span the coset space for forming the subtrellises (or to from the submatrix $G_c$ in the TBTOG matrix $G_{TB}$).

For $m = 3$ and $r = 1$, the RM(1,3) code is the (8,4,4) RM code given above. B(2,6) contains only one codeword which is $(1,1,0,0,0,0,1,1)$. The TBTOG matrix for this code is given in (13).

Consider the 2nd-order RM code, RM(2,5), which is a (32,16,8) code. There are a total of 4 generators in B(8,24) with a zero-span of length 16. They are:

$$\begin{array}{ccc} 11111111 & 0000000000000000 & 11111111 \\ 01011010 & 0000000000000000 & 01011010 \\ 00111100 & 0000000000000000 & 00111100 \\ 11110000 & 0000000000000000 & 00001111. \end{array} \tag{19}$$

Suppose we want to construct a tail biting trellis with 8 starting states and 8 ending states. We simply choose three generators from $G_c$. A choice to minimize the trellis complexity is desired. One choice gives the following TBTOG:

$$G_{TB} = \left[\begin{array}{c|c|c} \multicolumn{3}{c}{11111111000000000000000000000000} \\ \multicolumn{3}{c}{01010101101010100000000000000000} \\ \multicolumn{3}{c}{00110011110011000000000000000000} \\ \multicolumn{3}{c}{00010001111011101000100010001000} \\ \multicolumn{3}{c}{00001111111100000000000000000000} \\ \multicolumn{3}{c}{00000101111110101010000010100000} \\ \multicolumn{3}{c}{00000011111111001100000011000000} \\ \multicolumn{3}{c}{00000000111111110000000000000000} \\ \multicolumn{3}{c}{00000000000000001111111100000000} \\ \multicolumn{3}{c}{00000000000000000101010110101010} \\ \multicolumn{3}{c}{00000000000000000011001111001100} \\ \multicolumn{3}{c}{00000000000000000000111111110000} \\ \multicolumn{3}{c}{00000000000000000000000011111111} \\ \hline 01011010 & 0000000000000000 & 01011010 \\ 00111100 & 0000000000000000 & 00111100 \\ 11110000 & 0000000000000000 & 00001111 \end{array}\right]. \tag{20}$$

The first 13 rows of $G_{TB}$ is in TOF and generate a (32,13) subcode $C_0$. The dimension of the state space at the center point (boundary location-16) of the bit-level minimal conventional trellis $T_0$ for $C_0$ is 3. The 7 nonzero linear combinations of the last 3 rows of $G_{TB}$ generate 7 co-subtrellises of $T_0$. The state space dimension profile of the tail biting trellis is: (3,4,5,6,6, 7,7,7,6,7,7,7,6,6,5,4,3,4,5,6,6,7,7,7,6,7,7,7,6,6,5,4,3). The total number of states and total number of branches of this tail biting trellis are 2,392 and 3,392, respectively. The state space dimension at the midpoint of this tail biting trellis is 3. The maximum state space dimension is 7. However, the minimal conventional bit-level trellis with one pair of starting and ending states for this code has a total of 4,798 states and a total of 6,396 branches. The state space dimension at the midpoint of this conventional trellis is 6 and the maximum state space dimension is 9. We see that the above tail biting trellis representation of the (32,16) RM code reduces both the state and branch complexities by about one half. The number of states at the midpoint of the tail biting trellis is the square root of the number of states at the midpoint of conventional minimal trellis for the code. If we sectionalized the bit-level tail biting trellis at locations in $\{0, 4, 8, 12, 16, 20, 24, 28, 32\}$, we obtain an uniform 8-section tail biting trellis with state space dimension profile, (3,6,6,6,3,6,6,6,3). However, the state space dimension profile of the corresponding conventional 8-section trellis is (0,4,6,8,6,8,6,4,0). The TBTOG matrix given above meets the minimum sum of spans condition, and therefore the tail biting trellis is minimal in terms of product of the state-space sizes [6].

For $r = 2$ and $m = 6$, the second-order RM code, RM(2,6), is a (64,22,16) code. The conventional minimal bit-level 64-section trellis for this code has a total of 324,862 states and a total of 375,036 branches. Its maximum state complexity is $2^{14}$ and the number of states at midpoint (boundary location-32) is $2^{10}$. For this RM code, a bit-level 64-section tail biting trellis with 8 starting states, 8 ending states and a center common span of length 32 can be constructed. This tail biting trellis has a total of 117,360 states and a total 136,672 branches. Its maximum state complexity is $2^{12}$ and the number of states at midpoint is $2^7$. A tail biting trellis with 16 starting states and 16 ending states for this code can also be constructed.

For any positive integer $i$ such that $m - r - 1 \leq i \leq m - 2$, there exists a tail biting

[15] B. D. Kudryashov, "Decoding of block codes obtained from convolutional codes," *Problemy Peredachi Informatsii,* Vol. 26, No. 2, pp. 18-26. April-June 1990 (in Russian). English Translation, Plenum Publishing Corporation, October 1990.

[16] R. V. Cox and C. E. Sundberg, " An efficient adaptive circular Viterbi algorithm for decoding generalized tailbiting convolutional codes," *IEEE Trans. Vehicular Tech.*, Vol. 43, pp. 57-68, February 1994.

[17] J.B. Anderson and K. E. Tepe, Private communication.

[18] J.B. Anderson and S. M. Hladik, "An optimal circular Viterbi decoder," in preparation.

[19] T. Fujiwara, H. Yamamoto, T. Kasami and S. Lin, "A Trellis-Based Recursive Maximum Likelihood Decoding Algorithm for Linear Block Codes," *IEEE Trans. Inform. Theory,* Vol. 44, No. 2, March 1998.

[20] F. R. Kschischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inform. Theory*, Vol. 41, pp. 1924-1937, November 1995.

[21] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error Correcting Codes,* North-Holland, 1977.

[22] T. Kasami, T. Takata, T. Fujiwara and S. Lin, "On the Optimum Bit Orders with respect to the State Complexity of Trellis Diagrams of Binary Linear Codes," *IEEE Trans. Inform Theory*, Vol. 39, pp. 242-244, January 1993.

[23] L. R. Bahl. J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inform. Theory*, Vol. 20, No. 2, pp. 284-287, 1974.

[24] Y. Liu, S. Lin and M. Fossorier, "MAP Algorithm for Decoding Linear Codes based on Sectionalized Trellis Diagrams," To appear in *IEEE Trans. Commun*, 2000.

trellis for the RM$(r, m)$ code with a center common span of length $2^m - 2^{i+1}$ from boundary location-$2^i$ to boundary location-$(2^m - 2^i)$ [12].

## 6 Conclusion

In this paper, two new efficient iterative algorithms for decoding linear codes, block or convolutional, based on their tail biting trellises have been presented. One is an unidirectional algorithm and other is a bidirectional algorithm. Both algorithms achieve virtually optimum error performance with a maximum of 2 to 4 iterations with a significant reduction in decoding computational complexity. For short tail biting convolutional codes, the bidirectional algorithm converges to optimum MLD error performance faster and gives better error performance than the unidirectional algorithm. For long tail biting convolutional codes, both algorithms perform equally well and achieves virtually optimum error performance with a maximum of 2 iterations. The bidirectional algorithm reduces decoding delay. Both algorithms outperform all the existing tail biting trellis-based algorithms in performance and decoding complexity. In the paper, a simple method for constructing tail biting trellises for linear block codes has also been presented.

## References

[1] S. Lin and D. J. Costello Jr., *Error Control Coding: fundamentals and applications,* Prentice Hall, Englewood Cliffs, NY, 1983.

[2] S. Lin, T. Kasami, T. Fujiwara and M. P. C. Fossorier, *Trellises and Trellis Based Decoding Algorithms for Linear Block Codes,* Kluwer Academic Publishers, Boston, MA, 1998.

[3] A. Vardy, "Trellis Structure of Codes," the Handbook of Coding Theory, V.S. Pless and W. C. Huffman (Editors), Elsevier, Amsterdam, 1998.

[4] G. Solomon and H. van Tilborg, " A connection between block and convolutional codes", *SIAM J. Appl. Math.*, Vol. 37, No. 2, pp. 358-369, Oct. 1979.

[5] H. H. Ma and J. K. Wolf, "On tail biting convolutional codes," *IEEE Trans. Commun.*, Vol. 34, No. 2, pp. 104-111, Feb. 1986.

[6] A. R. Calderbank, G. D. Forney, Jr., and A. Vardy, "Minimal Tail-Biting Trellises: Golay Code and More," *IEEE Trans. Inform. Theory*, Vol. 45, No. 5, pp. 1435-1455. July 1999.

[7] G. D. Forney Jr., "Coset code - Part II: Binary Lattices and Related Codes," *IEEE Trans. Inform. Theory*, Vol. 34, No. 5, pp. 1152-1188, September 1988.

[8] D. J. Muder, "Minimal trellises for block codes," *IEEE Trans. Inform. Theory*, Vol. 34, pp. 1049-1053, September 1988.

[9] R. Kötter and A. Vardy, "Construction of Minimal Tail-Biting Trellises", in *Proc. 1998 IEEE Inform. Theory Workshop,* pp. 72-74, Killarney, Ireland, June 1998.

[10] ——, "The Theory of tail-biting trellises", in preparation.

[11] R. Shao, S. Lin, and M. Fossorier, "An Iterative Bidirectional Decoding Algorithm for Tail Biting Codes," *Proc. 1999 IEEE Inform. Theory Workshop,* Kruger National Park, South Africa, June 22-25, 1999.

[12] R. Shao, *Decoding of Linear Codes Based on Their Tail Biting Trellises and Efficient Stopping Criteria for Turbo Decoding*, Ph.D. Dissertation, University of Hawaii at Manoa, December 1999.

[13] Q. Wang and V. K. Bhargava, "An Efficient Maximum Likelihood Decoding Algorithm for Generalized Tail Biting Convolutional Codes Including Quasicyclic Codes," *IEEE Trans. Commun.*, Vol. 37, No. 8, pp. 875-879, August 1989.

[14] K. S. Zigangirov and V. V. Chepyshov, "Study of decoding tailbiting convolutional codes," *Proc. 4-th Joint Swedish-Soviet International Workshop Informat. Theory*, pp. 52-55, Gotland, Sweden, August 1989.

Table 1: Decoding complexity of various decoding algorithms in terms of number of Viterbi trials for the (64,32) tail biting convolutional code generated by $g_1 = (1,1,1,0,0,1,0,1)$ and $g_2 = (1,0,0,1,1,1,1,1)$ over BSC.

| SNR | Bar-David | Two-step | Wang-Bhargava | CVA | WA-V ($I_{max} = 4$) |
|---|---|---|---|---|---|
| 4.588 | 116.21 | 67.42 | 14.67 | 2.23 | 1.43 |
| 4.812 | 72.1 | 60.17 | 13.41 | 2.20 | 1.35 |
| 5.032 | 67.65 | 58.24 | 5.76 | 2.15 | 1.32 |
| 5.535 | 62.25 | 54.27 | 2.3 | 2.08 | 1.21 |
| 6.123 | 56.79 | 47.11 | 1.48 | 2.01 | 1.13 |

Table 2: Percentage of transmitted codewords that are decoded into most likely codewords for the (68,34) convolutional tail biting code generated by polynomials (712,476) using the WA-V and the IBD-V algorithms (minimum 10,000 transmitted codewords).

| | WA-V | | | IBD-V | |
|---|---|---|---|---|---|
| SNR | $I_{max} = 1$ | $I_{max} = 2$ | $I_{max} = 4$ | $I_{max} = 1$ | $I_{max} = 2$ |
| 1.0dB | 71.65% | 93.68% | 95.98% | 92.91% | 98.66% |
| 1.5dB | 79.49% | 97.65% | 99.34% | 95.11% | 99.81% |
| 2.0dB | 84.44% | 98.61% | 99.54% | 96.85% | 99.72% |
| 2.5dB | 90.54% | 99.67% | 99.89% | 98.71% | 99.96% |
| 3.0dB | 94.13% | 99.90% | 99.95% | 99.41% | 99.99% |

Table 3: Average number of Viterbi updates required for decoding the (68,34) convolutional tail biting code generated by polynomials (712,476) using the WA-V and the IBD-V algorithms.

| SNR | WA-V | | IBD-V | | |
|---|---|---|---|---|---|
| | $I_{max} = 2$ | $I_{max} = 4$ | $I_{max} = 1$ | $I_{max} = 2$ | $I_{max} = 2$ (per decoder) |
| 1.0dB | 53.02 | 79.33 | 53.02 | 79.98 | 39.99 |
| 2.0dB | 49.61 | 70.33 | 49.61 | 69.92 | 34.96 |
| 3.0dB | 45.90 | 59.96 | 45.90 | 60.52 | 30.26 |
| 4.0dB | 41.84 | 50.61 | 41.84 | 50.71 | 25.35 |
| 5.0dB | 39.17 | 44.69 | 39.17 | 44.66 | 22.33 |

Table 4: Average number of Viterbi updates required for decoding the (128,64) convolutional tail biting code generated by polynomials (554,744) using the WA-V and the IBD-V algorithms.

| SNR | WA-V | | IBD-V | | |
|---|---|---|---|---|---|
| | $I_{max} = 2$ | $I_{max} = 4$ | $I_{max} = 1$ | $I_{max} = 2$ | $I_{max} = 2$ (per decoder) |
| 1.0dB | 99.17 | 145.16 | 99.17 | 137.09 | 68.55 |
| 2.0dB | 92.18 | 125.92 | 92.18 | 122.36 | 61.18 |
| 3.0dB | 83.46 | 103.99 | 83.46 | 103.27 | 51.64 |
| 4.0dB | 77.69 | 92.89 | 77.69 | 91.46 | 45.73 |
| 5.0dB | 73.17 | 82.90 | 73.17 | 82.35 | 41.17 |

Table 5: Average number of Viterbi updates required for decoding the (24,12) Golay code with the 16-state tail biting trellis using the WA-V and the IBD-V algorithms.

| SNR | WA-V | | IBD-V | | |
|---|---|---|---|---|---|
| | $I_{max} = 2$ | $I_{max} = 4$ | $I_{max} = 1$ | $I_{max} = 2$ | $I_{max} = 2$ (per decoder) |
| 1.0dB | 17.79 | 26.04 | 17.79 | 26.42 | 13.21 |
| 2.0dB | 16.02 | 21.26 | 16.02 | 21.60 | 10.80 |
| 3.0dB | 14.24 | 16.88 | 14.24 | 17.12 | 8.56 |
| 4.0dB | 13.09 | 14.25 | 13.09 | 14.35 | 7.17 |
| 5.0dB | 12.43 | 12.87 | 12.43 | 12.90 | 6.45 |

Table 6: Decoding complexity in terms of number of additions and comparisons with various decoding algorithms at SNR=1.0dB (10,000 blocks).

| Code | VTMLD | RMLD | WA-V ($I_{max} = 4$) | IBD-V ($I_{max} = 2$) |
|---|---|---|---|---|
| Golay (24,12) | 9,615 | 3,379 | 1,383 | 1,668 |
| (64,32) ($m = 6$) | 397,375 | 250,043 | 14,739 | 17,694 |
| (128,64) ($m = 6$) | 794,687 | 643,323 | 28,314 | 32,493 |
| (68,34) ($m = 7$) | 1,679,999 | 1,000,251 | 32,493 | 39,761 |

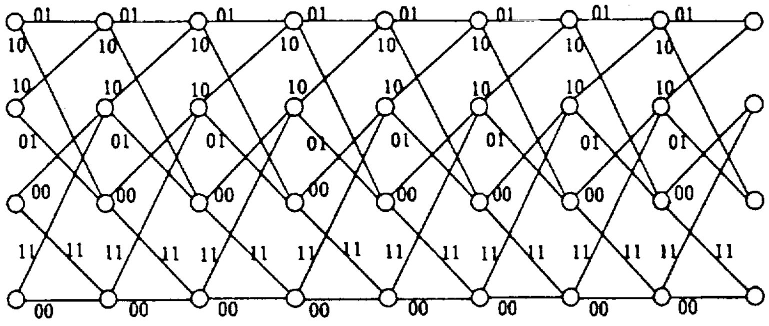


Figure 1: An 8-section tail biting trellis for the rate-1/2 (2,1,2) convolutional tail biting code generated by $\mathbf{g}_1 = (1,0,1)$ and $\mathbf{g}_2 = (1,1,1)$.

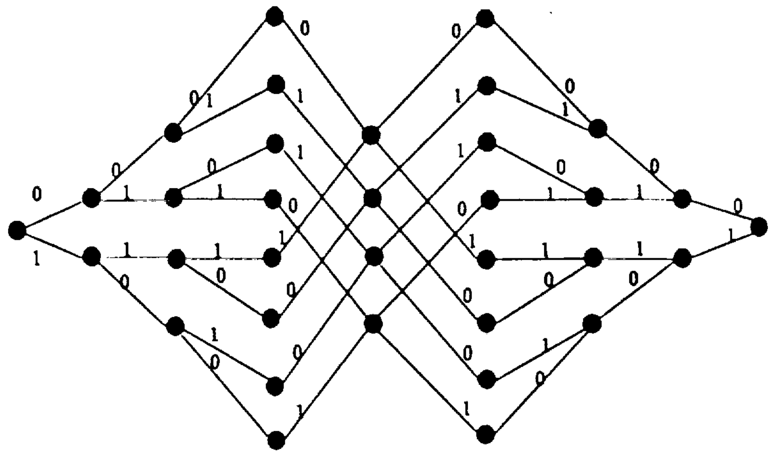


Figure 2: A minimal 8-section conventional trellis for the (8,4,4) RM code.

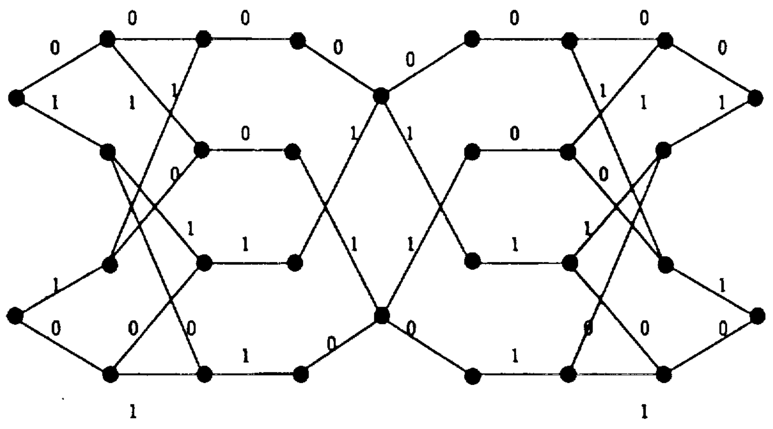


Figure 3: A minimal 8-section tail biting trellis for the (8,4,4) RM code.

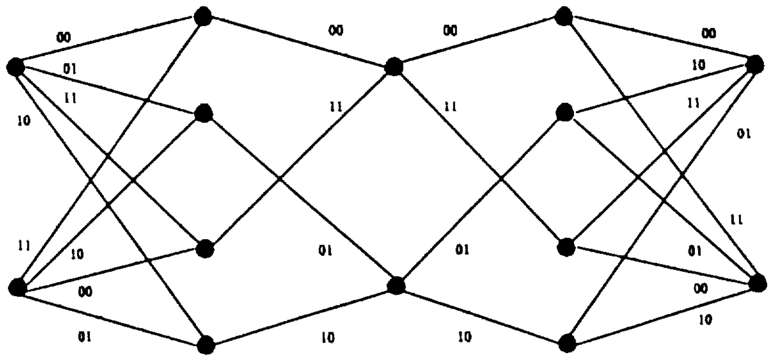


Figure 4: A 4-section tail biting trellis for the (8,4,4) RM code.

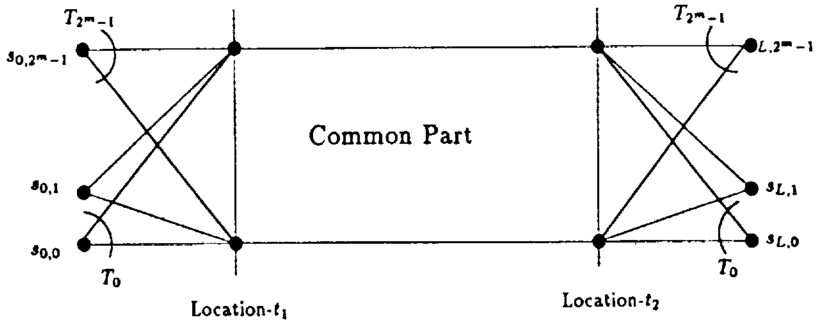


Figure 5: General structure of a tail biting trellis.

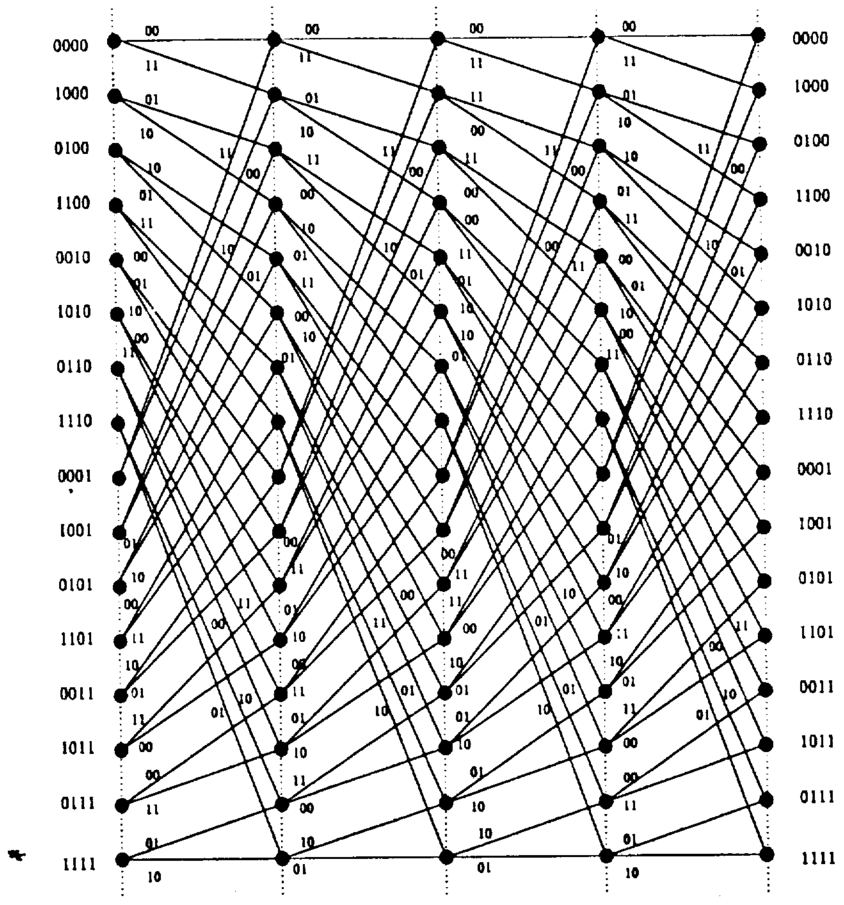
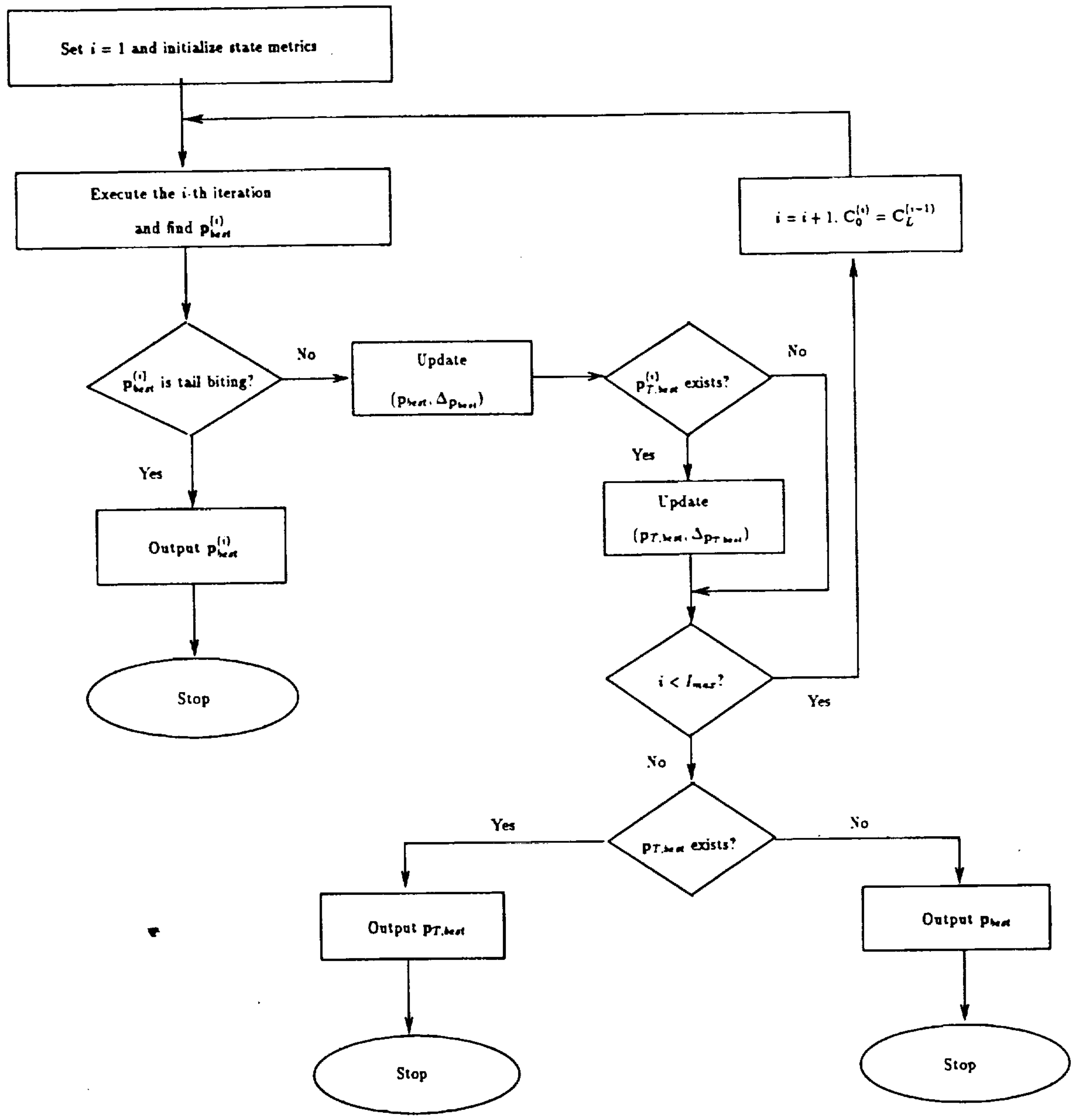


Figure 6: Four sections of the minimal tail biting trellis for the (24,12,8) Golay code.



Set $i = 1$ and initialize state metrics
Execute the $i$-th iteration and find $p_{best}^{(i)}$
$p_{best}^{(i)}$ is tail biting?
No
Yes
Update $(p_{best}, \Delta p_{best})$
$p_{T,best}^{(i)}$ exists?
No
Yes
Update $(p_{T,best}, \Delta p_{T,best})$
$i = i + 1$, $C_0^{(i)} = C_L^{(i-1)}$
Output $p_{best}^{(i)}$
Stop
$i < I_{max}$?
Yes
No
$p_{T,best}$ exists?
Yes
No
Output $p_{T,best}$
Output $p_{best}$
Stop
Stop

Figure 7: A flow-chart for the Wrap-Around Viterbi algorithm.

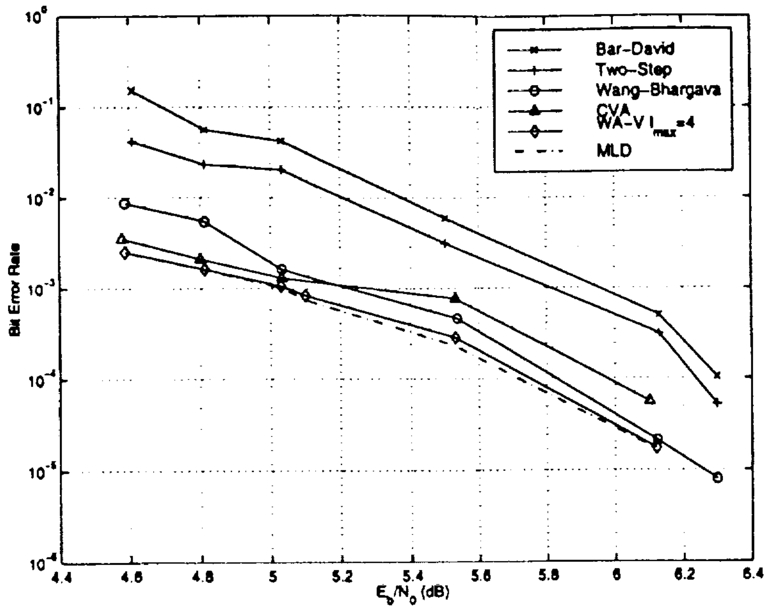


Figure 8: BER performance of various decoding algorithms for the rate-1/2 (64,32) tail biting convolutional code with memory order $m = 7$ and generated by (712,476) over BSC.

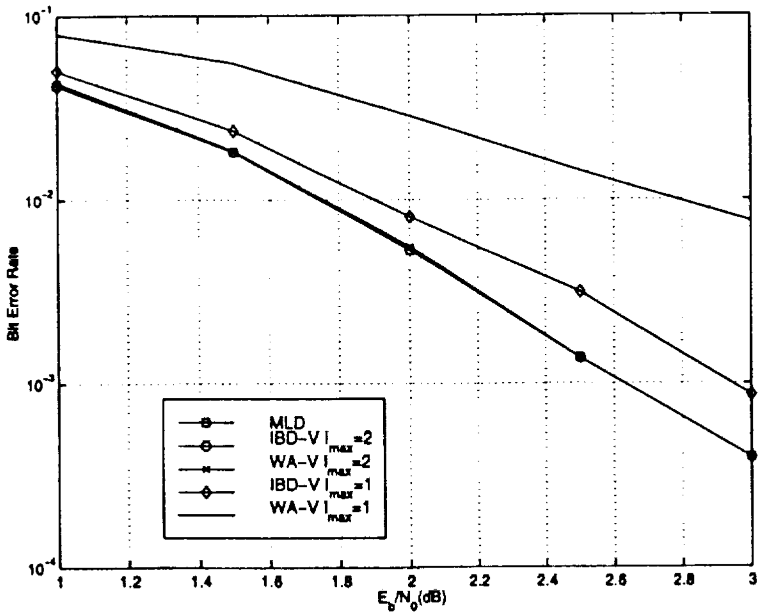


Figure 9: BER performance of the (128,64) tail biting convolutional code generated by (554,744) using the WA-V and the IBD-V algorithms.

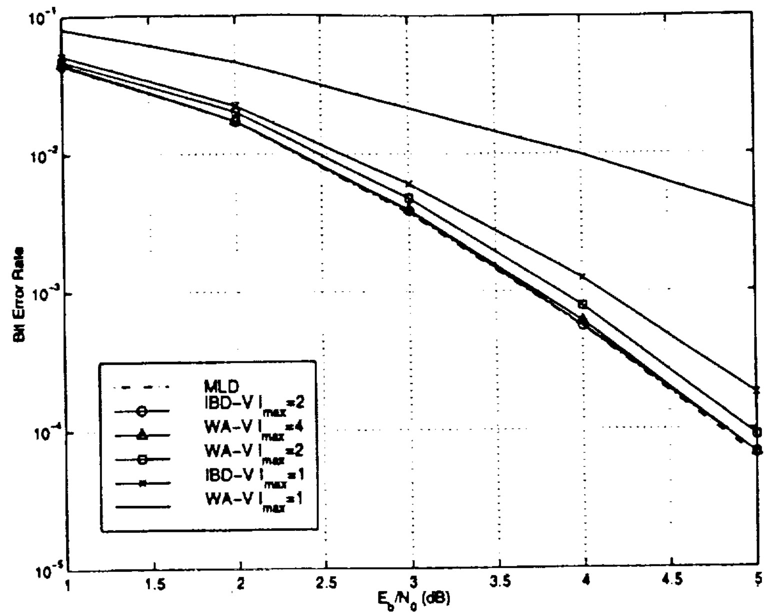


Figure 10: BER performance of the (24,12) Golay code with the 16-state tail biting trellis using the WA-V and the IBD-V algorithms.

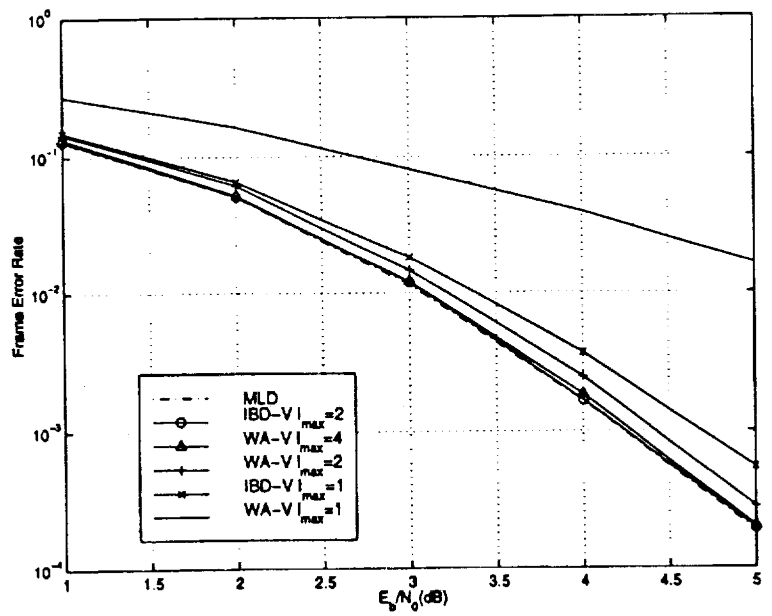


Figure 11: FER performance of the (24,12) Golay code with the 16-state tail biting trellis using the WA-V and the IBD-V algorithms.

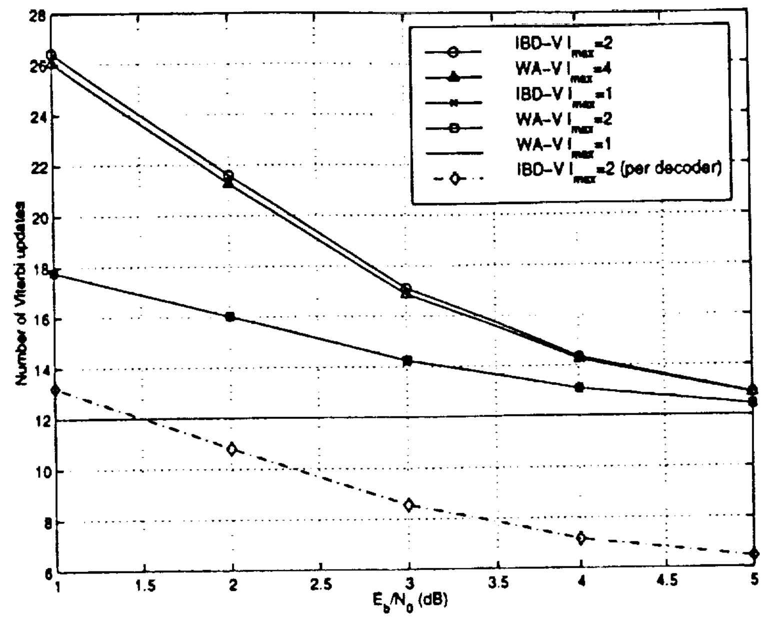


Figure 12: Decoding complexity of the (24,12) Golay code with the 16-state tail biting trellis using the WA-V and the IBD-V algorithms.

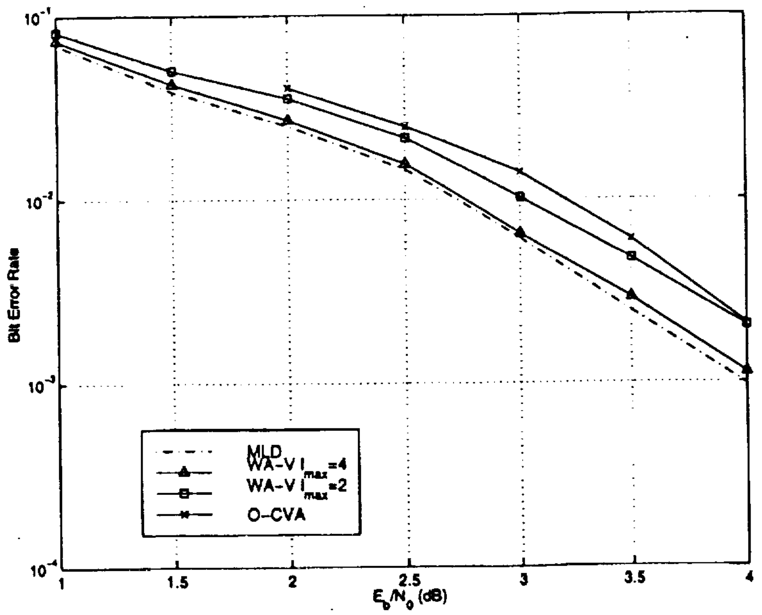


Figure 13: BER performance of the (24,12) Golay code with the 64-state tail biting trellis using the WA-V and the O-CVA algorithms.

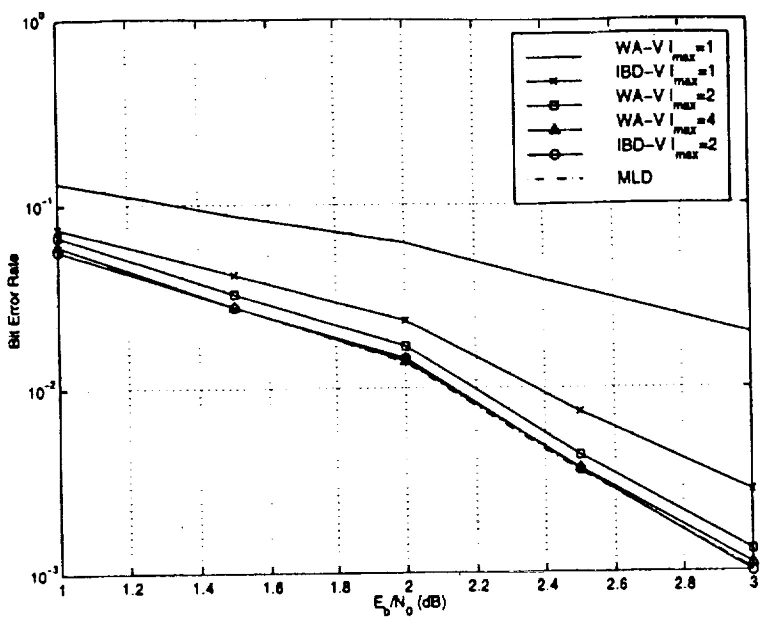


Figure 14: BER performance of the (68,34) tail biting convolutional code generated by polynomials (712,476) using the WA-V and the IBD-V algorithms.

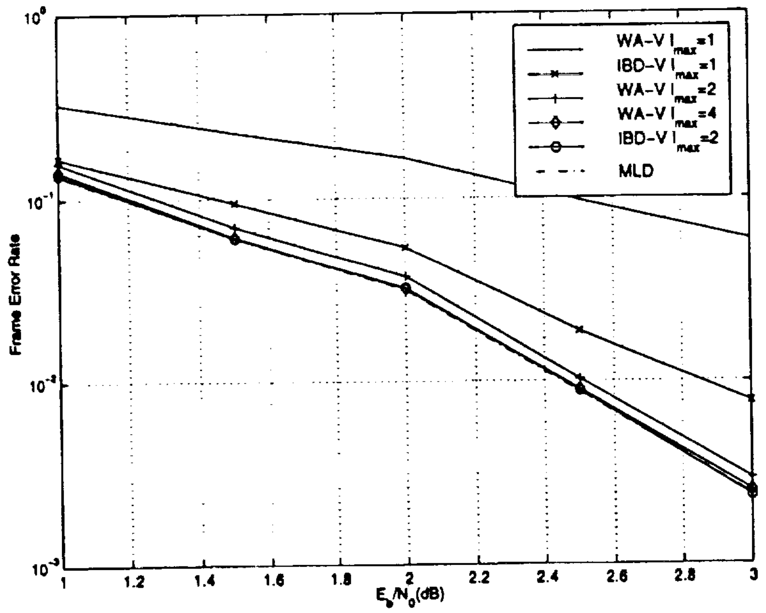


Figure 15: FER performance of the (68,34) tail biting convolutional code generated by polynomials (712,476) using the WA-V and the IBD-V algorithms.