TRELLISES AND TRELLIS-BASED DECODING ALGORITHMS FOR LINEAR BLOCK CODES

Technical Report

to

NASA

Goddard Space Flight Center
Greenbelt, Maryland 20771

Grant Numbers: NAG 5-931
NAG 5-2938

Report Number: 98-003

Principal Investigator: Shu Lin

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

April 20, 1998

TRELLISES AND TRELLIS-BASED DECODING ALGORITHMS FOR LINEAR BLOCK CODES

Part 3

Shu Lin and Marc Fossorier

April 20, 1998

N[redacted]

IN-61-CR
170[illegible]4
p.116

10 THE VITERBI AND DIFFERENTIAL TRELLIS DECODING ALGORITHMS

Decoding algorithms based on the trellis representation of a code (block or convolutional) drastically reduce decoding complexity. The best known and most commonly used trellis-based decoding algorithm is the Viterbi algorithm [23, 79, 105]. It is a maximum likelihood decoding algorithm. Convolutional codes with the Viterbi decoding have been widely used for error control in digital communications over the last two decades. This chapter is concerned with the application of the Viterbi decoding algorithm to linear block codes. First, the Viterbi algorithm is presented. Then, optimum sectionalization of a trellis to minimize the computational complexity of a Viterbi decoder is discussed and an algorithm is presented. Some design issues for IC (integrated circuit) implementation of a Viterbi decoder are considered and discussed. Finally, a new decoding algorithm based on the principle of **compare-select-add** is presented. This new algorithm can be applied to both block and convolutional codes and is more efficient than the conventional Viterbi algorithm based on the **add-compare-select** principle. This algorithm is particularly efficient for rate-$1/n$ antipodal convolutional codes and their high-rate punctured codes. It reduces computational complexity by one-third compared with the Viterbi algorithm.

10.1 THE VITERBI DECODING ALGORITHM

The Viterbi algorithm is based on the simple idea that among the paths merging into a state in the code trellis, only the most probable path needs to be saved for future processing and all the other paths can be eliminated without affecting decoding optimality. This elimination of the less probable paths from further consideration drastically reduces decoding complexity. The path being saved is called the **survivor**. Therefore, there is a survivor at each state in the trellis at every level. The survivors at each level of the code trellis are extended to the next level through the composite branches between the two levels. The paths that merge into a state at the next level are then compared and the most probable path is selected as the survivor. This process continues until the end of the trellis is reached. At the end of the trellis, there is only one state, the final state σ_f, and there is only one survivor, which is the most likely codeword. Decoding is then completed.

Viterbi decoding of a linear block code based on a sectionalized trellis diagram $T(\{h_0, h_1, \ldots, h_L\})$, with section boundary locations $0 = h_0 < h_1 < \cdots < h_L = N$, is carried out serially, section by section, from the initial state σ_0 to the final state σ_f. Suppose the decoder has processed j trellis sections up to time-h_j. There are $|\Sigma_{h_j}(C)|$ **survivors**, one for each state in $\Sigma_{h_j}(C)$. These survivors together with their **path** (or **state**) **metrics** are stored in memory. To process the $(j+1)$-th section, the decoder executes the following steps:

(1) Each survivor is extended through the composite branches diverging from it to the next state level at time-h_{j+1}.

(2) For each composite branch into a state in $\Sigma_{h_{j+1}}(C)$, find the single branch with the largest (correlation) metric. The metric computed is the branch metric.

(3) Replace each composite branch by the branch with the largest metric.

(4) **Add** the metric of a branch to the metric of the survivor from which the branch diverges. For each state σ in $\Sigma_{h_{j+1}}(C)$, **compare** the metrics of the paths converging into it and **select** the path with the largest path metric as the survivor terminating at state σ. This step is called the **add-compare-select** (ACS) procedure in the Viterbi algorithm.

The decoder executes the above steps repeatedly, section by section, until it reaches the final state σ_f. At this point, there is one and only one survivor, which is the decoded codeword and the most likely codeword. The information bits corresponding to this decoded codeword are then delivered to the user. The decoding window is simply the code length.

Using the above decoding algorithm, the total number of operations (additions and comparisons) can be computed easily. This number can be reduced significantly if sectionalization of a trellis is done properly [60]. This will be discussed in the next section.

10.2 OPTIMUM SECTIONALIZATION OF A CODE TRELLIS: LAFOURCADE-VARDY ALGORITHM

In decoding a block code with the Viterbi algorithm, the total number of computations depends on the sectionalization of the trellis diagram for the code. A sectionalization of a code trellis for a code C that gives the smallest total number of computations is called an optimum sectionalization for C. An optimum sectionalization is not necessarily unique. In the following, an algorithm for finding an optimum sectionalization is presented. This algorithm was devised by Lafourcade and Vardy [60].

The Lafourcade-Vardy (LV) algorithm is based on the following simple fact:

(F) For any integers x and y with $0 \leq x < y \leq N$, a section from time-x to time-y in any sectionalized trellis $T(U)$ with $x, y \in U$ and $x+1, x+2, \ldots, y-1 \notin U$ is identical.

Let $\varphi(x, y)$ denote the number of computations required in steps (1) to (4) of the Viterbi algorithm to process the trellis section from time-x to time-y in any sectionalized code trellis $T(U)$ with $x, y \in U$ and $x+1, x+2, \ldots y-1 \notin U$. It follows from the above simple fact (F) that $\varphi(x, y)$ is determined only by x and y. Let $\varphi_{\min}(x, y)$ denote the smallest number of computations of steps (1)

Table 10.1. Optimum sectionalizations.

Code	Optimum Sectionalization U	Complexity $\varphi_{\min}(0, N)$	Complexity N-section
$RM_{1,6}$	$\{0, 4, 8, 12, 16, 24, 32, 40, 48, 52, 56, 60, 63, 64\}$	806	2,825
$RM_{2,6}$	$\{0, 8, 16, 32, 48, 56, 61, 63, 64\}$	101,786	425,209
$RM_{3,6}$	$\{0, 8, 16, 24, 32, 40, 48, 56, 64\}$	538,799	773,881

to (4) to process the trellis section(s) from time-x to time-y in any sectionalized code trellis $T(U)$ with $x, y \in U$. The value, $\varphi_{\min}(0, N)$, gives the total number of computations of the Viterbi algorithm for the code trellis with an optimum sectionalization. Then, it follows from (F) and the definitions of $\varphi(x, y)$ and $\varphi_{\min}(x, y)$ that

$$\varphi_{\min}(0, y) = \begin{cases} \min\left\{\varphi(0, y), \min\limits_{0<x<y} \{\varphi_{\min}(0, x) + \varphi(x, y)\}\right\}, & \text{for } 1 < y \leq N, \\ \varphi(0, 1), & \text{for } y = 1. \end{cases} \tag{10.1}$$

We can compute $\varphi_{\min}(0, y)$ for every y with $0 < y \leq N$ efficiently in the following way: The values of $\varphi(x, y)$ for $0 \leq x < y \leq N$ are computed using the structure of the trellis section from time-x to time-y. First, the value of $\varphi_{\min}(0, 1)$ is computed. For an integer y with $1 < y \leq N$, $\varphi_{\min}(0, y)$ can be computed from $\varphi_{\min}(0, x)$ and $\varphi(x, y)$ with $0 < x < y$. By storing the information when the minimum value occurs in the right-hand side of (10.1), an optimum sectionalization is found from the computation of $\varphi_{\min}(0, N)$.

Example 10.1 Table 10.1 gives the optimum sectionalizations for three RM codes of length 64 using the LV algorithm.

△△

10.3 SOME DESIGN ISSUES FOR IC IMPLEMENTATION OF VITERBI DECODERS FOR LINEAR BLOCK CODES

Theoretically, any linear block code can be decoded by applying the Viterbi algorithm to a trellis for the code. However, practical limitations preclude the application of this algorithm to many good codes with long block lengths. The main reasons are the increases in state complexity, state connectivity, and branch complexity of the trellises for good block codes as the length of the codes increases. Much of the research on maximum likelihood decoding of linear block codes with the Viterbi algorithm over a code trellis has focussed on the minimization of the number of computations required for decoding a received sequence. If the actual decoding is intended to be performed using a stored program approach (a software implementation) that executes the operations needed to decode a received sequence sequentially, then this approach will lead to the fastest decoding speed. However, if an IC (hardware) implementation is intended, then many other factors besides the number of decoding computations must be considered. We must consider the factors that affect the circuit requirements, wire-routing within an IC chip, chip size, circuit utilization, power consumption, ACS computation speed, and other implementation issues. As a result, an alternate approach that is more suitable for IC implementation is desired.

For IC implementation of a Viterbi decoder for a linear block code, besides the state and branch complexities, other important trellis structural properties that should be included in the design considerations are state connectivity, the parallel structure, regularity, and symmetry. Proper use of these structural properties may result in a simpler decoding circuit and a higher decoding speed.

Optimum sectionalization in terms of minimizing the computational complexity, in general, results in a non-uniformly sectionalized trellis diagram. In a Viterbi decoder, quantities such as the branch labels, survivor path metrics, and survivor path labels generally reside in word registers, which are basically an ordered sequence of bit registers. The same hardware is used to process all trellis sections. If a register must store a particular variable, such as a branch metric or a state metric, it must be designed to accommodate the largest value of the variable over all trellis sections. Since the section lengths for a non-uniformly sectionalized trellis vary from one section to another, the registers involved must be designed based on the longest section. This may increase the relative complexity of an IC Viterbi decoder. Therefore, for IC implementation of a Viterbi decoder, a uniformly sectionalized trellis is more desirable.

Although a minimal trellis reduces the state and branch complexities, the states are densely connected. For long codes, this dense connection between the states causes serious wire-routing (interconnection) problems within an IC chip for hardware implementation of a Viterbi decoder and requires a large area of the chip (or a multilayer chip) to accommodate the decoding circuit. Furthermore, interconnections increase internal communications between various parts of the decoding circuit, which slow down the decoding speed and increase power consumption. Let $\rho_{\max}(C)$ be the maximum state space dimension of a minimal trellis for a code C. Then the number of registers required to store the survivor paths and their metrics must be $2^{\rho_{\max}(C)}$. If a separate ACS circuit is required for processing each state at each trellis level, then $2^{\rho_{\max}(C)}$ ACS circuits are needed. If the differences between $\rho_{\max}(C)$ and the state space dimensions at many section boundary locations are large, then many of the registers and ACS circuits are not used during the decoding process. This results in poor hardware utilization efficiency. All the above problems may be solved or partially solved by using a non-minimal trellis with a proper parallel decomposition, as discussed in Chapter 7. Regularity among the trellis sections also helps to overcome the above problems and reduces decoding complexity. Symmetry structure, such as mirror symmetry, allows bidirectional decoding, which speeds up the decoding process. Therefore, for hardware implementation of a Viterbi decoder for a linear block code, a non-minimal trellis may result in a simpler and faster decoding circuit with a higher hardware utilization efficiency. In design, both minimal and non-minimal trellises should be considered and the one that results in a simpler circuit and a higher decoding speed should be used.

In the following, we examine some key factors that affect the decoding complexity and speed of a Viterbi decoder based on a minimal or non-minimal trellis. The non-minimal trellis structure presented in Section 7.1 reduces internal communications and allows independent parallel processing of the subtrellises, while decreasing the complexity of an IC Viterbi decoder. It has significant advantages over the minimal trellis for IC implementation of a Viterbi decoder.

10.3.1 Hardware Utilization Efficiency and Effective Computational Complexity

Consider an IC Viterbi decoder based on an L-section trellis for a linear (N, K) block code C with section boundary locations at $h_0 = 0, h_1, h_2, \ldots, h_L = N$. While many VLSI structures have been described for a Viterbi decoder [10, 38, 100], the most widely implemented structure is based on the ACS-array architecture, wherein each abstract state in the trellis manifests itself as a physical ACS circuit on the IC, and the same ACS circuits are repeatedly used for all levels in the trellis. The ACS circuits can be labeled ACS-l for $1 \leq l \leq 2^{\rho_{L,\max}(C)}$, where $\rho_{L,\max}(C)$ is the maximum state space dimension of the L-section minimal trellis for C. We assume that $\rho_{L,\max}(C)$ is fixed no matter whether a minimal trellis or a non-minimal trellis is used in the decoder design.

The ACS circuits work as follows. At time-0, the metrics of the ACS circuits corresponding to the originating states of each parallel subtrellis are initialized to 0. At time-h_1, the ACS-l corresponding to state $\sigma^{(l)} \in \Sigma_{h_1}(C)$ at the end of section-1 of the trellis, for $1 \leq l \leq |\Sigma_{h_1}(C)|$, has the metric of state $\sigma^{(l)}$. The index of the surviving branch into state $\sigma^{(l)}$ is stored in ACS-l. Continuing in this way, at time-h_i, for $1 \leq i \leq L$, ACS-l corresponding to state $\sigma^{(l)} \in \Sigma_{h_i}(C)$ will have the metric for $\sigma^{(l)}$ and a sequence of i survivor branch indices corresponding to the most likely path from the initial state to $\sigma^{(l)}$.

Whenever the decoder is processing the trellis at a level at which the size of the state space is smaller than $2^{\rho_{L,\max}(C)}$, a number of ACS circuits will be idle. If the number of inactive ACS circuits is large and occurs often during the decoding process, the hardware utilization efficiency becomes poor. For example, consider the minimal 8-section trellis for the $(64, 42)$ RM code, $\text{RM}_{3,6}$. This trellis has a state space dimension profile $(0, 7, 10, 13, 10, 13, 10, 7, 0)$ with $\rho_{8,\max}(C) = 13$. For a Viterbi decoder designed based on this trellis, at time-h_1 and -h_7, there are $2^{13} - 2^7 = 8,064$ inactive ACS circuits. At time-h_2, -h_4 and -h_6, there are $2^{13} - 2^{10} = 7,168$ inactive ACS circuits. Only at time-h_3 and -h_5, all the ACS circuits are active. We see that the hardware utilization efficiency is very poor for a Viterbi decoder for the $\text{RM}_{3,6}$ code based on the minimal 8-section trellis using the ACS-array architecture.

Hardware utilization efficiency can be improved by a proper parallel decomposition of a minimal trellis into parallel isomorphic subtrellises. The decomposition results in a non-minimal trellis with the same maximum state space dimension $\rho_{L,\max}(C)$. Therefore, the number of ACS circuits in the ACS-array is still the same, but the number of active ACS circuits is increased at many, if not all, section boundary locations. We illustrate this with an example. Using the method presented in Section 7.1, the minimal 8-section trellis for the $(64, 42)$ RM code, $\mathrm{RM}_{3,6}$, can be decomposed into a non-minimal 8-section trellis with 128 parallel isomorphic subtrellises, each having a state space dimension profile $(0, 6, 6, 6, 3, 6, 6, 6, 0)$. Therefore, the state space dimension profile for the overall trellis is $(0, 13, 13, 13, 10, 13, 13, 13, 0)$. We see that the maximum state space dimension is still $\rho_{8,\max}(C) = 13$. However, for a decoder based on this non-minimal trellis, all 8,192 ACS circuits are active all the time, except at time-h_4. This greatly improves the hardware utilization efficiency.

For a trellis (minimal or non-minimal) that consists of parallel subtrellises, all the subtrellis decoders operate independently in parallel without communication between them. From the standpoint of speed, the effective computational complexity of decoding a received sequence is defined as the computational complexity of a single parallel subtrellis (viz. the minimal trellis for a subcode C') plus the cost of the final comparison among the survivors presented by each of the subtrellis decoders. The time required for final comparison is generally small relative to the time required for processing a subtrellis and this comparison can be pipelined. Since all the subtrellises are processed in parallel, the speed of decoding is therefore limited only by the time required to process one subtrellis. If a minimal trellis does not have enough parallel structure and decoding speed is critical, parallel decomposition can be used to reduce the effective computational complexity and thus to gain speed.

10.3.2 Complexity of the ACS Circuit

The converging branch dimension profile (CBDP), $(\delta_1, \delta_2, \ldots, \delta_L)$, defined in Section 6.2 also affects decoding speed and complexity. Each component δ_i is the base-2 logarithm of the number of composite branches converging into a state at a particular level of the trellis. The number 2^{δ_i} is called a **radix number** in the IC literature. At level-i of the trellis, each ACS circuit has to perform δ_i stages of a tree-type two-way comparison to find the best incoming branch. Hence, a reduction in the values of the components in the CBDP of a trellis will improve decoding speed and reduce the complexity of each ACS circuit. If the radix numbers in a minimal trellis are too large, then parallel decomposition can be used to achieve smaller radix numbers, and hence to reduce the complexity of each ACS circuit and to increase the decoding speed.

10.3.3 Traceback Complexity

Even though the branch and state metrics are computed and updated in the Viterbi decoder at every level of the trellis, the best (or most likely) path through the trellis must be determined. The process of determining the best path is called **traceback** in the literature.

Recall that the number of parallel branches in a composite branch in the i-th section of an L-section trellis for C with section boundary locations in $\{h_0, h_1, \ldots, h_L\}$ is

$$\left|C^{\text{tr}}_{h_{i-1},h_i}\right| = 2^{k(C_{h_{i-1},h_i})}.$$

For a Viterbi decoder based on the minimal trellis, the ACS-l corresponding to state $\sigma^{(l)} \in \Sigma_{h_i}(C)$ for $1 \leq i \leq L$ must store $\delta_i(C) + k(C_{h_{i-1},h_i})$ bits in order to identify which of the $2^{\delta_i(C)}$ composite branches converging into state $\sigma^{(l)}$ is chosen and which of the $2^{k(C_{h_{i-1},h_i})}$ parallel branches survives. Therefore, each ACS-l needs to store

$$\sum_{i=1}^{L}(\delta_i(C) + k(C_{h_{i-1},h_i})) = K \qquad (10.2)$$

bits in order to identify the sequence of survived incoming branches and to determine the decoded path. If this number is too large, parallel decomposition can be used to reduce it. Consider a non-minimal trellis with 2^q parallel subtrellises obtained by parallel decomposition of the minimal L-section trellis for C based on a subcode C'. If a Viterbi decoder is designed based on this non-minimal trellis, then the number of bits that must be stored for each ACS-l is

$$\sum_{i=1}^{L}(\delta_i(C') + k(C'_{h_{i-1},h_i})) = \dim(C'). \qquad (10.3)$$

Since $\dim(C') = K - q \leq K$, the ACS circuits based on this non-minimal trellis design require less storage than for the design based on the minimal trellis. The total savings in storage in all the $2^{\rho_{L,\max}(C)}$ ACS circuits are

$$2^{\rho_{L,\max}(C)}(K - \dim(C')). \tag{10.4}$$

This is a significant savings.

10.3.4 ACS-Connectivity

The hardware implementation of a Viterbi decoder is severely affected by the physical placement of the ACS circuits and the need to route information between them. The routing complexity should be minimized in a Viterbi decoder IC design in order to reduce the size of the IC chip.

The basic operations performed by an ACS circuit are: addition of the branch metrics of the incoming branches to the state metrics of the corresponding originating states, comparison of the resulting sums to find the best one, selection of the surviving sum as the new state metric and the corresponding surviving branch label. The ACS-array architecture is usually dominated by the area required by the interconnections to transfer the state metrics. For a state $\sigma^{(l)} \in \Sigma_{h_i}(C)$ with $1 \leq l \leq |\Sigma_{h_i}(C)|$ and $0 \leq i < L$, let $Q_i(\sigma^{(l)})$ denote the set of states in $\Sigma_{h_{i+1}}(C)$ that are adjacent to $\sigma^{(l)}$. For $l > |\Sigma_{h_i}(C)|$, $Q_i(\sigma^{(l)}) = \emptyset$. Then in the ACS-array implementation of a Viterbi decoder, paths to transfer the state metrics exist between ACS-l and all the ACS circuits that correspond to the states in

$$Q_0(\sigma^{(l)}) \cup Q_1(\sigma^{(l)}) \cup \cdots \cup Q_{(L-1)}(\sigma^{(l)}). \tag{10.5}$$

The above set, denoted $\tilde{Q}^{(l)}$, defines the connectivity of ACS-l in the ACS-array corresponding to state $\sigma^{(l)}$. We call $|\tilde{Q}^{(l)}|$ and $\tilde{q}^{(l)} \triangleq \log_2 |\tilde{Q}^{(l)}|$ the connectivity and connectivity dimension of the ACS-l, respectively. The connectivities of ACS circuits determine the areas on an IC chip needed for wiring [10, 38]. This area should be kept as small as possible.

The ACS-connectivity can be reduced by using a non-minimal trellis with a proper number of parallel isomorphic subtrellises. With such a trellis, the ACS circuits can be divided into blocks such that the ACS circuits corresponding to states in a single subtrellis form a block. A particular ACS circuit only needs to transfer its metric to a subset of ACS circuits within its own block. This greatly reduces the ACS-connectivity and hence the hardware complexity and the wiring area on the IC chip.

10.3.5 Branch Complexity

The decoding speed of a Viterbi decoder depends on the total number of branches in the trellis to be processed and how fast they are being processed. If the processing load is shared by many ACS circuits at any time instant, then each ACS circuit will carry a small amount of processing load. This will speed up the decoding process. Therefore, a more meaningful measure of branch complexity is the number of branches to be processed by an ACS circuit [73, 101].

As pointed out earlier in this section, the number of active ACS circuits can be increased by parallel decomposition of a minimal trellis. However, parallel decomposition, in general, results in an increase in the number of composite branches in a trellis section. If the rate of increase of active ACS circuits is larger than the increase rate of composite branches, then the number of branches to be processed by each ACS circuit will decrease. The processing load of an ACS circuit at time-h_i is determined by the number of composite branches diverging from its corresponding state in $\Sigma_{h_i}(C)$. Therefore the total number of branches to be processed by an ACS circuit is determined by the diverging branch dimension profile (DBDP) of the trellis being used in the design.

Consider the minimal 8-section trellis of the $(64, 42)$ RM code, $\text{RM}_{3,6}$. The state space dimension profile of this code is $(0, 7, 10, 13, 10, 13, 10, 7, 0)$ and its DBDP is $(7, 6, 6, 3, 6, 3, 3, 0)$. Consider section-2 of the trellis. The number of composite branches in this section is 2^{13}. However, the number of active ACS circuits corresponding to the states of the trellis at the end of section-1 is $2^7 = 128$. Since each state has 64 composite branches diverging from it, each active ACS circuit must process 64 composite branches. Now consider the parallel decomposition of this minimal trellis into 128 parallel isomorphic subtrellises. The resultant non-minimal trellis has a state space dimension profile $(0, 13, 13, 13, 10, 13, 13, 13, 0)$ and each subtrellis has a state space dimension profile $(0, 6, 6, 6, 3, 6, 6, 6, 0)$. The DBDP of this non-minimal trellis is $(13, 3, 3, 3, 6, 3, 3, 0)$. All the components of this DBDP, except for the first one, are smaller than (or equal to) the corresponding components of the DBDP for the minimal 8-section trellis of this code. Consider section-2 of this non-minimal trellis. The total number of composite branches is now 2^{16}, a large increase from 2^{13} for the minimal trellis. However, all the 2^{13} ACS circuits at time-h_1 are active and they share the processing load. Each ACS circuit processes only 8 composite branches, compared to 64 for the minimal trellis. It is the same at the other time instants, except at time-h_4, where each active ACS circuit needs to process 64 branches, the same as for the minimal trellis. Therefore, the number of operations performed per ACS circuit is smaller for a Viterbi decoder designed based on the above non-minimal trellis. Reducing the diverging branch profile also results in a reduction of ACS-connectivity and hence a reduction in implementation complexity and wiring area on an IC chip.

Based on the above analysis and discussions, we may conclude that in designing a hardware Viterbi decoder for a specific linear block code, if the minimal trellis for the code is not desirable, then a non-minimal trellis with proper structural properties should be considered.

10.4 DIFFERENTIAL TRELLIS DECODING

The Viterbi algorithm was first devised for decoding convolutional codes. This decoding algorithm is based on the simple principle of add-compare-select (ACS) to process the code trellis and eliminate the less probable paths at each trellis level. This simple ACS principle has been used for implementing Viterbi decoders over the last two decades. However, a trellis-based decoding algorithm for convolutional codes can be devised based on a different processing principle, namely **compare-select-add (CSA)**. This decoding algorithm is devised based on a specific partition of a trellis section and the CSA processing principle. It is more efficient than the conventional Viterbi decoding algorithm. This decoding algorithm is called the **differential trellis decoding (DTD)** algorithm [32].

Consider a rate-$1/n$ $(n, 1, m)$ convolutional code of memory order m. The encoder of this code has one input and n outputs. Let $a = (a_0, a_1, \ldots, a_i, \ldots)$ be the input information sequence. The n corresponding output code sequences are

$$u^{(1)} = (u_0^{(1)}, u_1^{(1)}, \ldots, u_l^{(1)}, \ldots),$$

$$\begin{aligned} u^{(2)} &= (u_0^{(2)}, u_1^{(2)}, \ldots, u_l^{(2)}, \ldots), \\ &\vdots \\ u^{(n)} &= (u_0^{(n)}, u_1^{(n)}, \ldots, u_l^{(n)}, \ldots). \end{aligned}$$

At time-i, the input to the encoder is a_i and the output of the encoder is a block of n code bits $(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(n)})$. The trellis for this code consists of 2^m states with two branches entering and leaving each state at any time (or level) greater than m.

A rate-$1/n$ $(n, 1, m)$ convolutional code is said to be **antipodal** if, in the generator matrix of (9.7), $G_0 = G_m = [11\ldots1]$. Most of the best rate-$1/n$ convolutional codes are antipodal. For an antipodal convolutional code, the two branches entering (or leaving) a state in its code trellis are **one's complement** to each other, i.e., if one branch is labeled with $(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(n)})$, then the other branch is labeled with $(1 \oplus u_i^{(1)}, 1 \oplus u_i^{(2)}, \ldots, 1 \oplus u_i^{(n)})$, where $\oplus$ denotes the modulo-2 addition.

At time-i, the state of the encoder is defined by and labeled with the information bits $(a_{i-1}, a_{i-2}, \ldots, a_{i-m})$, stored in the input shift register. Consider the trellis section from time-i to time-$(i+1)$ for $i \geq m$. This section can be partitioned into 2^{m-1} two-state **fully connected subtrellises** with the following structural properties: (1) the two states at time-i are labeled with

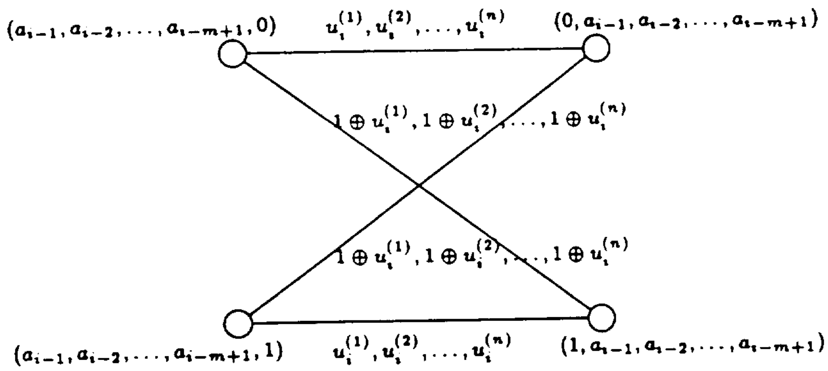
$(a_{i-1}, a_{i-2}, \ldots, a_{i-m+1}, 0)$
$u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(n)}$
$(0, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$
$1 \oplus u_i^{(1)}, 1 \oplus u_i^{(2)}, \ldots, 1 \oplus u_i^{(n)}$
$1 \oplus u_i^{(1)}, 1 \oplus u_i^{(2)}, \ldots, 1 \oplus u_i^{(n)}$
$(a_{i-1}, a_{i-2}, \ldots, a_{i-m+1}, 1)$
$u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(n)}$
$(1, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$

Figure 10.1. The structure of a 2-state subtrellis.

$(a_{i-1}, a_{i-2}, \ldots, a_{i-m+1}, 0)$ and $(a_{i-1}, a_{i-2}, \ldots, a_{i-m+1}, 1)$, respectively; (2) the two states at time-$(i+1)$ are labeled with $(0, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$ and $(1, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$, respectively; (3) the branches connecting the state $(a_{i-1}, a_{i-2}, \ldots, a_{i-m+1}, 0)$ to the states $(0, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$ and $(1, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$ are labeled with the code blocks $(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(n)})$ and $(1 \oplus u_i^{(1)}, 1 \oplus u_i^{(2)}, \ldots, 1 \oplus u_i^{(n)})$, respectively; and (4) the branches connecting state $(a_{i-1}, a_{i-2}, \ldots, a_{i-m+1}, 1)$ to the states $(0, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$ and $(1, a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$ are labeled with the code blocks $(1 \oplus u_i^{(1)}, 1 \oplus u_i^{(2)}, \ldots, 1 \oplus u_i^{(n)})$ and $(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(n)})$, respectively. The structure of such a two-state subtrellis is depicted in Figure 10.1. These 2^{m-1} fully connected subtrellises are commonly called "butterflies".

Based on the above state grouping and trellis partitioning between time-i and time-$(i+1)$, each subtrellis can be labeled by an $(m-1)$-tuple $\alpha = (a_{i-1}, a_{i-2}, \ldots, a_{i-m+1})$. In each subtrellis-$\alpha$, the states at time-$i$ and the states at time-$(i+1)$ are represented by (α, a_{i-m}) and (a_i, α), respectively, with $a_{i-m}, a_i \in \{0, 1\}$.

The decoding algorithm to be presented in the following is based on the above trellis partition. Assume that BPSK is used for transmission and each BPSK signal has unit energy. A code sequence is mapped into a bipolar signal sequence for transmission. The i-th code block $(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(n)})$ is mapped into the following bipolar sequence:

$$(2u_i^{(1)} - 1, 2u_i^{(2)} - 1, \ldots, 2u_i^{(n)} - 1). \tag{10.6}$$

Suppose correlation is used as the decoding metric. Let $r_i = (r_i^{(1)}, r_i^{(2)}, \ldots, r_i^{(n)})$ be the received block in the interval between time-i and time-$(i+1)$. It follows from properties (3), (4) of a butterfly subtrellis given above, that the four branch metrics between time-i and time-$(i+1)$ in subtrellis-α take two opposite values $\pm N_{i+1}^{\alpha}$, with

$$N_{i+1}^{\alpha} \triangleq \sum_{j=1}^{n} (2u_i^{(j)} - 1) \cdot r_i^{(j)}. \tag{10.7}$$

Let $M_i(\alpha, 0)$ and $M_i(\alpha, 1)$ denote the cumulative correlation metrics that have survived at time-i for states $(\alpha, 0)$ and $(\alpha, 1)$, respectively. Define

$$\Delta_{i+1}^{\alpha}(0, 1) \triangleq M_i(\alpha, 0) - M_i(\alpha, 1) \tag{10.8}$$

as the difference between these two metrics computed at time-$(i+1)$. Then at time-$(i+1)$, the difference between the cumulative metric candidates corresponding to transitions from states $(\alpha, 0)$ and $(\alpha, 1)$ to state (a_i, α) is given by

$$D_{i+1}^{\alpha}(a_i) = \Delta_{i+1}^{\alpha}(0,1) - 2(2a_i - 1)N_{i+1}^{\alpha} \tag{10.9}$$

for $a_i \in \{0, 1\}$.

Note that MLD maximizes the correlation metric. Hence, from (10.9), we conclude that at state (a_i, α) of subtrellis-α, we select the branch diverging from state $(\alpha, 0)$ if $\Delta_{i+1}^{\alpha}(0,1) > 2(2a_i - 1)N_{i+1}^{\alpha}$, and the branch diverging from state $(\alpha, 1)$ otherwise. Therefore, this decision can be made by first determining

$$|M_{\Delta,N}| = \max\{|\Delta_{i+1}^{\alpha}(0,1)|, |2N_{i+1}^{\alpha}|\}, \tag{10.10}$$

and then checking the sign of the value $M_{\Delta,N}$ corresponding to this maximum, denoted $\mathrm{sgn}(M_{\Delta,N})$. Based on the comparison result given in (10.10) and $\mathrm{sgn}(M_{\Delta,N})$, the selection of the surviving branches into states (a_i, α) with $a_i \in \{0, 1\}$ is made. All the four selections of surviving branches are shown in Figure 10.2. The selection rules are given below:

(1) If $|\Delta_{i+1}^{\alpha}(0,1)| > |2N_{i+1}^{\alpha}|$ and $\Delta_{i+1}^{\alpha}(0,1) > 0$, the two branches diverging from state $(\alpha, 0)$ into states $(0, \alpha)$ and $(1, \alpha)$ are selected as the surviving branches.

(2) If $|\Delta_{i+1}^{\alpha}(0,1)| > |2N_{i+1}^{\alpha}|$ and $\Delta_{i+1}^{\alpha}(0,1) \leq 0$, the two branches diverging from state $(\alpha, 1)$ into states $(0, \alpha)$ and $(1, \alpha)$ are selected as the surviving branches.

(3) If $|\Delta_{i+1}^{\alpha}(0,1)| \leq |2N_{i+1}^{\alpha}|$ and $2N_{i+1}^{\alpha} > 0$, the branch diverging from state $(\alpha, 0)$ into state $(0, \alpha)$ and the branch diverging from state $(\alpha, 1)$ into state $(1, \alpha)$ are selected as the surviving branches.

(4) If $|\Delta_{i+1}^{\alpha}(0,1)| \leq |2N_{i+1}^{\alpha}|$ and $2N_{i+1}^{\alpha} \leq 0$, the branch diverging from state $(\alpha, 0)$ into state $(1, \alpha)$ and the branch diverging from state $(\alpha, 1)$ into state $(0, \alpha)$ are selected as the surviving branches.

For each subtrellis-α, the decoding process from time-i to time-$(i+1)$ can be carried out as follows:

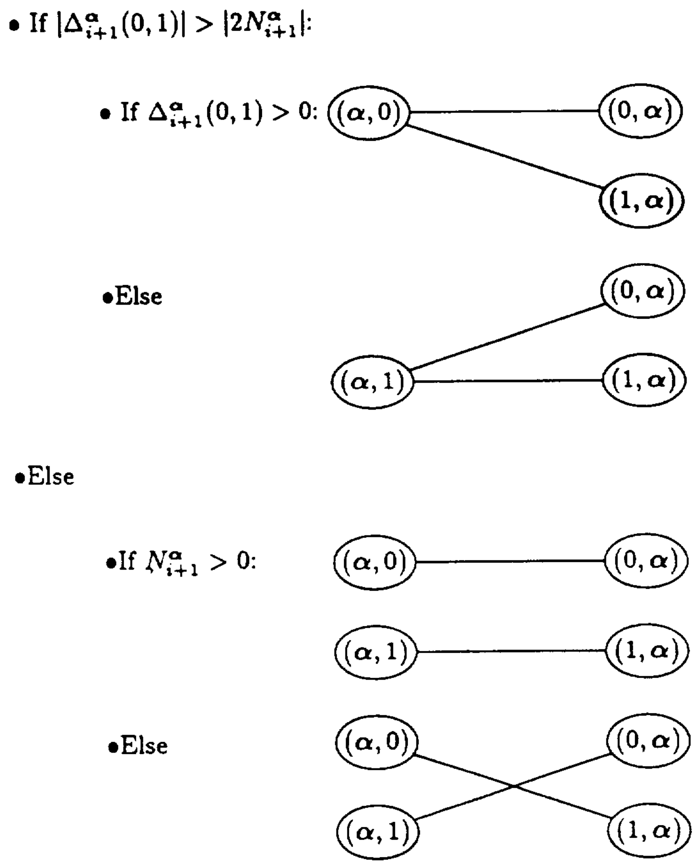

Figure 10.2. Branch selections for subtrellis-α.

Step-1 Compute the four possible branch metrics $\pm N_{i+1}^{\alpha}$ (preprocessing) and scale them by 2.

Step-2 From Step-1, identify $2N_{i+1}^{\alpha}$ and compute the metric difference $\Delta_{i+1}^{\alpha}(0,1)$.

Step-3 Compare $|\Delta_{i+1}^{\alpha}(0,1)|$ with $|2N_{i+1}^{\alpha}|$.

Step-4 Based on the comparison result of Step-3, determine either $\mathrm{sgn}(\Delta_{i+1}^{\alpha}(0,1))$ or $\mathrm{sgn}(N_{i+1}^{\alpha})$, and select the surviving branches based on the selection rule shown in in Figure 10.2.

Step-5 For each state at time-$(i+1)$, update the new survivor metric based on Step-4.

The above decoding algorithm is called the **differential CSA-algorithm**. The metric computations in Step-1, which are also performed by the conventional Viterbi algorithm, can be preprocessed since at most 2^{n-1} values must be computed. Also, if the branch from state (α, a_{i-m}) to state (a_i, α) survives, the surviving metric at state (a_i, α) in Step-5 can be computed as follows:

$$M_{i+1}(a_i, \alpha) = M_i(\alpha, a_{i-m}) + (2a_{i-m} - 1)(2a_i - 1)N_{i+1}^{\alpha}. \tag{10.11}$$

Note that the scaling by 2 of the preprocessed values $\pm N_{i+1}^{\alpha}$ at Step-1 and the sign checks at Step-4 are elementary binary operations (scaling is done by shifting the register once). The real number operations are performed at Step-2, Step-3, and Step-5. There are 2^{m-1} subtractions at Step-2, 2^{m-1} comparisons at Step-3 and 2^m additions at Step-5. Therefore **a total of $2 \cdot 2^m$ real number operations** is required to process a trellis section. However, after Step-1, the conventional Viterbi algorithm requires 2^{m+1} additions to evaluate the cumulative metrics for 2^m states and 2^m comparisons to determine the 2^m survivors. This results in **a total of $3 \cdot 2^m$ real number operations** to process a trellis section. As a result, the the differential CSA-algorithm requires about 1/3 less real number operations than the conventional Viterbi algorithm for rate-$1/n$ antipodal convolutional codes as well as high-rate punctured codes obtained from them.

Example 10.2 Consider the (2,1,6) convolutional code with generating pattern

$$[1 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 1 \;\; 1],$$

which is the most commonly used convolutional code. This code is antipodal. Its trellis consists of 64 states and can be decomposed into 32 fully connected 2-state subtrellises as shown in Figure 10.1. Since $n = 2$, at time-i, there are four possible branch metrics of the form $\pm(r_{i,1} \pm r_{i,2})$, which are computed with two real additions, and then scaled by 2. For this code, at time-i, the Viterbi algorithm computes 128 cumulative metric candidates and then performs 64 comparisons, so a total of 192 real value operations is required. The differential CSA-algorithm first computes 32 metric differences at Step-2, and then performs 32 comparisons at Step-3. Finally, based on the 32 sign checks of Step-4, 64 surviving cumulative metrics are updated at Step-5. As a result, only 128 real value additions are executed. Therefore, 64 real value operations are saved by the differential CSA-algorithm at the expense of 32 sign checks and 2 scalings by 2.

In practical applications, high-rate convolutional codes are often constructed from a low-rate $(n, 1, m)$ convolutional code by puncturing. The trellis for the punctured code has the same structure and state complexity as that of the original rate-$1/n$ convolutional code, except that the lengths of its sections vary periodically. As a result, the decoder for the rate-$1/n$ convolutional code can be used for decoding the punctured code. If the base rate-$1/n$ convolutional code is antipodal, then any punctured code constructed from it is also antipodal. Each trellis section for the punctured code can be partitioned into 2^{m-1} butterfly subtrellises in exactly the same manner as described above. The two branches leaving (or entering) a state in a butterfly subtrellis are one's complement of each other. Consequently, the differential CSA-algorithm can be used for decoding the punctured code. All the rate-$k/(k+1)$ punctured convolutional codes presented in [16] are time-varying antipodal codes. Also, this construction can be generalized to the case where k rate-$1/n$ base convolutional codes rather than only one are periodically selected, with period k. Again, if the resulting time-varying punctured code is antipodal, then the differential CSA-algorithm can be used.

The application of the differential CSA-algorithm to rate-k/n convolutional codes with $k > 1$ also allows 1/3 real value computation saving after proper pairing of the states in the code trellis [32]. The differential CSA-algorithm can also be applied efficiently to trellis decoding of block codes. For example, trellis decoding based on the 4-section trellis diagram for the (16, 5) RM code requires 59 real value operations for the differential CSA-algorithm and 95 real value operations for the conventional Viterbi algorithm.