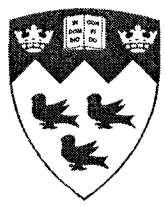
NOTE TO USERS

SOVA Based on a Sectionalized Trellis of Linear Block Codes

Sanja Kovacevic



Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

January 2004

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirements for the degree of Master of Engineering.

2004/01/26

Your file Votre référence
ISBN: 0-612-98538-5
Our file Notre référence
ISBN: 0-612-98538-5

Abstract

The use of block codes is a well known error-control technique for reliable transmission of digital information over noisy communication channels. However, a practically implementable soft-input soft-output (SISO) decoding algorithm for block codes is still a challenging problem.

This thesis examines a new decoding scheme based on the soft-output Viterbi algorithm (SOVA) applied to a sectionalized trellis for linear block codes. The computational complexities of the new SOVA decoder and of the conventional SOVA decoder based on the bit-level trellis are theoretically analyzed and derived. These results are used to obtain the optimum sectionalization of a trellis for SOVA. The optimum sectionalization of a trellis for Maximum A Posteriori (MAP), Maximum Logarithm MAP (Max-Log-MAP), and Viterbi algorithms, and their corresponding computational complexities are included for comparisons. The results confirm that SOVA based on a sectionalized trellis is the most computationally efficient SISO decoder examined in this thesis.

The simulation results of the bit error rate (BER) over additive white Gaussian noise (AWGN) channel demonstrate that the BER performance of the new SOVA decoder is not degraded. The BER performance of SOVA used in a serially concatenated block codes scheme reveals that the soft outputs of the proposed decoder are the same as those of the conventional SOVA decoder. Iterative decoding of serially concatenated block codes reveals that the quality of reliability estimates of the proposed SOVA decoder is the same as that of the conventional SOVA decoder.

Résumé

L'utilisation de codes linéaires en bloc est une technique bien connue qui permet la transmission numérique fiable dans des canaux de communication bruités. Cependant, il rest difficile d'implémenter en pratique des décodeurs à entrées et sorties souples pour les codeurs en blocs, du fait de leur complexité.

La présente thèse étudie un nouveau système de décodage des codes linéaires en bloc basé sur l'algorithme de Viterbi à sorties souples (SOVA), appliqué à un treillis divisé en sections. Les complexités algorithmiques associées à ce nouveau décodeur SOVA et au décodeur SOVA conventionnel sont dérivés dans ce travail. Ces résultats sont exploités afin d'obtenir la division en sections optimale des treillis considérés, pour l'application du SOVA. Des divisions optimales pour l'application des algorithmes MAP, Max-Log-MAP et Viterbi sont incluses à des fins de comparaison. Les résultats confirment que le SOVA basé sur un treillis en sections est le plus économique du point de vue calculs de tous les algorithmes à entrées et sorties souples considérés. Des simulations sur un canal BBGA démontrent que la division en sections ne dégrade pas la performance du SOVA du point de vue du taux d'erreurs binaires. L'tude du taux d'erreur binaire du SOVA utilis dans un shcma de correction d'erreur par codes concatns en srie dmontre que la qualit des sorties douces gnres par notre algorithme est similaire celle du SOVA classique. La mme conclusion peut tre tire d'une tude des performances dans un dcodage itratif.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Fabrice Labeau, for his guidance, suggestions, and continuous support throughout my graduate studies. Special thanks also goes to the Canadian Institute for Telecommunications Research (CITR) and to the Canadian Space Agency (CSA) for sponsoring this research project.

I would also like to extend my thanks to fellow researchers and close friends: Tania, Karim, Martin, Alex, Youssef, Ricky, Kamal, Dave, Fred, and Eric, for all your useful suggestions, great friendships, and the fun research environment you have created.

Special thanks to my parents and Bebo for all their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Error control coding is essential for a reliable data transmission in all data communication systems. The use of block codes is a well known error-control technique. Many theorists have investigated soft-input soft-output (SISO) decoding of block codes. However, a computationally efficient and practically implementable SISO algorithm for block codes is still a challenging problem.

The theoretical fundamental limit on the transmission rates in the digital communication systems have been established by Shannon [1]. The recently discovered Block Turbo Codes (BTCs) are able to achieve an error performance near this Shannon limit [2].

Motivated by the search for a computationally efficient decoding algorithm for the BTCs, the objective of this thesis is to establish a low complexity near optimum SISO algorithm that can be implemented in the component decoders of the BTC loop. This is accomplished through the examination of the new decoding scheme based on the trellis-based soft-output Viterbi algorithm (SOVA) applied to a sectionalized trellis for linear block codes.

The sectionalization method allows the decoder to output more code bits in one trellis section. This can reduce the number of computations required by the decoder to process a received signal. The computation-wise optimum sectionalization algorithm provides a sectionalized trellis for which a minimum number of computations are required by a decoder. This algorithm has been devised for Viterbi decoding in [3], and has been applied to MAP and Max-Log-MAP decoding in [4].

This thesis provides the optimum sectionalizations for SOVA decoding of different linear block codes and the corresponding computational complexity of the decoder. The computational

complexity measures only the number of operations required by the algorithm to decode a signal. Storage and memory requirements, decoding delay and speed, and the addressing issues are not taken into consideration. The performance of the proposed SOVA decoder, measured in terms of its BER for additive white Gaussian noise (AWGN) environment, is also provided. The obtained results are compared to those of the conventional SOVA decoder based on a bit-level trellis, as well as to the already investigated sectionalized trellis-based MAP, Max-Log-MAP, and Viterbi algorithms. Similar BER performance evaluations are provided for the concatenated scheme using SOVA, MAP, and Max-Log-MAP as the inner decoders and Viterbi as the outer decoder. Results of iterative SOVA decoding of concatenated block codes are also provided and discussed.

This chapter begins with the system model of the proposed decoding scheme. This is followed by the review of linear Reed-Muller block codes, in Section 1.2. The purpose of Section 1.3 is to give the reader an overview of trellis decoding. Next, in Section 1.4, the principal elements of trellis sectionalization are explained. The subsequent two sections explain maximum-likelihood (ML) and soft-input soft-output (SISO) decoding. Block Turbo Coding, the motivation behind this thesis, is described in Section 1.7. Section 1.8 presents some of the earlier work done related to the complexity reduction methods. Section 1.9 states the contribution of this thesis made in establishing a low-complexity near-optimum SISO algorithm. Section 1.10 gives the organization of the rest of the thesis.

1.1 System Model

The system model built in this thesis is illustrated in Figure 1.1.

The encoder encodes an information sequence, $\mathbf{m} = \{m_1, m_2, \ldots, m_K\}$, of K bits into a code sequence, $\mathbf{c} = \{c_1, c_2, \ldots, c_N\}$, of N bits. It is assumed that all information bits are equally likely. Each code bit, formed and transmitted by the encoder, depends only on the current state of the encoder, defined by a finite set, and the current input information bit. As such, the encoder is modelled as a finite state Markov process. This encoder and its generated codewords are graphically represented by a trellis, as discussed in Section 1.3.

The binary phase - shift keying (BPSK) modulator maps the code sequence, $\mathbf{c}$, into a bipolar sequence represented by $\mathbf{u} = \{u_1, u_2, \ldots, u_N\}$, where $u_i = 2c_i - 1$ for $1 \leq i \leq N$. Hence, the components of the codewords are mapped from $\{0, 1\}$ to $\{-1, +1\}$ in $\mathbf{u}$.

The channel is the physical medium that connects the transmitter and the receiver. As the signal is transmitted through the channel, it is distorted due to channel imperfections. Noise is

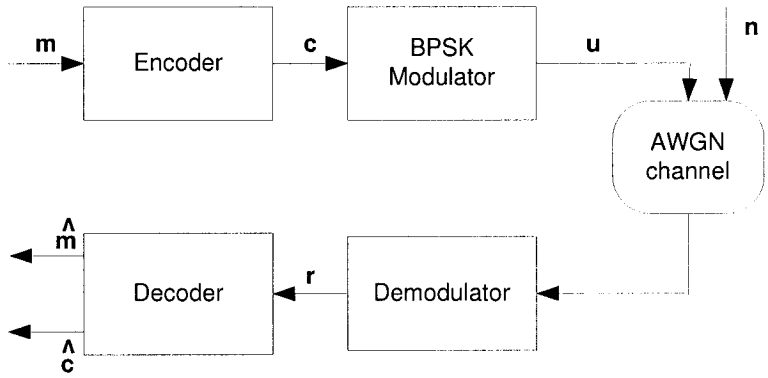


Fig. 1.1 System Model.

added to the channel output, resulting in a corrupted version of the transmitted signal. In this thesis, the additive white Gaussian noise (AWGN) channel is used as the transmission media. The transmitted signal is distorted by an additive, stationary, white, Gaussian noise $\mathbf{n}$, a sequence of N Gaussian random variables of zero-mean and variance $N_0/2$.

At the receiving end, the demodulator passes the unquantized received signal sequence, $\mathbf{r} = \mathbf{u} + \mathbf{n}$, to a decoder for processing. A trellis is used to decode linear codes by applying trellis-based Viterbi, SOVA, Max-Log-MAP, or MAP decoding algorithms. The estimate of the transmitted codeword, $\hat{\mathbf{c}}$, based on the trellis-based decoding algorithm is provided. For all, but Viterbi algorithm, the reliabilities of symbols in $\hat{\mathbf{c}}$ are also provided. The estimated information sequence, $\hat{\mathbf{m}}$, is obtained from $\hat{\mathbf{c}}$ using the inverse mapping of the one used by the encoder, between information sequences and the codewords.

1.2 Reed-Muller Codes

Reed-Muller (RM) codes, introduced by Muller in 1954 [5], are systematic linear block codes. The first decoding algorithm for these codes was devised by Reed in the same year [6]. Their trellises are easily constructed, enabling these codes to be decoded effectively with trellis-based decoding algorithms.

The RM (r, m) code of order r, where $0 \leq r \leq m$, can be defined by the generated vectors represented by the Boolean polynomials of degree r or less in m variables, [7], with the following

parameters:

Code Length:	$N = 2^m$
Message Length:	$K = 1 + \binom{m}{1} + \cdots + \binom{m}{r}$
Minimum Distance:	$d = 2^{m-r}$

The 2^K possible codewords, corresponding to the 2^K possible messages, form a block code of length N, denoted as (N, K), generated by a $K \times N$ generator matrix. It is common to describe the RM (r, m) code with the notation RM (N, K), as is done in this thesis.

A general generator matrix, $G_{(N,N)}$, for all RM codes with the same parameter m is the m-fold Kronecker product of the base matrix $G_{(2,2)}$, where a Kronecker product of general matrices $A = [A_{ij}]$ and $B = [B_{ij}]$ is defined as:

$$A \otimes B = \begin{bmatrix} A_{00}B & \cdots & A_{0N}B \\ \vdots & & \vdots \\ A_{N0}B & \cdots & A_{NN}B \end{bmatrix}.$$

For RM codes of specific order r, the generator matrix is formed from the rows of $G_{(N,N)}$ with weights equal to or greater than the minimum distance, 2^{m-r}, of the RM code. The encoding is performed based on the reduced echelon form (REF) of this matrix.

For example, the general generator matrix, $G_{(8,8)}$, for RM codes of length eight, is the 3-fold Kronecker product of the base matrix,$G_{(2,2)}$:

$$G_{(8,8)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

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

where the base matrix $G_{(2,2)}$ is:

$$G_{(2,2)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

For the first order RM code of length eight, RM (8,4) code, the generator matrix is the REF of the matrix obtained from the rows of $G_{(8,8)}$ with weights equal to or greater than 2^2:

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

The 2^K linear combinations of the K rows form the codewords. The codeword $\mathbf{c} = \{c_1, c_2, \ldots, c_N\}$ for the message $\mathbf{m} = \{m_1, m_2, \ldots, m_N\}$ is given by

$$\begin{aligned} \mathbf{c} &= \mathbf{m} \cdot \mathbf{G} \\ &= [m_1, m_2, \ldots, m_K] \cdot \begin{bmatrix} \mathbf{g_1} \\ \mathbf{g_2} \\ \vdots \\ \mathbf{g_K} \end{bmatrix} \\ &= m_1 \cdot \mathbf{g_1} + m_2 \cdot \mathbf{g_2} + \ldots + m_K \cdot \mathbf{g_K}. \end{aligned} \tag{1.1}$$

where $\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g_K}$ represent the row vectors of the generator matrix.

As RM codes are linear systematic block codes, the information bits received by the encoder are part of the generated codewords. Their positions in the codeword correspond to the K columns of the $\mathbf{I_K}$ identity matrix in $\mathbf{G}$, with the same order of appearance.

1.3 Trellis Decoding

A trellis, introduced for linear block codes by Bahl *et al.* in [8], graphically represents the encoder by a state diagram expanded in time. All the codewords of any linear block code can be represented by a trellis.

A trellis T is composed of $N + 1$ time instants, numbered from 0 to N that represent one encoding interval and that border N sections, numbered from 1 to N, corresponding to N bit

intervals. It is defined as $T = (S, A, B)$, where S is the set of possible states of the encoder, A is the set of possible encoder outputs represented by the code bits in branch labels, and B is the set of branches in the trellis. An example of the trellis representation of encoder is shown in Figure 1.2.

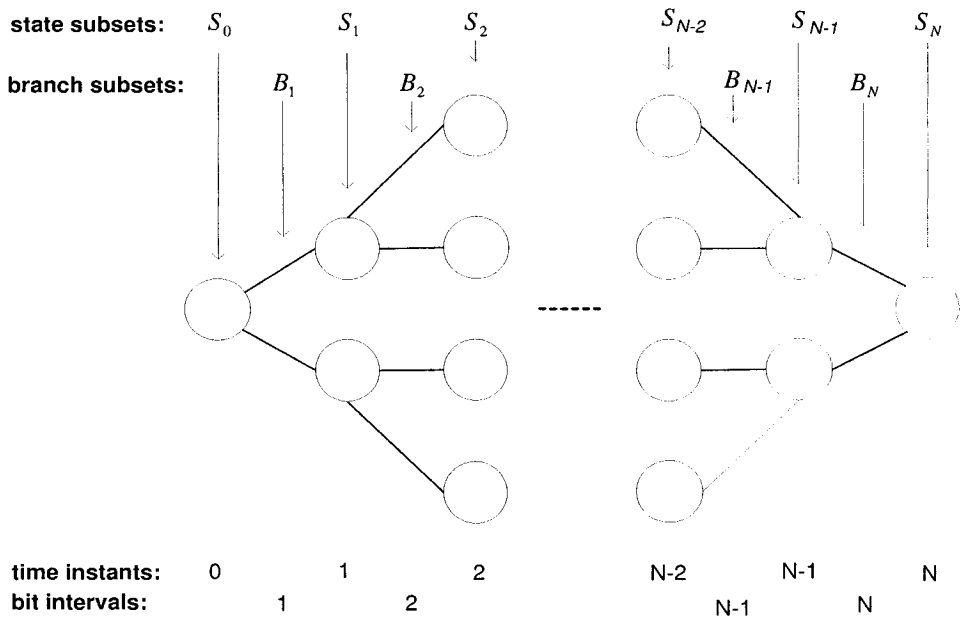


Fig. 1.2 Trellis Representation of the Encoder.

The set S is partitioned into disjoint subsets S_i, for each time instant i, containing a possible state of encoder after transmitting the ith bit. The subsets S_0 and S_N, each consist of a single state, σ_0 and σ_f, respectively. The set B is also partitioned into disjoint subsets B_i, for each section $i = 1$ to N, containing branches $(\sigma, \sigma', \alpha)$ that connect states $\sigma \in S_{i-1}$ to states $\sigma' \in S_i$, with label $\alpha \in A$ corresponding to the encoder output of that transition. For a bit level trellis that represents binary codes, every state has at least one, but no more than two incoming and outgoing branches, except for the initial state, σ_0, which has no incoming branches and for the final state, σ_f, which has no outgoing branches. Every path in the forward direction connecting σ_0 to σ_f represents a codeword.

During each encoding interval, as the encoder transverses a sequence of states $(\sigma_0, \sigma_1, \ldots, \sigma_i, \ldots, \sigma_f)$, by branches with a label sequence $\mathbf{c} = \{c_1, c_2, \ldots, c_i, \ldots, c_N\}$, representing a codeword, K information bits are encoded into N code bits, and shifted onto the

channel in N bit intervals.

The first study of trellis construction and structure for linear block codes was presented by Wolf in 1978 [9]. However, at that time, it was believed that block codes did not have simple trellis structures and that ML decoding of linear block codes was practically impossible except for very short block codes. There was not much research in the trellis structure of linear block codes until Forney showed, in 1988, that some block codes, such as RM codes, have simple trellis structures [10].

1.4 Trellis Sectionalization

It is possible to sectionalize a bit-level trellis with section boundaries at selected instants in the encoding interval, as is shown in Figure 1.3. The encoder can form and transmit more than one code bit in the resulting intervals. Considering only a subset of time instants $\{h_0, h_1, h_2, \ldots, h_v\}$ where $0 = h_0 < h_1 < \ldots < h_v = N$, the number of sections is reduced. The resulting trellis consists of v sections, where $1 \leq v \leq N$. Each section, T_j, is $h_j - h_{j-1}$ bit intervals long.

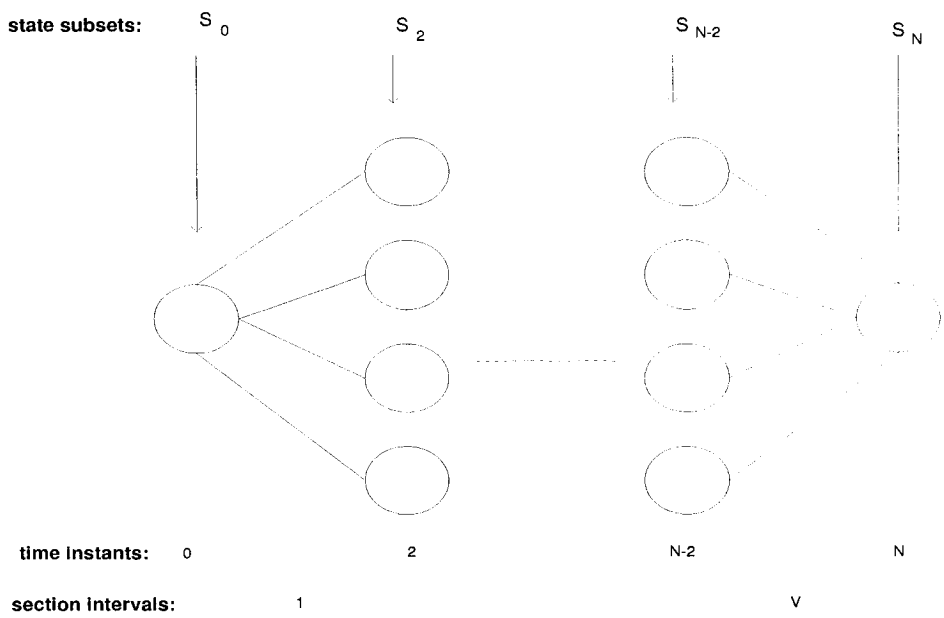


Fig. 1.3 A Sectionalized Trellis Representation.

The sectionalization is achieved by:

1. deleting all states, belonging to the state subsets that are not at the new boundary locations, and all their incoming and outgoing branches, and

2. connecting states at the adjacent new boundary locations by multiple branches corresponding to the paths that connect these states in a bit-level trellis. These branches, known as parallel branches, are therefore labelled by an ℓ_j-tuple, where $\ell_j = h_j - h_{j-1}$. A composite branch represents the set of parallel branches between each pair of adjacent states.

A trellis representation makes it possible to implement maximum-likelihood and SISO decoding of a code with reduced decoding complexity.

1.5 Maximum-Likelihood Decoding

Maximum-likelihood (ML) sequence decoding minimizes the code sequence error probability, $P(E)$, for equally probable transmitted codewords. The decoding is achieved by finding the most likely transmitted codeword given what was received.

If $\hat{\mathbf{c}}$ is the estimate of the transmitted codeword sequence $\mathbf{c}$, a decoding error occurs if $\hat{\mathbf{c}} \neq \mathbf{c}$, and is defined as

$$P(E) = \sum_{\mathbf{r}} P(\hat{\mathbf{c}} \neq \mathbf{c}|\mathbf{r})P(\mathbf{r}) \tag{1.2}$$

Minimizing $P(E)$ is equivalent to minimizing $P(\hat{\mathbf{c}} \neq \mathbf{c}|\mathbf{r})$ or maximizing $P(\hat{\mathbf{c}} = \mathbf{c}|\mathbf{r})$, and by Bayes theorem:

$$P(\mathbf{c}|\mathbf{r}) = \frac{P(\mathbf{r}|\mathbf{c})P(\mathbf{c})}{P(\mathbf{r})} \tag{1.3}$$

Since each codeword is equiprobable, $P(\mathbf{c})$ is a constant and can be omitted from (1.3). Hence, in order to minimize $P(E)$ in (1.2), the decoder selects the estimate of the transmitted code sequence, $\hat{\mathbf{c}}$, that maximizes $P(\mathbf{r}|\hat{\mathbf{c}})$.

This type of decoding outputs only estimated code bits, also called hard outputs, without providing their reliability measures. The ML decoding can be implemented by applying the Viterbi algorithm to the trellis.

1.5.1 Viterbi algorithm

The Viterbi algorithm [11],[12],[13] computes $P(\mathbf{r}|\mathbf{c})$ for all code sequences entering each state in a trellis and selects the maximum. This is done by first computing all the branch probabilities, $\bar{P}(b_i(\sigma, \sigma'))$, representing the probability of the transition of the encoder from state $\sigma \in S_{i-1}$ to state $\sigma' \in S_i$ through the branch $b_i(\sigma, \sigma')$ associated with r_i, in each trellis section, and expressed as:

$$\bar{P}(b_i(\sigma, \sigma')) \propto \frac{2}{N_0} r_i u_i$$

Next, all the state probabilities in the forward recursion, $\bar{\alpha}_i(\sigma')$, for time instants $i = 0$ to $i = N$, are calculated as:

$$\bar{\alpha}_i(\sigma') = \max_{\sigma \in \Omega_{i-1}(\sigma')} \{\bar{P}(b_i(\sigma, \sigma')) + \bar{\alpha}_{i-1}(\sigma)\}$$

At the final state, the code sequence that provides the maximum probability is the maximum likelihood (ML) path. The decoder outputs the bits corresponding to the most likely transmitted codeword that correspond to the branch labels of this ML path. The reliability measures of the estimated code bits are not provided.

Since the Viterbi algorithm minimizes the sequence error probability, it is optimum in terms of the word error rate (WER). However, it does not minimize the bit error probability, and is, therefore, only suboptimum with respect to the bit error rate (BER).

1.6 Soft-Input Soft-Output Decoding

In many error-control coding schemes, it is desirable to provide estimated bits as well as their reliabilities, also called soft outputs, for further processing. The soft outputs can be used as an input to another decoder in a concatenated scheme or in iterative decoding, such as turbo coding.

SISO algorithms, such as MAP, Max-Log-MAP, and SOVA, provide the soft information associated with the decision on each code bit c_i, based on the received sequence $\mathbf{r} = \{r_1, r_2, \ldots, r_N\}$, in the form of the log-likelihood ratio (LLR):

$$L(\hat{c}_i) = \log \frac{P(c_i = 1|\mathbf{r})}{P(c_i = 0|\mathbf{r})}, \qquad \text{for } 1 \leq i \leq N, \tag{1.4}$$

where $P(c_i = b|\, mathbfr)$, $b \in \{1, 0\}$, represents the a posteriori probability (APP) of the

transmitted code bit.

The equations for generating this soft output information for MAP, Max-Log-MAP, and SOVA decoders are defined in Section 2.2 for bit level trellises, and in Section 3.2 for sectionalized trellises.

The decoders output each estimated code bit, $\hat{c}_i$, by comparing its LLR value, $L(\hat{c}_i)$, to a threshold of zero:

$$\hat{c}_i = \begin{cases} 1 & , \quad \text{for } L(\hat{c}_i) > 0, \\ 0 & , \quad \text{for } L(\hat{c}_i) \leq 0. \end{cases}$$

1.6.1 Maximum A Posteriori Algorithm

The Maximum A Posteriori (MAP) algorithm was developed by Bahl *et al.* in 1974 [8]. MAP algorithm computes all the branch probabilities in each trellis section as:

$$P(b_i(\sigma, \sigma')) \propto \exp\{\frac{2}{N_0} r_i u_i\}$$

Next, the state probabilities in the forward recursion, $\alpha_i(\sigma')$, and in the backward recursion, $\beta_{i-1}(\sigma)$, are calculated as:

$$\alpha_i(\sigma') = \sum_{\sigma \in \Omega_{i-1}(\sigma')} P(b_i(\sigma, \sigma'))\alpha_{i-1}(\sigma)$$

$$\beta_{i-1}(\sigma) = \sum_{\sigma' \in \Psi_i(\sigma)} P(b_i(\sigma, \sigma'))\beta_i(\sigma')$$

To obtain the soft output, MAP considers all paths in each trellis section, and partitions them into two sets corresponding to the two possible encoder outputs. The soft output of (1.4) for MAP is defined as the ratio of the probabilities of these two sets in the logarithmic domain as:

$$L(\hat{c}_i) = \log \frac{\sum\limits_{\substack{(\sigma,\sigma') \\ c_i=1}} \alpha_{i-1}(\sigma)P(b_i(\sigma, \sigma'))\beta_i(\sigma')}{\sum\limits_{\substack{(\sigma,\sigma') \\ c_i=0}} \alpha_{i-1}(\sigma)P(b_i(\sigma, \sigma'))\beta_i(\sigma')}$$

Although optimal in terms of BER performance, the MAP algorithm requires a large number of computations, and as such, is too complex for implementations in many communication sys-

tems. Approximations of the MAP algorithm, such as ML SOVA and Max-Log-MAP, have been derived to reduce the number of operations, but are suboptimal in terms of BER.

1.6.2 Maximum Logarithm MAP Algorithm

The Maximum Logarithm MAP (Max-Log-MAP) algorithm [14] is an approximation of the MAP algorithm that operates in the logarithmic domain. As such, its computational complexity is much lower than that of MAP algorithm. However, due to its approximation of (1.4), it is suboptimum in terms of BER.

First all the branch probabilities in each trellis section are computed as for the Viterbi algorithm. Next, the state probabilities in the forward recursion, $\bar{\alpha}_i(\sigma')$, and in the backward recursion, $\bar{\beta}_{i-1}(\sigma)$, are calculated as:

$$\bar{\alpha}_i(\sigma') = \max_{\sigma \in \Omega_{i-1}(\sigma')} \{\bar{P}(b_i(\sigma, \sigma')) + \bar{\alpha}_{i-1}(\sigma)\}$$

$$\bar{\beta}_{i-1}(\sigma) = \max_{\sigma' \in \Psi_i(\sigma)} \{\bar{P}(b_i(\sigma, \sigma')) + \bar{\beta}_i(\sigma')\}$$

The Max-Log-MAP decoding algorithm obtains the soft output by considering two ML paths, corresponding to the two possible encoder outputs, in each trellis section. The soft output is approximated as the difference between the probabilities of these two paths as:

$$\begin{aligned} L(\hat{c}_i) \approx & \max_{\substack{(\sigma,\sigma') \\ c_i=1}} \{\bar{\alpha}_{i-1}(\sigma) + \bar{P}(b_i(\sigma, \sigma')) + \bar{\beta}_i(\sigma')\} \\ & - \max_{\substack{(\sigma,\sigma') \\ c_i=0}} \{\bar{\alpha}_{i-1}(\sigma) + \bar{P}(b_i(\sigma, \sigma')) + \bar{\beta}_i(\sigma')\} \end{aligned}$$

1.6.3 Soft-Output Viterbi Algorithm

The soft-output Viterbi algorithm (SOVA) is a modified Viterbi algorithm that also provides the approximate soft output for each estimated code bit [15],[16]. The version of SOVA examined in this thesis is presented in [17]. The algorithm is based in the logarithmic domain providing only the approximation of (1.4). As such, it is suboptimum with respect to BER.

All the branch probabilities, and the state probabilities in the forward and in the backward recursions are computed the same way as for the Max-Log-MAP algorithm.

To obtain the soft output, SOVA considers two paths in each trellis section: the ML path associated with that section, and the most probable path that is complementary to the ML path.

Unlike Max-Log-MAP algorithm, SOVA only guarantees to find one best path, the ML path. The other path is not necessarily the best one in its set, due to the fact that the best path with the complementary bit to the ML path may be discarded before it merges with the ML path. The soft output is approximated as the difference between the probabilities of these paths as:

$$L(\hat{c}_i) \approx \bar{P}(c_i = 0|\mathbf{r}) - \bar{P}(c_i = 1|\mathbf{r})$$

where the most probable path whose label is complementary, x, to the ML estimate is obtained as:

$$\bar{P}(c_i = x|\mathbf{r}) = \max_{\substack{(\sigma,\sigma') \\ c_i = x}} \{\bar{\alpha}_{i-1}(\sigma) + \bar{P}(b_i(\sigma,\sigma')) + \bar{\beta}_i(\sigma')\}$$

1.7 Block Turbo Coding

Turbo error-control coding was introduced in 1993 by Berrou *et al.* [18], [19]. The conventional scheme consists of two recursive systematic convolutional codes concatenated in parallel, called Convolutional Turbo Codes (CTC). In 1994, Block Turbo Codes (BTC) were presented by Pyndiah *et al.* [2], [20], [21]. They are constructed from product codes, introduced by Elias in 1954 [22], built using linear block codes. In both schemes, the component codes are decoded using SISO decoding algorithms. It has been shown that CTCs and BTCs achieve performance close to Shannon's theoretical limit [1] on an AWGN channel. Due to such superior performance, CTCs have been proposed for many communication systems, such as deep space, cellular mobile and satellite communication networks [23], [24], [25]. They are especially useful for mobile wireless applications to overcome channel fading and have been approved for the 3rd generation IMT2000 mobile systems, such as cdma2000 [26].

BTCs possess several advantages over CTCs. The results in [2] indicate that BTCs are the most efficient known codes for high code rate applications. In [27], it is shown that BTCs are more efficient than CTCs for small data blocks used in time-division multiple access (TDMA) applications. For applications especially in cell-based transmission, BTCs are more suitable than CTCs because of the fixed cell size. They have been proposed for satellite asynchronous transfer mode (ATM) applications [28]. For BTCs to be used in a wider range of applications, it is desirable to have a practically implementable SISO decoder.

1.8 Previous Related Work

For BTCs, several lower complexity decoders have been suggested. The Chase algorithm, proposed in [29], was applied to BTCs by Pyndiah in [20], [2]. This algorithm is based on reviewing only the most probable codewords located in the sphere of radius equal to the minimum distance of the linear block codes. In [30], Kaneko's algorithm was applied. This algorithm is similar to the Chase algorithm in the manner in which it generates the candidate codewords; the difference is that the most likely codeword is definitely included in those codewords. SOVA based on a bit-level trellis was applied to BTCs in [31]. All these algorithms were shown to offer a compromise between BER performance and complexity of BTCs.

For CTCs, several complexity reduction schemes have also been presented. Luukkanen and Zhang in [32], examined the performance and complexity of CTCs using Max-Log-MAP and SOVA decoding schemes based on bit-level trellises. Their simulation results show that although the use of these decoders slightly degrades the performance of turbo codes, the computational complexity is significantly reduced. SOVA based on a bit-level trellis was also applied to CTCs in [26] and [33] and was proven to have lower complexity than the conventional MAP decoder, with suboptimal BER performance.

1.9 Thesis Contribution

The focus of this thesis is the new SOVA decoding scheme based on a sectionalized trellis. The decoder's BER performance and its complexity, a measure of the number of required computational operations, are examined.

The techniques for computationally efficient implementation of Viterbi algorithm, presented in [3], and of MAP and Max-Log-MAP algorithms, presented in [4], based on a bit-level trellis and on a sectionalized trellis are reviewed. Using similar techniques, the computational complexity of the proposed SOVA decoder and of the conventional SOVA decoder, based on a bit-level trellis, are theoretically analyzed. The comparisons of these complexities for different linear block codes show that the new SOVA decoder is significantly less computationally complicated than the conventional one.

The computation-wise optimum sectionalization algorithm is applied to SOVA for different linear block codes. The resulting optimum sectionalizations and the corresponding computational complexities are compared to those of Viterbi, MAP, and Max-Log-MAP decoders. The results

corroborate the fact that the proposed SOVA is the most computationally efficient SISO decoding algorithm examined in this thesis .

The performance of the proposed SOVA decoder is obtained through BER simulations over AWGN channel for different RM codes. The comparison with the performance of the conventional SOVA decoder reveals that sectionalization does not degrade the algorithm's performance. The performances of Viterbi, MAP, and Max-Log-MAP algorithms based on a sectionalized trellis are also included for comparisons. The BER performance of using the new SOVA as the inner decoder in the concatenated block code scheme is examined and compared to that obtained using the conventional SOVA as the inner decoder. These are further compared to the BER performances of using MAP and Max-Log-MAP as the inner decoders and Viterbi as the outer decoder in a concatenated scheme based on a bit-level trellis and on a sectionalized trellis. The BER performance of iterative decoding of concatenated block codes using the proposed SOVA is provided and compared to that of applying the conventional SOVA to the component decoders.

1.10 Thesis Organization

Chapter 2 outlines the construction of the minimal bit-level trellis for linear block codes. The fundamental concepts of the trellis construction are explained. These include: the formation of the trellis oriented generator matrix; trellis complexity measures, such as state and branch complexities; state labelling and transition. The chapter further presents the computational complexities of the decoding algorithms applied to a bit-level trellis. The assumptions employed are specified, and techniques for the computationally efficient implementation of the decoding steps of all algorithms considered are explained.

Chapter 3 studies the sectionalization of a trellis. The sectionalized trellis complexity, in terms of its structural properties, including state and branch complexities, is examined. Next, the possible state transitions of the encoder and the corresponding generated code bits, are provided. The following section provides the sectionalization algorithm that yields the optimum section boundaries. The chapter next reviews the required computational complexities of the decoding algorithms applied to a sectionalized trellis. The complexities of Viterbi, MAP, and Max-Log-MAP algorithms are examined. Using similar methods, the computational complexity of SOVA applied to a sectionalized trellis is attained.

Chapter 4 presents the optimum trellis sectionalizations for the algorithms of interest for different RM codes. The computational complexities of SOVA are compared to the much larger

ones required for the decoding of a bit-level trellis. These results are further compared with those obtained for MAP, Max-Log-MAP and Viterbi algorithms. The simulations of the BER performances, over AWGN channel, for SOVA based on a bit-level trellis and based on a sectionalized trellis are discussed. The performances of Viterbi, MAP, and Max-Log-MAP algorithms applied to bit-level trellises and to sectionalized trellises are included for comparisons. The chapter further evaluates the BER performances of using SOVA, MAP, and Max-Log-MAP as the inner decoders and Viterbi as the outer decoder in a concatenated scheme based on a bit-level trellis and on a sectionalized trellis. Also included, are the BER performances of using the proposed SOVA and the conventional SOVA as the component decoders in an iterative decoding of concatenated block codes scheme.

Chapter 5 presents a summary of the thesis contributions, the concluding remarks, along with ideas for the potential future work.

Appendix A provides the discussion on the differences with the best results found in the literature.

Appendix B provides more simulation results of the BER performances of SOVA, MAP, Max-Log-MAP, and Viterbi algorithms based on a bit-level trellis and on a sectionalized trellis for different RM codes.

Chapter 2

Bit-Level Trellis

Every linear block code can be represented by a trellis, a state diagram expanded in time that is used as a template for ML and soft decision decoding. McEliece in [34] constructed a trellis that has minimum complexity measures, such as the number of states and branches. This trellis is also known as a minimal trellis.

The first part of this chapter, Section 2.1, outlines the construction of the minimal bit-level trellis using a special form of the generator matrix for linear block codes and its structural properties. The second part of this chapter, Section 2.2, reviews the computational complexity of Viterbi, Max-Log-MAP, and MAP algorithms and obtains the complexity of SOVA for decoding a bit-level trellis. The chapter concludes with the summary of the results.

2.1 Bit-Level Trellis Construction

In this thesis, a minimal trellis for a linear block code was constructed, using a trellis oriented generator matrix that is explained in the next section. The following section, Section 2.1.2, explains how to obtain the fundamentals of the minimal bit-level trellis, such as its state and branch complexities. These are necessary to determine the possible encoder outputs, represented by the branch labels in a trellis, that are explained in Section 2.1.3.

2.1.1 Trellis Oriented Generator Matrix

The construction of a minimal trellis is accomplished by using the generator matrix in a trellis oriented form, called trellis oriented generator matrix (TOGM) [35]. This is done by employing

the following two conditions on the generator matrix of the code:

1. The first nonzero component of each row appears in a column before that of any row below it.

2. No two rows have their last nonzero component in the same column.

By applying elementary matrix operations, the generator matrix of RM (8,4) code from Section 1.2, is put in the TOGM form:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.1}$$

Next, the complexity measures of the minimal trellis representation of the code are derived from the TOGM. These are necessary in constructing the trellis and are useful in the computation of the decoding complexity of a trellis.

2.1.2 Trellis Complexity

The complexity of a bit-level trellis for a linear block code is measured in terms of the state and branch complexities [36], [37], [38]. These two measures are obtained from the TOGM for that code.

In a minimal trellis, the ith section corresponds to the ith column of the TOGM, and the ith time instant corresponds to the "space between" columns i and $i+1$ of the TOGM. The smallest interval of the columns containing all the nonzero bits of a row $\mathbf{g_i}$ of the TOGM, denoted as span($\mathbf{g_i}$), represents the interval of the sections during which the information bit associated with the row $\mathbf{g_i}$ can affect the encoder output. The spans of the rows in the TOGM for RM (8,4) code, of (2.1), are: span($\mathbf{g_1}$)=[1,4], span($\mathbf{g_2}$)=[2,7], span($\mathbf{g_3}$)=[3,6], and span($\mathbf{g_4}$)=[5,8]. The set of time instants at which the information bit associated with the row $\mathbf{g_i}$ can affect the encoder state are contained in this interval.

State Complexity

The state space of the encoder at each time instant i is determined by the set of information bits, denoted as A_i^s, corresponding to the rows of the TOGM, denoted as $\mathbf{G}_i^s$, that affect that time

instant. The total number of states at each time instant i, denoted as $|S_i|$, is: $|S_i| = 2^{|\mathbf{G}_i^s|}$, where $|\mathbf{G}_i^s|$ is the dimension of the set $\mathbf{G}_i^s$. For example, for the RM (8,4) code, the total number of states at each time instant i is $\{1,2,4,8,4,8,4,2,1\}$.

The number of rows in the TOGM that affect a time instant i is always equal to the number of rows that affect a time instant $N - i$, for all RM codes [7]. This implies that the number of states at these time instants is equal, and consequently the bit-level trellis has a mirror symmetry with respect to the $N/2$ time instant.

Branch Complexity

The total number of branches in each section i, denoted as $|B_i|$, is determined by the number of rows that affect that section, in other words, by the rows whose spans contain i. The information bits that correspond to these rows affect the encoder output during the interval associated with that section. For the RM (8,4) code, the total number of branches in each section i is $\{2,4,8,8,8,8,4,2\}$.

2.1.3 State Labelling and Transition

The label of the current state of the encoder at time instant i is defined by the set A_i^s. Each state is labelled by a $|A_i^s|_{\max}$-tuple, where $|A_i^s|_{\max}$ is the maximum dimension of the A_i^s sets. This is sufficient, since the number of states at any level of the trellis is at most equal to $2^{|A_i^s|_{\max}}$. The first $|A_i^s|$ components of the state label correspond to the specific combinations of the information bits in A_i^s, and the remaining $|A_i^s|_{\max} - |A_i^s|$ components are set to 0.

The encoder output, c_{i+1}, generated during the $(i+1)$th bit interval, connects the current state of the encoder, $\sigma \in S_i$, to a state $\sigma' \in S_{i+1}$, whose label contains the same combination of bits that also define σ. The encoder output is determined by $\sigma \in S_i$ and by the information bit, denoted as a^*, that only starts to affect the encoder during that interval, for which the first nonzero component of its corresponding row is in the $(i+1)$th column. It is defined as:

$$c_{i+1} = a^* + \sum_{l=1}^{|\mathbf{G}_i^s|} a_l g_{l,i+1} \tag{2.2}$$

where a^* is the current information bit, and the second term of (2.2) is the contribution of the current state of the encoder defined by $A_i^s = \{a_1, a_2, \ldots, a_{|A_i^s|}\}$ corresponding to $\mathbf{G}_i^s = \{\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g}_{|\mathbf{G}_i^s|}\}$, and $g_{l,i+1}$ is the $(i+1)$th component of $\mathbf{g_l}$. The current information bit a^*,

if it exists, starts to affect the encoder while c_{i+1} is being formed, and as such, its value is not known to the encoder until after the code bit c_{i+1} is transmitted. For this reason, the code bit c_{i+1} can have two possible values, depending on the value of a^*, represented by two branches diverging from each state $\sigma \in S_i$ to two states in S_{i+1}. If there is no current input information bit, the code bit c_{i+1} has only one possible value, determined by the current state of the encoder, for each $\sigma \in S_i$. Hence, there is one branch diverging from each state $\sigma \in S_i$ to one state in S_{i+1}.

Figure 2.1 shows the bit-level trellis diagram for a RM (8,4) code.

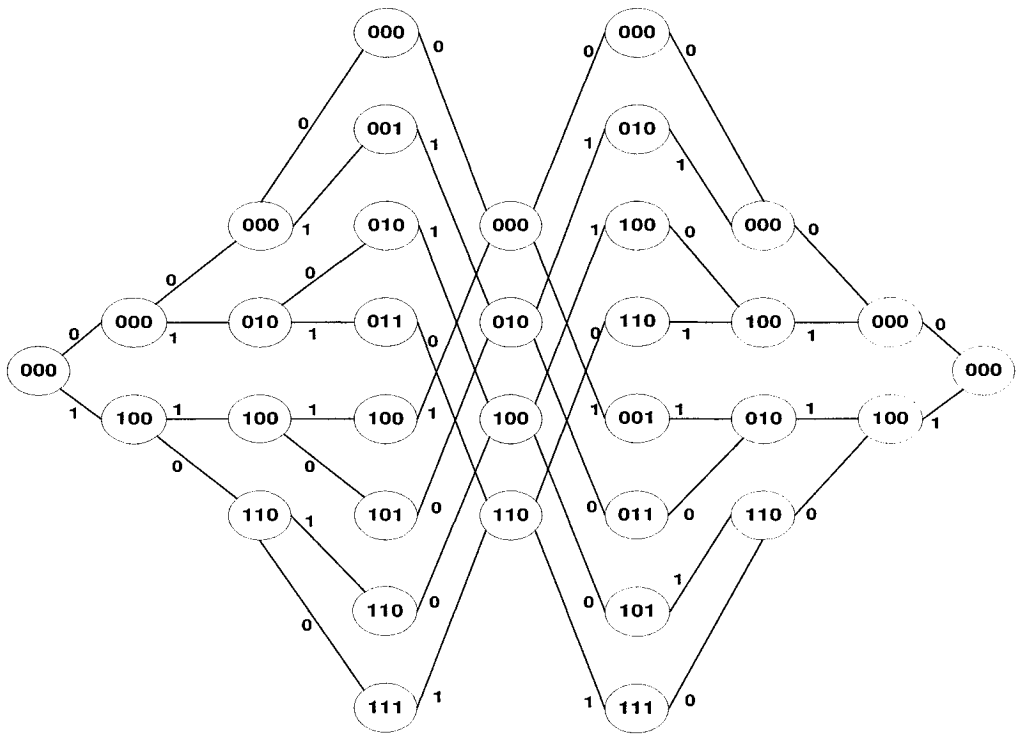


Fig. 2.1 Bit-level trellis for RM (8,4) code.

2.2 Computational Complexity

The computational complexity of an algorithm considered in this thesis, as mentioned in the Introduction, measures only the number of real operations required to decode a bit-level trellis.

The storage and memory requirements, decoding delay and speed, and the addressing issues are not taken into account.

In the next section, the assumptions made for computing the decoding complexity are stated. The computational complexity of the decoding steps for all algorithms considered is presented in the next section, using methods for the most computationally efficient implementation for the algorithms on a bit-level trellis.

2.2.1 Assumptions

The following assumptions were applied in this thesis to ease the calculation of the computational complexities of the decoding algorithms of interest:

- No computations are necessary for exp() and log() operations, as they are accomplished by a table lookup.
- No computations are necessary for a negation operation.
- Comparison, subtraction, and addition operations are equivalent in terms of complexity.
- In a digital signal processing chip, a multiplication is often a one-cycle operation, while a division requires many cycles. However, in this thesis, division and multiplication operations are assumed to be equivalent in terms of complexity.

2.2.2 Computational Complexity of the Decoding Steps

The computational complexity of a trellis-based algorithm is essentially a function of the number of states, $|S_i|$, at each time instant, i, for $0 \leq i \leq N$, and the number of branches, $|B_i|$, in each section, T_i, for $1 \leq i \leq N$, of a trellis. The number of computations required by Viterbi, MAP, and Max-Log-MAP for each decoding step have been derived in [3] and [4]. The results are presented next along with the required complexity for each decoding step of SOVA in section T_i of a bit-level trellis. The complexity of decoding an entire trellis for each algorithm is the sum of the required computations for each T_i, for $1 \leq i \leq N$.

For all the decoding algorithms examined in this thesis, it is necessary to compute all the branch probabilities, all the state probabilities in the forward recursion, and for the SISO algorithms, also all the state probabilities in the backward recursion to be used in the calculation of the soft-output for the estimated code bits.

The processing of Viterbi, Max-Log-MAP, and SOVA algorithms that consider ML paths is in the log domain. The notation of $\bar{P}$, $\bar{\alpha}$, and $\bar{\beta}$ is used to represent $\log P$, $\log \alpha$, $\log \beta$, respectively, the probabilities that are computed by these algorithms in the log domain.

Branch Probability

The branch probability, $P(b_i(\sigma, \sigma'))$, representing the probability of the transition of the encoder from state $\sigma \in S_{i-1}$ to state $\sigma' \in S_i$ through the branch $b_i(\sigma, \sigma')$ associated with r_i, in T_i, is defined as:

$$\begin{aligned} P(b_i(\sigma, \sigma')) &= P(\sigma', b_i(\sigma, \sigma'), r_i|\sigma) \\ &= P(\sigma', b_i(\sigma, \sigma')|\sigma) P(r_i|(\sigma, \sigma'), b_i(\sigma, \sigma')) \end{aligned} \tag{2.3}$$

For equiprobable signalling, $P(\sigma', b_i(\sigma, \sigma')|\sigma)$ is constant in T_i. Therefore, for AWGN channel with zero mean and variance $N_0/2$,

$$\begin{aligned} P(b_i(\sigma, \sigma')) &\propto P(r_i|(\sigma, \sigma'), b_i(\sigma, \sigma')) \\ &\propto \frac{1}{\sqrt{2\pi N_0/2}} \exp\{-\frac{(r_i - u_i)^2}{2N_0/2}\} \end{aligned} \tag{2.4}$$

where $u_i = 2c_i - 1$. The number of calculations required to compute $P(b_i(\sigma, \sigma'))$ is reduced, without changing the code bit estimate, by factoring out the first term and expanding the exponent:

$$P(b_i(\sigma, \sigma')) \propto \exp\{-\frac{(r_i^2 - 2r_i u_i + u_i^2)}{2N_0/2}\} \tag{2.5}$$

The first term in the numerator of the exponent, r_i^2, is common for all branches in T_i, and the last term, u_i^2, always equals 1. Therefore, $P(b_i(\sigma, \sigma'))$ is simplified to:

$$P(b_i(\sigma, \sigma')) \propto \exp\{\frac{2}{N_0} r_i u_i\} \tag{2.6}$$

For Viterbi, SOVA, and Max-Log-MAP, the branch probability is expressed as:

$$\bar{P}(b_i(\sigma, \sigma')) \propto \frac{2}{N_0} r_i u_i \tag{2.7}$$

As these algorithms consider only branches with maximum $\bar{P}(b_i(\sigma, \sigma'))$ for each state $\sigma' \in$

S_i, and as the above expression is an increasing function of $r_i u_i$, it suffices to compute only that for these algorithms. This requires at most a negation operation, since $u_i \in \{-1, +1\}$.

For the MAP algorithm that considers all branches, and taking into account that exponential operations are costless, computing (2.6) requires one multiplication in each T_i.

Forward Recursion

The probability of the encoder reaching a state $\sigma' \in S_i$ from a state $\sigma \in S_{i-1}$, through the branch $b_i(\sigma, \sigma')$ in T_i, in the forward recursion for time instants $i = 0$ to $i = N$, is defined for Viterbi, SOVA, and Max-Log-MAP, as:

$$\bar{\alpha}_i(\sigma') = \max_{\sigma \in \Omega_{i-1}(\sigma')} \{\bar{P}(b_i(\sigma, \sigma')) + \bar{\alpha}_{i-1}(\sigma)\} \tag{2.8}$$

where $\Omega_{i-1}(\sigma')$ denotes the set of states in S_{i-1} that are adjacent to a state $\sigma' \in S_i$. The initial state probability in the forward recursion is $\bar{\alpha}_0(\sigma_0) = 0$.

For each state $\sigma' \in S_i$, an addition operation is required for each branch converging into that state, and the results are then compared to find the maximum value. For the first section, T_i, considering that $\bar{\alpha}_0(\sigma_0) = 0$, no additions are required. In summary, the following number of operations is required in the forward recursion for each T_i:

$$\begin{aligned} &\textbf{comparisons}: \quad |B_i| - |S_i|, \quad \text{for} \quad 1 \leq i \leq N \\ &\textbf{additions}: \quad \begin{cases} 0, & \text{for} \quad T_1, \\ |B_i|, & \text{for} \quad 1 < i \leq N. \end{cases} \end{aligned}$$

For MAP decoding, the state probabilities in the forward recursion are defined as:

$$\alpha_i(\sigma') = \sum_{\sigma \in \Omega_{i-1}(\sigma')} P(b_i(\sigma, \sigma'))\alpha_{i-1}(\sigma) \tag{2.9}$$

with the initial state probability in the forward recursion $\alpha_0(\sigma_0) = 1$.

Applying the analysis and consideration of (2.8), MAP algorithm requires the same number of operations, except that comparisons are replaced with additions, and additions are replaced with multiplications.

Backward Recursion

For all SISO algorithms, it is also necessary to calculate the state probabilities in the backward recursion for time instants $i = N$ to $i = 0$. For SOVA and Max-Log-MAP, the probability of the encoder, reaching a state $\sigma \in S_{i-1}$ from state $\sigma' \in S_i$, through the branch $b_i(\sigma, \sigma')$ in T_i, is defined as:

$$\bar{\beta}_{i-1}(\sigma) = \max_{\sigma' \in \Psi_i(\sigma)} \{\bar{P}(b_i(\sigma, \sigma')) + \bar{\beta}_i(\sigma')\} \tag{2.10}$$

where $\Psi_i(\sigma)$ denotes the set of states in S_i that are adjacent to a state $\sigma \in S_{i-1}$. The final state probability in the backward recursion is $\bar{\beta}_N(\sigma_f) = 0$.

Similarly to calculating the forward state probabilities, an addition operation is required for each branch diverging from the state $\sigma \in S_{i-1}$, and the results are then compared to find the maximum value. For the last section, considering that $\bar{\beta}_N(\sigma_f) = 0$, no additions are required. In summary, the following number of operations is required in the backward recursion for each T_i:

$$\begin{aligned} &\textbf{comparisons}: \quad |B_i| - |S_{i-1}|, \quad \text{for} \quad 1 \leq i \leq N \\ &\textbf{additions}: \quad \begin{cases} 0, & \text{for} \quad T_N, \\ |B_i|, & \text{for} \quad 1 \leq i < N. \end{cases} \end{aligned}$$

For MAP decoding, the state probabilities in the backward recursion are defined as:

$$\beta_{i-1}(\sigma) = \sum_{\sigma' \in \Psi_i(\sigma)} P(b_i(\sigma, \sigma'))\beta_i(\sigma') \tag{2.11}$$

with the final state probability in the backward recursion $\beta_N(\sigma_f) = 1$.

MAP algorithm requires the same number of operations as SOVA and Max-Log-MAP, except that comparisons and additions are replaced with additions and multiplications, respectively.

Soft Output

The soft output for each estimated code bit is obtained for each SISO algorithm, using all the probabilities specified above, in the form of a LLR of the APP of the transmitted bits. The sign of this value determines the estimated code bits.

For SOVA, the soft output for each decoded bit, $L(\hat{c}_i)$, is approximated as the difference

between the probability of the ML path and the probability of the most probable path with the complementary label to the ML estimate in T_i:

$$L(\hat{c}_i) \approx \bar{P}(c_i = 0|\mathbf{r}) - \bar{P}(c_i = 1|\mathbf{r}) \tag{2.12}$$

Depending on the ML estimate in T_i, the probability of the ML path, $\bar{\alpha}_N(\sigma_f)$, is assigned to the appropriate path probability. The computations become necessary in finding the probability of the most probable path whose label is complementary, x, to the ML estimate $\hat{c}_i$:

$$\bar{P}(c_i = x|\mathbf{r}) = \max_{\substack{(\sigma,\sigma') \\ c_i = x}} \{\bar{\alpha}_{i-1}(\sigma) + \bar{P}(b_i(\sigma,\sigma')) + \bar{\beta}_i(\sigma')\} \tag{2.13}$$

Computing the above requires 2 additions for each branch with label x in each section of a trellis, other than in the first, T_1, and last, T_N. For these sections, considering that $\bar{\alpha}_0(\sigma_0) = 0$ and $\bar{\beta}_N(\sigma_f) = 0$, only 1 addition, for each branch with appropriate label, is required. These results are compared to find the maximum. The LLR of the estimated code is evaluated requiring only one subtraction. In summary, the following number of operations is required in computing the soft output for SOVA in each T_i:

$$\begin{aligned} &\textbf{comparisons}: && |B_i|/2 - 1, && \text{for} \quad 1 \le i \le N \\ &\textbf{additions}: && \begin{cases} 1 \cdot |B_i|/2 + 1, & \text{for} \quad i = 1, N, \\ 2 \cdot |B_i|/2 + 1, & \text{for} \quad 1 < i < N. \end{cases} \end{aligned}$$

For Max-Log-MAP, the soft output, $L(\hat{c}_i)$, for each decoded bit, $\hat{c}_i$, is approximated by the difference between the probabilities of the most probable path associated with bit one and the most probable path associated with bit zero:

$$\begin{aligned} L(\hat{c}_i) \approx & \max_{\substack{(\sigma,\sigma') \\ c_i = 1}} \{\bar{\alpha}_{i-1}(\sigma) + \bar{P}(b_i(\sigma,\sigma')) + \bar{\beta}_i(\sigma')\} \\ & - \max_{\substack{(\sigma,\sigma') \\ c_i = 0}} \{\bar{\alpha}_{i-1}(\sigma) + \bar{P}(b_i(\sigma,\sigma')) + \bar{\beta}_i(\sigma')\} \end{aligned} \tag{2.14}$$

Computing each part of (2.14) requires 2 additions for each branch labelled appropriately in each section of a trellis, other than in the first, T_1, and last, T_N. For these sections, considering that $\bar{\alpha}_0(\sigma_0) = 0$ and $\bar{\beta}_N(\sigma_f) = 0$, only 1 addition, for each branch with appropriate label, is required.

These results are then compared to find the maximum. Finally, the soft output of the estimated code bit is evaluated, requiring one subtraction. In summary, the following number of operations is needed for computing the soft output for Max-Log-MAP in each T_i:

$$\begin{aligned} &\textbf{comparisons}: && 2\cdot(|B_i|/2-1), && \text{for} \quad 1\le i\le N \\ &\textbf{additions}: && \begin{cases} 2\cdot(|B_i|/2)+1, & \text{for} \quad i=1,N, \\ 2\cdot(2\cdot|B_i|/2)+1, & \text{for} \quad 1<i<N. \end{cases} \end{aligned}$$

MAP algorithm provides the soft output, $L(\hat{c}_i)$, for each estimated code bit, $\hat{c}_i$, by the ratio, in the logarithmic domain, of the probabilities of all paths in T_i associated with label 1, to the probabilities of all paths in T_i associated with label 0:

$$L(\hat{c}_i) = \log \frac{\sum\limits_{\substack{(\sigma,\sigma') \\ c_i=1}} \alpha_{i-1}(\sigma)P(b_i(\sigma,\sigma'))\beta_i(\sigma')}{\sum\limits_{\substack{(\sigma,\sigma') \\ c_i=0}} \alpha_{i-1}(\sigma)P(b_i(\sigma,\sigma'))\beta_i(\sigma')} \tag{2.15}$$

Computing the numerator, requires 2 multiplications for all branches labelled with a positive bit in each section of a trellis, other than in the first, T_1, and last, T_N. For these sections, considering that $\alpha_0(\sigma_0)=1$ and $\beta_N(\sigma_f)=1$, only 1 multiplication, for each branch with appropriate label, is required. These results are then added. The same number of operations is needed for the denominator. Finally, the soft output of the estimated code bit is evaluated, requiring one division since it is assumed that no computations are necessary for log() operation. In summary, the following number of operations is required for computing the soft output for MAP in each T_i:

$$\begin{aligned} &\textbf{additions}: && 2\cdot(|B_i|/2-1), && \text{for} \quad 1\le i\le N \\ &\textbf{multiplications}: && \begin{cases} 2\cdot(|B_i|/2)+1, & \text{for} \quad i=1,N, \\ 2\cdot(2\cdot|B_i|/2)+1, & \text{for} \quad 1<i<N. \end{cases} \end{aligned}$$

2.3 Summary

A minimal trellis was constructed from the trellis oriented generator matrix. The computationally efficient methods were employed for the analysis of the computational complexities of Viterbi, SOVA, Max-Log-MAP, and MAP algorithms for decoding a bit-level trellis. The total number of computations required by each algorithm for decoding a unit section of a bit-level trellis is presented in Table 2.1. In Chapter 4, these computational complexities are compared for different RM (N, K) codes. In the next chapter, the decoding complexities of these algorithms are reduced by means of sectionalizing a trellis.

Table 2.1 Computational complexities of Viterbi, SOVA, Max-Log-MAP, and MAP algorithms for decoding a unit section of a trellis.

Decoding Steps	Viterbi	SOVA	Max-Log-MAP	MAP
Branch Probabilities				
Comparisons	0	0	0	0
Additions	0	0	0	0
Multiplications	0	0	0	1
Forward Recursion				
Comparisons	$\lvert B_i\rvert - \lvert S_i\rvert$	$\lvert B_i\rvert - \lvert S_i\rvert$	$\lvert B_i\rvert - \lvert S_i\rvert$	0
Additions	0, for $i = 1$ $\lvert B_i\rvert$, for $1 < i \leq N$	0, for $i = 1$ $\lvert B_i\rvert$, for $1 < i \leq N$	0, for $i = 1$ $\lvert B_i\rvert$, for $1 < i \leq N$	$\lvert B_i\rvert - \lvert S_i\rvert$
Multiplications	0	0	0	0, for $i = 1$ $\lvert B_i\rvert$, for $1 < i \leq N$
Backward Recursion	N/A			
Comparisons		$\lvert B_i\rvert - \lvert S_{i-1}\rvert$	$\lvert B_i\rvert - \lvert S_{i-1}\rvert$	0
Additions		0, for $i = N$ $\lvert B_i\rvert$, for $1 \leq i < N$	0, for $i = N$ $\lvert B_i\rvert$, for $1 \leq i < N$	$\lvert B_i\rvert - \lvert S_{i-1}\rvert$
Multiplications		0	0	0, for $i = N$ $\lvert B_i\rvert$, for $1 \leq i < N$
Soft Output	N/A			
Comparisons		$\lvert B_i\rvert/2 - 1$	$\lvert B_i\rvert - 2$	0
Additions		$\lvert B_i\rvert/2 + 1$, for $i = 1, N$ $\lvert B_i\rvert + 1$, for $1 < i < N$	$\lvert B_i\rvert + 1$, for $i = 1, N$ $2 \cdot \lvert B_i\rvert + 1$, for $1 < i < N$	$\lvert B_i\rvert - 2$
Multiplications		0	0	$\lvert B_i\rvert + 1$, for $i = 1, N$ $2 \cdot \lvert B_i\rvert + 1$, for $1 < i < N$

Chapter 3

Sectionalized Trellis

This chapter investigates how sectionalization of a trellis diagram can reduce the computational complexity of the decoding algorithms. The first part of this chapter, Section 3.1, defines the sectionalized trellis complexities and explains the construction of the trellis for any set of section boundaries. The following section presents the computation-wise optimum sectionalization algorithm, that was applied in this thesis to obtain the section boundaries that are optimal for the computational complexity of the algorithms considered. The second part of this chapter, Section 3.2, analyzes the computational complexity of the decoding algorithms of interest applied to a sectionalized trellis. The methods used to analyze computational complexity of the Viterbi algorithm [3], MAP and Max-Log-MAP algorithms [4] based on sectionalized trellis diagrams are considered and applied to obtain the computational complexity for SOVA based on a sectionalized trellis. The chapter concludes with the summary of the results.

3.1 Sectionalized Trellis Construction

In this thesis, a trellis for a linear block code was sectionalized. The following sections define the complexity of the sectionalized trellis, in terms of the encoder states at the new section boundaries, and the number of composite and parallel branches connecting these states. The expression for the possible state transitions of the encoder is also presented.

3.1.1 Sectionalized Trellis Complexity

The complexity of a sectionalized trellis for a linear block code is measured in terms of the state and branch complexities, both of which are obtained from the TOGM of that code.

State Complexity

The set of information bits, denoted as $A^s_{h_j}$, corresponding to the rows of TOGM, denoted as $\mathbf{G}^s_{h_j}$, that affect the time instant h_j determine the state space of the encoder at that instant. The total number of states at each time instant h_j, denoted as $|S_{h_j}|$, is $|S_{h_j}| = 2^{|\mathbf{G}^s_{h_j}|}$, where $|\mathbf{G}^s_{h_j}|$ is the dimension of $\mathbf{G}^s_{h_j}$ set. For example, for the RM (8,4) code, with section boundaries $\{0,4,8\}$ the total number of states at the three time instants is $\{1,4,1\}$.

Branch Complexities

The branch complexity of a sectionalized trellis is measured in terms of the number of composite branches, distinct composite branches, and the number of branches contained within each composite branch, known as parallel branches.

The set of composite branches, B^c_{j+1}, in the interval $[h_j, h_{j+1}]$, belong to the subcode generated by the rows of the TOGM, denoted as $\mathbf{G}^c_{h_{j+1}}$, that affect that interval as well as time instant h_{j+1}. In other words, by the rows whose spans are [y, w], where $h_j \leq y \leq h_{j+1}$ and $w > h_{j+1}$, and the subcode is denoted as C'. The number of composite branches diverging from each state $\sigma \in S_{h_j}$, in a section T_{j+1} is $|B^{c,div}_{j+1}| = 2^{k(C')}$, where $k(C')$ is the dimension of the code C'. The total number of composite branches in a section is then $|B^c_{j+1}| = |S_{h_j}| \cdot |B^{c,div}_{j+1}|$. For example, for RM (8,4) code, with $\{0,4,8\}$ section boundaries, there are 4 composite branches diverging from the state $\sigma_0 \in S_0$ since $\mathbf{g_2}, \mathbf{g_3} \in C'$ for the interval [0,4]. In the interval [4,8], $C' = \{\}$, and therefore, there is one composite branch diverging from each of the four states, and the total number of composite branches in that section is $|B^c_2| = 4 \cdot 1 = 4$.

The set of distinct composite branches, B^d_{j+1}, is a subset of the set of composite branches, consisting of only those whose parallel branches are different. The set B^d_{j+1} is determined by the linearly independent combinations of the punctured rows, obtained by removing the first h_j and the last $N - h_{j+1}$ components, of the subcode C'. The number of distinct composite branches in a section T_{j+1} is denoted as $|B^d_{j+1}|$. For the RM (8,4) code, the number of distinct branches in both intervals, [0,4] and [4,8], is equal to the number of composite branches, since the

dimension of C' for the two sections is the same as the dimension of the punctured C'. Therefore, $|B_1^d| = |B_2^d| = 4$.

The set of parallel branches, B_{j+1}^p, in the interval $[h_j, h_{j+1}]$, are generated by the subcode, $C_{h_j,h_{j+1}}$, formed by the rows of TOGM, denoted as $\mathbf{G}_{h_{j+1}}^p$, that only affect that interval. In other words, by the rows whose spans are contained within that interval. The number of parallel branches within each composite branch in the T_{j+1} section is then $|B_{j+1}^p| = 2^{k(C_{h_j,h_{j+1}})}$, where $k(C_{h_j,h_{j+1}})$ is the dimension of the code $C_{h_j,h_{j+1}}$. For example, for RM (8,4) code, in the interval [0,4], $\mathbf{g_1} \in C_{0,4}$. Therefore, in that section, there are 2 parallel branches in each composite one. In the interval [4,8], $\mathbf{g_4} \in C_{4,8}$, and hence, $|B_2^p| = 2$.

3.1.2 State Labelling and Transition

The encoder output, generated during the interval $[h_j, h_{j+1}]$, that connects the current state of the encoder at time instant h_j, $\sigma \in S_{h_j}$, to a state $\sigma' \in S_{h_{j+1}}$ at time instant h_{j+1}, is determined by the current encoder state and the current information bits.

The current state of the encoder is defined by the information bits corresponding to the rows in the set $\mathbf{G}_{h_j}^s$. The state labelling of a sectionalized trellis corresponds to that of a bit-level trellis. Each state is labelled by a $|\mathbf{G}_{h_j}^s|_{\max}$-tuple, with the first $|\mathbf{G}_{h_j}^s|$ components corresponding to the combinations of the information bits A_j^s and the remaining components are set to zero.

Since the information bits that define parallel branches B_{j+1}^p and the ones that define composite branches B_{j+1}^c start to affect the encoder during $[h_j, h_{j+1}]$ interval, they are considered current information bits. The encoder output in that interval, $\mathbf{c}_{h_{j+1}}$, is defined as:

$$\mathbf{c}_{h_{j+1}} = \sum_{l=1}^{|\mathbf{G}_{h_j}^s|} a_l p_{h_j,h_j+1}(g_l^s) + \sum_{t=1}^{|\mathbf{G}_{h_j+1}^c|} a_t p_{h_j,h_j+1}(g_t^c) + \sum_{f=1}^{|\mathbf{G}_{h_j+1}^p|} a_f p_{h_j,h_j+1}(g_f^p) \tag{3.1}$$

The first term is the contribution of the current state of the encoder, defined by $A_{h_j}^s = \{a_1, a_2, \ldots, a_{|A_{h_j}^s|}\}$ corresponding to $\mathbf{G}_{h_j}^s = \{\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g}_{|\mathbf{G}_{h_j}^s|}\}$, and $p_{h_j,h_{j+1}}(g_l^s)$ is the punctured $\mathbf{g_l}$ row of $\mathbf{G}_{h_j}^s$, obtained by removing the first h_j and the last $N - h_{j+1}$ components of that row. The second term is the contribution of the composite branches, defined by $A_{h_j+1}^c = \{a_1, a_2, \ldots, a_{|A_{h_j+1}^c|}\}$ corresponding to $\mathbf{G}_{h_j+1}^c = \{\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g}_{|\mathbf{G}_{h_j+1}^c|}\}$, and $p_{h_j,h_{j+1}}(g_t^c)$ is the punctured row of $\mathbf{G}_{h_{j+1}}^c$. The third term is the contribution of the parallel branches, defined by $A_{h_j+1}^p = \{a_1, a_2, \ldots, a_{|A_{h_j+1}^p|}\}$ corresponding to $\mathbf{G}_{h_j+1}^p = \{\mathbf{g_1}, \mathbf{g_2}, \ldots, \mathbf{g}_{|\mathbf{G}_{h_j+1}^p|}\}$, and

$p_{h_j,h_{j+1}}(g_f^p)$ is the punctured row of $\mathbf{G}_{h_{j+1}}^p$.

Figure 3.1 shows the sectionalized trellis for RM (8,4) code with $\{0,4,8\}$ section boundaries.

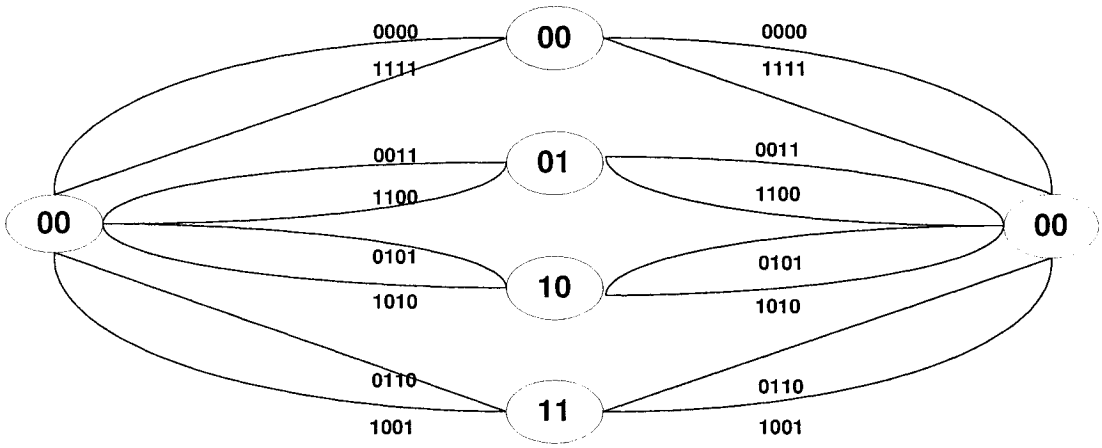


Fig. 3.1 Sectionalized trellis for RM (8,4) code with $\{0,4,8\}$ section boundaries.

3.1.3 Optimum Sectionalization

There are 2^{N-1} possible ways to select the section boundaries and each may substantially affect the decoding complexity. In this thesis, the algorithm that obtains the optimal section boundaries, based on the optimality criterion of minimizing the total number of required computations for decoding a trellis of a linear block code is applied to the decoders of interest. The algorithm was devised for Viterbi decoding of block codes by Lafourcade and Vardy in [3], and was applied to MAP and Max-Log-MAP decoding by Liu, Lin, and Fossorier in [4].

In the following algorithm, the expression $\{T_i \star T_{i+1} \star \cdots\}$ represents the formation of one section from the adjacent unit sections, $\{T_i, T_{i+1}, \ldots\}$, by joining these sections together. Their shared boundaries are omitted from the sectionalized boundaries. The expression $\{T_i \circ T_{i+1}^{\min}\}$

represents the addition of section T_i to the sectionalized trellis up to that point, $T_{i+1}^{\min}$. Their shared boundary, hence, is one of the chosen section boundaries. The expression $F(T_i \star T_{i+1} \star \cdots)$ represents the total number of computations required by the decoder for the section formed from the unit sections within the parentheses. In Section 3.2, the number of computations required by Viterbi, SOVA, Max-Log-MAP and MAP algorithms for decoding a section in a trellis, are analyzed and are summarized in Table 3.1.

The optimum sectionalization algorithm consists of the following three steps:

1. Initialization: $T_N^{\min} = T_N$ and $i = N - 1$

2. The minimum value among the following is selected:

   a) $F(T_i) + F(T_{i+1}^{\min})$

   b) $F(T_i \star T_{i+1} \star \cdots \star T_j) + F(T_{j+1}^{\min}), \qquad \text{for } j = i+1, i+2, \cdots, N-1$

   c) $F(T_i \star T_{i+1} \star \cdots \star T_N)$

3. Corresponding to the expression selected above, $T_i^{\min}$ is set to one of the following:

   a) $T_i^{\min} = T_i \circ T_{i+1}^{\min}$

   b) $T_i^{\min} = (T_i \star T_{i+1} \star \cdots \star T_{j_{\min}}) \circ T_{j_{\min}+1}^{\min}, \qquad$ where $j_{\min}$ is the j value in (b)

   c) $T_i^{\min} = T_i \star T_{i+1} \star \cdots \star T_N$

The value of i is decremented at the end of step 3, and steps 2 and 3 are repeated until $T_1^{\min}$ is obtained, representing the optimal sequence:

$$T_1^{\min} = (T_1 \star \cdots \star T_{h_1}) \circ (T_{h_1+1} \star \cdots \star T_{h_2}) \circ \cdots \circ (T_{h_{v-1}+1} \star \cdots \star T_{h_v})$$

corresponding to the optimal sectionalization of the trellis with section boundaries:

$$0 = h_0, h_1, h_2, \ldots, h_v = N$$

that requires the minimal computational complexity of the decoder to which the above algorithm is applied to. In this thesis, this algorithm was applied to SOVA, Viterbi, Max-Log-MAP, and MAP decoders. The obtained optimal sectionalizations and the corresponding computational complexities of these decoders are presented in Section 4.1.

3.2 Computational Complexity

This section presents the computationally efficient methods, applied in this thesis, to obtain the computational complexity of decoding steps of Viterbi, Max-Log-MAP, and MAP algorithms based on a sectionalized trellis. In the following section, these methods are applied to determine the computational complexity of decoding a sectionalized trellis by SOVA.

3.2.1 Computational Complexities of the Decoding Steps of Viterbi, Max-Log-MAP and MAP Algorithms

This thesis applied the approaches suggested by Lafourcade and Vardy in [3] for efficient computation of the branch and state probabilities in the forward recursion for a sectionalized trellis. These were employed to derive efficient computation of state probabilities in the backward recursion and of soft output for SISO algorithms.

This section presents the computational complexity of Viterbi, Max-Log-MAP and MAP decoding of one section T_j, from time h_{j-1} to time h_j, of length $\ell_j = h_j - h_{j-1}$, as a function of the number of states $|S_{h_j}|$, the number of composite branches $|B_j^c|$, the number of parallel branches $|B_j^p|$, and the number of distinct composite branches $|B_j^d|$, for a trellis sectionalized into v sections, with section boundaries $\{h_0, h_1, \ldots, h_v\}$. The total complexity of the sectionalized trellis is the sum of the complexities for each T_j, for $1 \leq j \leq v$. Using similar methods, the application of SOVA on a sectionalized trellis and its computational complexity is analyzed in the following section.

The same assumptions are applied as in Section 2.2.1 where the decoding complexity of a bit-level trellis was analyzed. The notation of $\bar{P}$, $\bar{\alpha}$, and $\bar{\beta}$ is used to represent the probabilities computed by Viterbi, Max-Log-MAP, and SOVA algorithms in the log domain.

Branch Probability

The branch probability, $P(b_{h_j}(\sigma, \sigma'))$, representing the probability of the transition of the encoder from state $\sigma \in S_{h_{j-1}}$ to state $\sigma' \in S_{h_j}$ through the branch $b_{h_j}(\sigma, \sigma')$, associated with the jth section of the received sequence, $\mathbf{r_j} = \{r_{h_{j-1}+1}, r_{h_{j-1}+2}, \ldots, r_{h_j}\}$, in T_j, is defined as:

$$\begin{aligned} P(b_{h_j}(\sigma, \sigma')) &= P(\sigma', b_{h_j}(\sigma, \sigma'), \mathbf{r_j}|\sigma) \\ &= P(\sigma', b_{h_j}(\sigma, \sigma')|\sigma) P(\mathbf{r_j}|(\sigma, \sigma'), b_{h_j}(\sigma, \sigma')) \end{aligned} \tag{3.2}$$

As for computing the branch probabilities of the bit-level trellis, in Section 2.2.2, simplifying $P(b_{h_j}(\sigma, \sigma'))$, and factoring out common terms in its exponent, for AWGN channel with zero mean and variance $N_0/2$,

$$P(b_{h_j}(\sigma, \sigma')) \propto \exp\left\{\frac{2}{N_0} \sum_{m=h_{j-1}+1}^{h_j} r_m u_m\right\}, \qquad \text{where } u_i = 2c_i - 1. \tag{3.3}$$

For Viterbi and Max-Log-MAP algorithms, the branch probability is expressed as:

$$\bar{P}(b_{h_j}(\sigma, \sigma')) \propto \frac{2}{N_0} \sum_{m=h_{j-1}+1}^{h_j} r_m u_m. \tag{3.4}$$

As these algorithms are only interested in a branch with maximum $\bar{P}(b_{h_j}(\sigma, \sigma'))$ for each composite branch for each state $\sigma' \in S_{h_j}$, it suffices to compute only $\sum_{m=h_{j-1}+1}^{h_j} r_m u_m$. Considering that no operations are required for computing the inside of the summation, obtaining $\bar{P}(b_{h_j}(\sigma, \sigma'))$, requires $\ell_j - 1$ additions for each parallel branch, $|B_j^p|$, within each distinct composite branch $|B_j^d|$. Summarizing, the number of addition operations required for each T_j:

$$\textbf{additions}: \quad |B_j^d| \cdot |B_j^p| \cdot (\ell_j - 1), \quad \text{for} \quad 1 \leq j \leq v$$

In some cases, this complexity can be further reduced by considering the following conditions:

- If the code generated by each section is self-complementary, only $|B_j^d| \cdot |B_j^p|/2$ branch probabilities need to be computed. The remaining ones are obtained by negating the computed ones. This is the case for RM codes. The number of addition operations, in this case, is reduced to: $|B_j^d| \cdot |B_j^p| \cdot (\ell_j - 1)/2$.

- Gray code ordering may be applied to compute $2^{\ell_j - 1}$ dominant branch probabilities of 2^{ℓ_j} possible ones for a section of length ℓ_j. This method requires $\ell_j - 1$ additions for computing the first branch probability, and only one addition for the remaining $2^{\ell_j - 1} - 1$ ones. Depending on the value of ℓ_j, the number of addition operations may be reduced further to $(\ell_j - 1) + (2^{\ell_j - 1} - 1)$.

- If the codewords in T_j are only of even weight and the length of the section is even, the

number of required additions can be reduced further. Gray code ordering may be applied to compute $2^{\ell_j/2-1}$ dominant branch probabilities corresponding to each half of the section. To obtain from these, the probabilities corresponding to the entire section, an addition operation is required for each of the 2^{ℓ_j-2} dominant branches of even weight. Again, depending on the value of ℓ_j, the number of addition operations may be reduced even further to $2(2^{\ell_j/2-1}+\ell_j/2-2)+2^{\ell_j-2}=2^{\ell_j/2}+\ell_j-4+2^{\ell_j-2}$.

For MAP decoding, applying the analysis of the above and also considering that no computations are needed for the exponential operation, computing (3.3) requires also one multiplication for each of the ℓ_j bits in T_j.

Composite Branch Probability

If there is more than one branch connecting a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$, it is necessary to compute the composite branch probability, $P(L_{h_j}(\sigma,\sigma'))$.

For the Viterbi and Max-Log-MAP algorithms, this probability is defined as the maximum $\bar{P}(b_{h_j}(\sigma,\sigma'))$, among all the ones within that composite branch:

$$\bar{P}(L_{h_j}(\sigma,\sigma')) = \max_{b(\sigma,\sigma')\in L(\sigma,\sigma')} \{\bar{P}(b_{h_j}(\sigma,\sigma'))\} \tag{3.5}$$

This requires all the parallel branch probabilities, within each distinct composite branch, to be compared in order to find the maximum one, hence, requiring:

$$\textbf{comparisons}: \quad |B_j^d| \cdot (|B_j^p| - 1), \quad \text{for} \quad 1 \le j \le v$$

Considering that the objective of these algorithms is to maximize the overall path probability, if the parallel branches are self-complementary, branches with negative $\sum_{m=h_{j-1}+1}^{h_j} r_m u_m$ can be discarded, and only the remaining ones need to be compared. In this case, the number of comparisons for obtaining the composite branch probabilities for each T_j is reduced to:

$$\textbf{comparisons}: \quad |B_j^d| \cdot (|B_j^p|/2 - 1), \quad \text{for} \quad 1 \le j \le v$$

For the MAP algorithm, the composite branch probability, $P(L_{h_j}(\sigma,\sigma'))$, is defined as:

$$P(L_{h_j}(\sigma,\sigma')) = P(L_t^+(\sigma,\sigma')) + P(L_t^-(\sigma,\sigma')), \tag{3.6}$$

computed for one value of t in the range of $h_{j-1}+1 \leq t \leq h_j$. The composite bit probability $P(L_t^{\pm}(\sigma,\sigma'))$ represents the probability of each possible encoder output for each bit u_t within a composite branch, $L_{h_j}(\sigma,\sigma')$ and is defined as:

$$P(L_t^{\pm}(\sigma,\sigma')) = \sum_{\substack{b(\sigma,\sigma')\in L(\sigma,\sigma') \\ u_t=\pm 1}} P(b_{h_j}(\sigma,\sigma')), \quad \text{for } h_{j-1}+1 \leq t \leq h_j.$$

The number of operations required to compute $P(L_t^{\pm}(\sigma,\sigma'))$ is given in the Composite Bit Probability section.

Computing the composite branch probability in (3.6), requires 1 addition for each distinct composite branch.

Forward Recursion

The probability of the encoder reaching a state $\sigma' \in S_{h_j}$ from a state $\sigma \in S_{h_{j-1}}$ in T_j in the forward recursion from time instant $h_j = 0$ to $h_j = N$, is defined for Viterbi and Max-Log-MAP as:

$$\bar{\alpha}_{h_j}(\sigma') = \max_{\sigma\in\Omega_{h_{j-1}}(\sigma')}\{\bar{P}(L_{h_j}(\sigma,\sigma')) + \bar{\alpha}_{h_{j-1}}(\sigma)\}, \qquad \text{with } \bar{\alpha}_0(\sigma_0) = 0. \tag{3.7}$$

If there is only one branch connecting a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$, then $\bar{P}(L_{h_j}(\sigma,\sigma'))$ is replaced with $\bar{P}(b_{h_j}(\sigma,\sigma'))$.

An addition operation is required for each composite branch converging into $\sigma' \in S_{h_j}$, and the results are then compared to find the maximum value. For the first section, T_1, considering that $\bar{\alpha}_0(\sigma_0) = 0$, no additions are required. In summary, the following number of operations is required in the forward recursion for each T_j:

$$\textbf{comparisons}: \quad |B_j^c| - |S_{h_j}|, \quad \text{for} \quad 1 \leq j \leq v$$

$$\textbf{additions}: \quad \begin{cases} 0, & \text{for} \quad T_1, \\ |B_j^c|, & \text{for} \quad 1 < j \leq v. \end{cases}$$

For MAP decoding, the state probabilities in the forward recursion are defined as:

$$\alpha_{h_j}(\sigma') = \sum_{\sigma \in \Omega_{h_{j-1}}(\sigma')} P(L_{h_j}(\sigma, \sigma'))\alpha_{h_{j-1}}(\sigma) \quad \text{with } \alpha_0(\sigma_0) = 1. \tag{3.8}$$

MAP algorithm requires the same number of operations, for obtaining the forward state probabilities, as Viterbi and Max-Log-MAP algorithms, except that comparisons and additions are replaced with additions and multiplications, respectively.

Backward Recursion

For all SISO algorithms, it is also necessary to calculate the state probabilities in the backward recursion from time instant $h_j = N$ to $h_j = 0$.

For Max-Log-MAP algorithm, the probability of the encoder, reaching a state $\sigma \in S_{h_{j-1}}$ from state $\sigma' \in S_{h_j}$ is defined as:

$$\bar{\beta}_{h_{j-1}}(\sigma) = \max_{\sigma' \in \Psi_{h_j}(\sigma)} \{\bar{P}(L_{h_j}(\sigma, \sigma')) + \bar{\beta}_{h_j}(\sigma')\}, \quad \text{with } \bar{\beta}_v(\sigma_f) = 0. \tag{3.9}$$

If there is only one branch connecting a state $\sigma' \in S_{h_j}$ to a state $\sigma \in S_{h_{j-1}}$, then $\bar{P}(L_{h_j}(\sigma, \sigma'))$ is replaced with $\bar{P}(b_{h_j}(\sigma, \sigma'))$.

Similarly to the calculations in a bit-level trellis, an addition operation is required for each branch diverging from the state $\sigma \in S_{h_{j-1}}$, and the results are then compared to find the maximum value. For the last section, T_v, considering that $\bar{\beta}_v(\sigma_f) = 0$, no additions are required. In summary, the following number of operations is required in the backward recursion for each T_j:

$$\textbf{comparisons}: \quad |B_j^c| - |S_{h_{j-1}}|, \quad \text{for} \quad 1 \leq j \leq v$$

$$\textbf{additions}: \quad \begin{cases} 0, & \text{for} \quad T_v, \\ |B_j^c|, & \text{for} \quad 1 \leq j < v. \end{cases}$$

For MAP decoding, the state probabilities in the backward recursion are defined as:

$$\beta_{h_{j-1}} = \sum_{\sigma' \in \Psi_{h_j}(\sigma)} P(L_{h_j}(\sigma, \sigma'))\beta_{h_j}(\sigma') \quad \text{with } \beta_v(\sigma_f) = 1. \tag{3.10}$$

MAP algorithm requires the same number of operations as Max-Log-MAP algorithm, except that comparisons and additions are replaced with additions and multiplications, respectively.

Composite Bit Probability

For the composite branches that contain more than one branch, it is necessary to define the probability of each possible encoder output for each bit u_t for $h_{j-1}+1 \leq t \leq h_j$, in T_j, within $L_{h_j}(\sigma,\sigma')$, represented by a composite bit probability, $P(L_t^{\pm}(\sigma,\sigma'))$.

For Max-Log-MAP, $\bar{P}(L_t^{\pm}(\sigma,\sigma'))$ is defined as:

$$\bar{P}(L_t^{\pm}(\sigma,\sigma')) = \max_{\substack{b(\sigma,\sigma')\in L(\sigma,\sigma') \\ u_t=\pm 1}} \{\bar{P}(b_{h_j}(\sigma,\sigma'))\}, \quad \text{for} \quad h_{j-1}+1 \leq t \leq h_j. \tag{3.11}$$

Depending on the value of each bit of the composite branch, $L_{h_j}(\sigma,\sigma')$, its probability, $\bar{P}(L_{h_j}(\sigma,\sigma'))$, defined in (3.5), is assigned to the appropriate $\bar{P}(L_t^{\pm}(\sigma,\sigma'))$. This is a logic operation, requiring no computations. Obtaining $\bar{P}(L_t^{\pm}(\sigma,\sigma'))$ corresponding to the complementary label of $L_{h_j}(\sigma,\sigma')$, requires that for each bit and for each distinct composite branch, the probabilities of only the branches with the appropriate bits be compared to find the maximum value. In summary, the following number of comparisons are required in each T_j:

$$\textbf{comparisons}: \quad (|B_j^p|/2-1)\cdot|B_j^d|\cdot\ell_j, \quad \text{for} \quad 1\leq j\leq v$$

For MAP, $P(L_t^{\pm}(\sigma,\sigma'))$ is defined as:

$$P(L_t^{\pm}(\sigma,\sigma')) = \sum_{\substack{b(\sigma,\sigma')\in L(\sigma,\sigma') \\ u_t=\pm 1}} P(b_{h_j}(\sigma,\sigma')), \quad \text{for} \quad h_{j-1}+1 \leq t \leq h_j. \tag{3.12}$$

For the evaluation of each possible encoder output, in the above expression, it is necessary to add the probabilities of all the branches corresponding to that encoder output for all ℓ_j bits in each section, within each distinct composite branch. In summary, the following number of additions is required to obtain $P(L_t^{\pm}(\sigma,\sigma'))$ for each T_j:

$$\textbf{additions}: \quad 2\cdot|B_j^d|\cdot(|B_j^p|/2-1)\cdot\ell_j, \quad \text{for} \quad 1\leq j\leq v$$

Soft Output

The soft output for each estimated code bit is obtained for each SISO algorithm, using all the probabilities specified above, in the form of a LLR of the APP of the transmitted bits. The sign of this value determines the estimated code bits.

For the Max-Log-MAP algorithm, the soft output, $L(\hat{c}_t)$, for each decoded bit, $\hat{c}_t$, for $h_{j-1} + 1 \leq t \leq h_j$, in T_j is approximated by:

$$\begin{aligned} L(\hat{c}_t) \approx & \max_{\substack{(\sigma,\sigma') \\ u_t=1}} \{\bar{\alpha}_{h_{j-1}}(\sigma) + \bar{P}(L_t^+(\sigma,\sigma')) + \bar{\beta}_{h_j}(\sigma')\} \\ & - \max_{\substack{(\sigma,\sigma') \\ u_t=-1}} \{\bar{\alpha}_{h_{j-1}}(\sigma) + \bar{P}(L_t^-(\sigma,\sigma')) + \bar{\beta}_{h_j}(\sigma')\} \end{aligned} \tag{3.13}$$

If there is only one branch connecting a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$, $\bar{P}(L_t^{\pm}(\sigma,\sigma'))$ is replaced with the probability of the branches, $\bar{P}(b_{h_j}(\sigma,\sigma'))$, corresponding to the appropriate bits. Evaluating each part of the LLR in (3.13), requires 2 additions for each branch labelled appropriately in each section of a trellis, other than in the first, T_1, and last, T_v. For these sections, considering that $\bar{\alpha}_0(\sigma_0) = 0$ and $\bar{\beta}_v(\sigma_f) = 0$, only 1 addition, for each branch with appropriate label, is required. Since the results are stored, there is no need for repeating the computations, and thus, this needs to be done for only 1 bit. However, these results need to be compared for each bit. Finally the LLR of the estimated code bits for a section of length ℓ_j, requires ℓ_j subtractions. In summary, to obtain the soft output for each bit in T_j in which only one branch connects a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$, the following number of operations is required:

$$\begin{aligned} \textbf{comparisons}: \quad & 2 \cdot (|B_j^c|/2 - 1) \cdot \ell_j, \quad \text{for} \quad 1 \leq j \leq v \\ \textbf{additions}: \quad & \begin{cases} 2 \cdot (|B_j^c|/2) + \ell_j, & \text{for} \quad j = 1, v, \\ 2 \cdot (2|B_j^c|/2) + \ell_j, & \text{for} \quad 1 < j < v. \end{cases} \end{aligned}$$

However, if the size of the composite branch exceeds one, an efficient way to evaluate (3.13), is to first compute the following:

$$S = \max_{(\sigma,\sigma')} \{\bar{\alpha}_{h_{j-1}}(\sigma) + \bar{P}(L_{h_j}(\sigma,\sigma')) + \bar{\beta}_{h_j}(\sigma')\} \tag{3.14}$$

This requires 2 additions for each composite branch, in every section, T_j, other than in the

first, T_1, and in the last, T_v. For these sections, 1 addition is required, considering that $\bar{\alpha}_0(\sigma_0) = 0$ and $\bar{\beta}_v(\sigma_f) = 0$. These results are then compared. In summary, for each T_j, the following number of operations is needed:

$$\begin{aligned} &\textbf{comparisons}: \quad |B_j^c| - 1, \quad \text{for} \quad 1 \le j \le v \\ &\textbf{additions}: \quad \begin{cases} |B_j^c|, & \text{for} \quad j = 1, v, \\ 2 \cdot |B_j^c|, & \text{for} \quad 1 < j < v. \end{cases} \end{aligned}$$

The first part of the LLR in (3.13), corresponding to the $+1$ bit, is evaluated for each bit in a section, using partial results of the previous step where the addition of $\bar{\alpha}_{h_{j-1}}(\sigma)$ and $\bar{\beta}_{h_j}(\sigma')$ was already computed. Hence, only one addition is needed for each composite branch, and the results are then compared. This needs to be done for each bit in T_j. In summary, the following number of operations is needed:

$$\begin{aligned} &\textbf{comparisons}: \quad (|B_j^c| - 1) \cdot \ell_j, \quad \text{for} \quad 1 \le j \le v \\ &\textbf{additions}: \quad |B_j^c| \cdot \ell_j, \quad \text{for} \quad 1 \le j \le v \end{aligned}$$

The second part of the LLR, corresponding to the -1 bit, is obtained by logic operations, requiring no computations. Finally, the LLR of the estimated code bits in each section for a section of length ℓ_j, requires ℓ_j subtractions.

The MAP algorithm provides the soft output, $L(\hat{c}_t)$, for each estimated code bit, $\hat{c}_t$, for $h_{j-1} + 1 \le t \le h_j$, in T_j in the form of the LLR:

$$L(\hat{c}_t) = \log \frac{\sum\limits_{\substack{(\sigma,\sigma') \\ u_t=+1}} \alpha_{h_{j-1}}(\sigma) P(L_t^+(\sigma,\sigma')) \beta_{h_j}(\sigma')}{\sum\limits_{\substack{(\sigma,\sigma') \\ u_t=-1}} \alpha_{h_{j-1}}(\sigma) P(L_t^-(\sigma,\sigma')) \beta_{h_j}(\sigma')} \tag{3.15}$$

If the size of a composite branch in T_j does not exceed one, evaluating each part of the LLR in (3.15) requires 2 multiplications for each composite branch in a section other than the first and last in which 1 multiplication is required. This needs to be done for only 1 bit in a section, since the results are stored. However, for each bit, these results need to be added. Finally the LLR of the estimated code bits for a section of length ℓ_j, requires ℓ_j divisions. In summary, to obtain

the soft output for each bit in T_j in which only one branch connects a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$, the following number of operations is required:

$$\begin{aligned} &\textbf{additions}: && 2 \cdot (|B_j^c|/2 - 1) \cdot \ell_j, && \text{for} \quad 1 \leq j \leq v \\ &\textbf{multiplications}: && \begin{cases} |B_j^c| + \ell_j, & \text{for} \quad j = 1, v, \\ 2 \cdot |B_j^c| + \ell_j, & \text{for} \quad 1 < j < v. \end{cases} \end{aligned}$$

However, if the size of the composite branch exceeds one, an efficient way to evaluate (3.15), is to first compute the following:

$$S = \sum_{(\sigma,\sigma')} \alpha_{h_{j-1}}(\sigma) P(L_{h_j}(\sigma, \sigma')) \beta_{h_j}(\sigma') \tag{3.16}$$

This requires 2 multiplications for each composite branch, in every T_j, except in T_1 and T_v, in which 1 multiplication is required, considering that $\alpha_0(\sigma_0) = 1$ and $\beta_v(\sigma_f) = 1$. These results are then added. In summary, for each T_j, the following number of operations is needed:

$$\begin{aligned} &\textbf{additions}: && |B_j^c| - 1, && \text{for} \quad 1 \leq j \leq v \\ &\textbf{multiplications}: && \begin{cases} |B_j^c|, & \text{for} \quad j = 1, v, \\ 2 \cdot |B_j^c|, & \text{for} \quad 1 < j < v. \end{cases} \end{aligned}$$

The numerator of the LLR in (3.15), is evaluated for each bit in a section, using the results of the multiplication of $\alpha_{h_{j-1}}(\sigma)$ and $\beta_{h_j}(\sigma')$ from the previous step. Hence, only one multiplication is needed for each composite branch, and the results are then added. In summary, the following number of additions and multiplications is needed:

$$\begin{aligned} &\textbf{additions}: && (|B_j^c| - 1) \cdot \ell_j, && \text{for} \quad 1 \leq j \leq v \\ &\textbf{multiplications}: && |B_j^c| \cdot \ell_j && \text{for} \quad 1 \leq j \leq v \end{aligned}$$

The denominator of the LLR is obtained by taking the difference between the results obtained from the evaluation of the numerator of the LLR and those obtained in (3.16) for each bit in T_j, requiring ℓ_j subtractions in total. Finally, the LLR is evaluated for each bit in T_j, requiring ℓ_j divisions.

The computations required by each algorithm examined for each decoding step in one section, T_j, of a trellis are summarized at the end of the chapter in Table 3.1. The total number of computations required by each algorithm, $F(T_j)$, for each section is the sum of these.

3.2.2 Soft Output Viterbi Algorithm

This section investigates the computational complexity of SOVA for decoding one section T_j, from time h_{j-1} to time h_j, of length $\ell_j = h_j - h_{j-1}$, for a trellis sectionalized into v sections, with section boundaries $\{h_0, h_1, \ldots, h_v\}$. The sum of the required computations for each T_j, for $1 \leq j \leq v$, determines the total computational complexity of SOVA based on a sectionalized trellis.

The computational complexity of SOVA based on a sectionalized trellis, consists of obtaining the probabilities of each branch and of each composite branch, forward and backward state probabilities, and the approximation of the soft output for each code bit. These probabilities are computed in the log domain, and are denoted as $\bar{P}$, $\bar{\alpha}$, and $\bar{\beta}$.

Branch Probability

The branch probability, $\bar{P}(b_{h_j}(\sigma, \sigma'))$, representing the probability of the encoder reaching a state $\sigma' \in S_{h_j}$ from a state $\sigma \in S_{h_{j-1}}$ through that branch in T_j, is defined in (3.4), as:

$$\bar{P}(b_{h_j}(\sigma, \sigma')) \propto \frac{2}{N_0} \sum_{m=h_{j-1}+1}^{h_j} r_m u_m$$

As for Viterbi and Max-Log-MAP algorithms, for SOVA it also suffices to compute only $\sum_{m=h_{j-1}+1}^{h_j} r_m u_m$ from the above expression. The methods described in Section 3.2.1 to compute branch probabilities are applied here to compute $\sum_{m=h_{j-1}+1}^{h_j} r_m u_m$ efficiently. Hence, the same number of additions is required for SOVA as for Viterbi and Max-Log-MAP algorithms.

Composite Branch Probability

The composite branch probability, $\bar{P}(L_{h_j}(\sigma, \sigma'))$, is necessary to be computed if there is more than one branch connecting a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$. Its definition for SOVA is

equivalent to that for Viterbi and Max-Log-MAP algorithms, as in (3.5),

$$\bar{P}(L_{h_j}(\sigma,\sigma')) = \max_{b(\sigma,\sigma')\in L(\sigma,\sigma')}\{\bar{P}(b_{h_j}(\sigma,\sigma'))\}$$

Therefore, to compute the above expression for SOVA, the same number of operations is required as for Viterbi and Max-Log-MAP: $|B_j^d| \cdot (|B_j^p| - 1)$ comparisons. Since the objective of SOVA is also to maximize the overall path probability, as it is for Viterbi and Max-Log-MAP algorithms, if the parallel branches in each composite branch in T_j are self-complementary the number of comparisons may be reduced to: $|B_j^d| \cdot (|B_j^p|/2 - 1)$.

Forward Recursion

The state probabilities in the forward recursion from time instant $h_j = 0$ to $h_j = N$, are defined for SOVA, as for Viterbi and Max-Log-MAP algorithms in (3.7), as:

$$\bar{\alpha}_{h_j}(\sigma') = \max_{\sigma\in\Omega_{h_{j-1}}(\sigma')}\{\bar{P}(L_{h_j}(\sigma,\sigma')) + \bar{\alpha}_{h_{j-1}}(\sigma)\}, \qquad \text{with } \bar{\alpha}_0(\sigma_0) = 0.$$

If the size of the composite branch does not exceed one, then $\bar{P}(b_{h_j}(\sigma,\sigma'))$ is used in place of $\bar{P}(L_{h_j}(\sigma,\sigma'))$.

Taking into account that $\bar{\alpha}_0(\sigma_0) = 0$, for computing the above expression, the same number of operations is required for SOVA as for Viterbi and Max-Log-MAP algorithms. In summary, for each T_j:

$$\textbf{comparisons}: \quad |B_j^c| - |S_{h_j}|, \quad \text{for} \quad 1 \le j \le v$$

$$\textbf{additions}: \quad \begin{cases} 0, & \text{for} \quad T_1, \\ |B_j^c|, & \text{for} \quad 1 < j \le v. \end{cases}$$

Backward Recursion

The state probabilities in the backward recursion from time instant $h_j = N$ to $h_j = 0$ are defined for SOVA, as for Max-Log-MAP algorithm in (3.9), as:

$$\bar{\beta}_{h_{j-1}}(\sigma) = \max_{\sigma'\in\Psi_{h_j}(\sigma)}\{\bar{P}(L_{h_j}(\sigma,\sigma')) + \bar{\beta}_{h_j}(\sigma')\}, \quad \text{with } \bar{\beta}_v(\sigma_f) = 0.$$

As for the forward recursion, if the size of the composite branch, in T_j, does not exceed one

then $\bar{P}(L_{h_j}(\sigma,\sigma'))$ is replaced with $\bar{P}(b_{h_j}(\sigma,\sigma'))$ in the above expression.

Taking into account that $\bar{\beta}_v(\sigma_f)=0$, the same number of operations is required for SOVA as for Max-Log-MAP algorithm. Summarizing for each T_j:

$$\begin{aligned}\textbf{comparisons}: &\quad |B_j^c|-|S_{h_{j-1}}|, \quad \text{for} \quad 1\le j\le v\\ \textbf{additions}: &\quad \begin{cases}0, & \text{for} \quad T_v,\\ |B_j^c|, & \text{for} \quad 1\le j<v.\end{cases}\end{aligned}$$

Composite Bit Probability

If the size of a composite branch exceeds one, it is necessary to find the composite bit probability, $\bar{P}(L_\ell^{\pm}(\sigma,\sigma'))$, representing the probability of each possible encoder output for each bit u_t within a composite branch, $L_{h_j}(\sigma,\sigma')$:

For SOVA, $\bar{P}(L_t^{\pm}(\sigma,\sigma'))$ is defined, as for Max-Log-MAP, in (3.11), as :

$$\bar{P}(L_t^{\pm}(\sigma,\sigma'))=\max_{\substack{b(\sigma,\sigma')\in L(\sigma,\sigma')\\ u_t=\pm 1}}\{\bar{P}(b_{h_j}(\sigma,\sigma'))\},\quad \text{for } h_{j-1}+1\le t\le h_j.$$

Same procedure is followed for obtaining $\bar{P}(L_t^{\pm}(\sigma,\sigma'))$ for SOVA, as for Max-Log-MAP, hence, resulting in the following number of required comparisons:

$$\textbf{comparisons}: \quad (|B_j^p|/2-1)\cdot|B_j^d|\cdot\ell_j, \quad \text{for} \quad 1\le j\le v$$

Soft Output

For SOVA, the soft output, $L(\hat{c}_t)$, for each decoded bit, $\hat{c}_t$, for $h_{j-1}+1\le t\le h_j$, in T_j is approximated by:

$$L(\hat{c}_t)\approx\bar{P}(c_t=0|\mathbf{r})-\bar{P}(c_t=1|\mathbf{r}),\quad \text{for}\quad h_{j-1}+1\le t\le h_j. \tag{3.17}$$

For each bit in T_j, the probability of the ML path, $\bar{\alpha}_v(\sigma_f)$, is assigned to the part of the above expression corresponding to the ML estimate for that bit, $\hat{c}_t$. The probability of the most probable path whose label is complementary, x, to the ML estimate $\hat{c}_t$ is defined as:

$$\bar{P}(c_t=x|\mathbf{r})=\max_{\substack{(\sigma,\sigma')\\ c_t=x}}\{\bar{\alpha}_{h_{j-1}}(\sigma)+\bar{P}(L_t^{sign(x)}(\sigma,\sigma'))+\bar{\beta}_{h_j}(\sigma')\} \tag{3.18}$$

If there is only one branch connecting a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$, $\bar{P}(L_t^{\pm}(\sigma, \sigma'))$ is replaced with the probability of the branch, $\bar{P}(b_{h_j}(\sigma, \sigma'))$, whose label has value x for bit c_t. Evaluating (3.18) for each bit, 2 additions are necessary for all the branches, corresponding to the complement of the ML estimate, in a section other than in T_1 and T_v. For the first bit, $2 \cdot |B_j^c|/2$ additions are necessary. However, for the subsequent bits of that section, taking into account that all computations are stored and there is no need for repetition, only the branches, with complement bits to the ML estimate that have not yet been considered, require additions. Hence, for the entire section, all the branches will need to be considered except the ones representing the ML estimate, resulting in $|B_j^c| - |B_j^c|/|B_j^d|$ branches. Sections T_1 and T_v required one addition for the same number of branches. These results are compared for each bit to find the maximum. Computing the LLR, in (3.17), of the estimated code bits for a section of length ℓ_j, requires ℓ_j subtractions. In summary, to obtain the soft output for each bit in T_j in which only one branch connects a state $\sigma \in S_{h_{j-1}}$ to a state $\sigma' \in S_{h_j}$, the following number of operations is required:

$$\begin{aligned}
&\textbf{comparisons}: && (|B_j^c|/2 - 1) \cdot \ell_j, && \text{for} \quad 1 \leq j \leq v \\
&\textbf{additions}: && \begin{cases} 1 \cdot (|B_j^c| - |B_j^c|/|B_j^d|) + \ell_j, & \text{for} \quad j = 1, v, \\ 2 \cdot (|B_j^c| - |B_j^c|/|B_j^d|) + \ell_j, & \text{for} \quad 1 < j < v. \end{cases}
\end{aligned}$$

However, if the size of the composite branch exceeds one, evaluating the part of the LLR in (3.18) corresponding to the complementary bit, for a section other than the first or last, requires 2 additions for the first bit for each composite branch, and for the remaining $\ell_j - 1$ bits in T_j, only 1, since the addition of $\bar{\alpha}_{h_{j-1}}(\sigma)$ and $\bar{\beta}_{h_j}(\sigma')$ was already computed. For T_1 and T_v, only one addition is needed for each composite branch for all ℓ_j bits. These results are compared for each bit. Finally the LLR of the estimated code bits for a section of length ℓ_j, requires ℓ_j subtractions. In summary, the following number of operations is needed:

$$\begin{aligned}
&\textbf{comparisons}: && (|B_j^c| - 1) \cdot \ell_j, && \text{for} \quad 1 \leq j \leq v \\
&\textbf{additions}: && \begin{cases} |B_j^c| \cdot \ell_j + \ell_j, & \text{for} \quad j = 1, v, \\ 2 \cdot |B_j^c| + |B_j^c| \cdot (\ell_j - 1) + \ell_j, & \text{for} \quad 1 < j < v. \end{cases}
\end{aligned}$$

The computations required by SOVA for each decoding step in one section, T_j, of a trellis are presented in Table 3.1. The sum of these, represents the total number of computations required

by SOVA, $F(T_j)$, for decoding each section.

3.3 Summary

A minimal trellis was sectionalized using the section boundaries obtained from the optimum sectionalization algorithm. The computational complexities for the decoding steps of Viterbi, Max-Log-MAP, and MAP algorithms based on a sectionalized trellis were examined, and their computationally efficient methods were applied to determine the computational complexity of SOVA.

Table 3.1 summarizes the computational operations required for each decoding step of the algorithms considered for decoding one section T_j of a sectionalized trellis for linear block code. The total number of computations, $F(T_j)$, required by each algorithm for decoding T_j is the sum of these. These $F(T_j)$ values are used in the implementation of the computation-wise optimum sectionalization algorithm, presented in Section 3.1.3.

The computation-wise optimum sectionalizations for the algorithms of interest for different RM codes are presented in the next Chapter. Their decoding complexities are discussed and compared to the much larger computational complexities required for decoding of a bit-level trellis.

Table 3.1 Computational complexities of Viterbi, SOVA, Max-Log-MAP, and MAP algorithms for decoding a section of a trellis.

Decoding Steps	Viterbi	SOVA	Max-Log-MAP	MAP
Branch Probabilities				
Comparisons	0	0	0	0
Additions	$\|B^d_j\|\|B^p_j\|(\ell_j - 1)$ $\|B^d_j\|\|B^p_j\|(\ell_j - 1)/2$ $2^{\ell_j/2} + \ell_j - 4 + \frac{2^{\ell_j}}{4}$ $\ell_j + 2^{\ell_j - 1} - 2$	$\|B^d_j\|\|B^p_j\|(\ell_j - 1)$ $\|B^d_j\|\|B^p_j\|(\ell_j - 1)/2$ $2^{\ell_j/2} + \ell_j - 4 + \frac{2^{\ell_j}}{4}$ $\ell_j + 2^{\ell_j - 1} - 2$	$\|B^d_j\|\|B^p_j\|(\ell_j - 1)$ $\|B^d_j\|\|B^p_j\|(\ell_j - 1)/2$ $2^{\ell_j/2} + \ell_j - 4 + \frac{2^{\ell_j}}{4}$ $\ell_j + 2^{\ell_j - 1} - 2$	$\|B^d_j\|\|B^p_j\|(\ell_j - 1)$ $\|B^d_j\|\|B^p_j\|(\ell_j - 1)/2$ $2^{\ell_j/2} + \ell_j - 4 + \frac{2^{\ell_j}}{4}$ $\ell_j + 2^{\ell_j - 1} - 2$
Multiplications	0	0	0	ℓ_j
Composite Branch Probabilities				
Comparisons	$\|B^d_j\|(\|B^p_j\| - 1)$ $\|B^d_j\|(\|B^p_j\|/2 - 1)$	$\|B^d_j\|(\|B^p_j\| - 1)$ $\|B^d_j\|(\|B^p_j\|/2 - 1)$	$\|B^d_j\|(\|B^p_j\| - 1)$ $\|B^d_j\|(\|B^p_j\|/2 - 1)$	0
Additions	0	0	0	$\|B^d_j\|$
Multiplications	0	0	0	0
Forward Recursion				
Comparisons	$\|B^c_j\| - \|S_{h_j}\|$	$\|B^c_j\| - \|S_{h_j}\|$	$\|B^c_j\| - \|S_{h_j}\|$	0
Additions	0, for $j = 1$ $\|B^c_j\|, for 1 < j \leq v$	0, for $j = 1$ $\|B^c_j\|, for 1 < j \leq v$	0, for $j = 1$ $\|B^c_j\|, for 1 < j \leq v$	$\|B^c_j\| - \|S_{h_j}\|$
Multiplications	0	0	0	0, for $j = 1$ $\|B^c_j\|, for 1 < j \leq v$
Backward Recursion	N/A			
Comparisons		$\|B^c_j\| - \|S_{h_{j-1}}\|$	$\|B^c_j\| - \|S_{h_{j-1}}\|$	0
Additions		0, for $j = v$ $\|B^c_j\|, for 1 \leq j < v$	0, for $j = v$ $\|B^c_j\|, for 1 \leq j < v$	$\|B^c_j\| - \|S_{h_{j-1}}\|$
Multiplications		0	0	0, for $j = v$ $\|B^c_j\|, for 1 \leq j < v$
Composite Bit Probabilities	N/A			
Comparisons		$\|B^d_j\|(\|B^p_j\|/2 - 1)\ell_j$	$\|B^d_j\|(\|B^p_j\|/2 - 1)\ell_j$	0
Additions		0	0	$\|B^d_j\|(\|B^p_j\| - 2)\ell_j$
Multiplications		0	0	0

Table 3.1 Computational complexities of Viterbi, SOVA, Max-Log-MAP, and MAP algorithms for decoding a section of a trellis.

Decoding Steps	**Viterbi**	**SOVA**	**Max-Log-MAP**	**MAP**
Soft Output	N/A			
If $\|B_j^p\| = 1$				
Comparisons		$(\|B_j^c\|/2 - 1)\ell_j$	$(\|B_j^c\| - 2)\ell_j$	0
Additions		$\|B_j^c\|(1 - 1/\|B_j^d\|) + \ell_j$, $for j = 1, v$ $2\|B_j^c\|(1 - 1/\|B_j^d\|) + \ell_j$, $for 1 < j < v$	$\|B_j^c\| + \ell_j$, $for j = 1, v$ $2\|B_j^c\| + \ell_j$, $for 1 < j < v$	$(\|B_j^c\| - 2)\ell_j$
Multiplications		0	0	$\|B_j^c\| + \ell_j$, $for j = 1, v$ $2\|B_j^c\| + \ell_j$, $for 1 < j < v$
If $\|B_j^p\| > 1$				
Comparisons		$(\|B_j^c\| - 1)\ell_j$	$(\|B_j^c\| - 1)(\ell_j + 1)$	0
Additions		$(\|B_j^c\| + 1)\ell_j$, $for j = 1, v$ $(\|B_j^c\| + 1)\ell_j + \|B_j^c\|$, $for 1 < j < v$	$(\|B_j^c\| + 1)\ell_j + \|B_j^c\|$, $for j = 1, v$ $(\|B_j^c\| + 1)\ell_j + 2\|B_j^c\|$, $for 1 < j < v$	$\|B_j^c\|\ell_j + \|B_j^c\| - 1$
Multiplications		0	0	$(\|B_j^c\| + 1)\ell_j + \|B_j^c\|$, $for j = 1, v$ $(\|B_j^c\| + 1)\ell_j + 2\|B_j^c\|$, $for 1 < j < v$

Chapter 4

Simulation Results

The first part of this chapter, Section 4.1 presents, analyzes, and compares the computational complexities of SOVA based on the sectionalized trellises for different RM codes, with those required by SOVA for the decoding of bit-level trellises. The obtained complexities of Viterbi, Max-Log-MAP, and MAP algorithms are included for comparisons. In the second part, Section 4.2, the simulation results of BER performance over AWGN channel for SOVA based on a sectionalized trellis are examined. These are compared with those attained for the bit-level trellises and with the BER performances of Viterbi, Max-Log-MAP, and MAP algorithms. This section further analyzes the BER performances of the decoders, examined in this thesis, when applied to a concatenated block code scheme. Also included are the BER performances of using the proposed SOVA and the conventional SOVA as the component decoders in an iterative decoding of concatenated block codes scheme. The chapter concludes with the summary of the results.

4.1 Computational Complexity Evaluation

The total number of computations required by the algorithms considered, to decode a bit-level trellis was obtained by applying the analysis of Section 2.2 to each unit section. Each bit-level trellis was sectionalized using the optimum section boundaries obtained by applying the computation-wise optimum sectionalization algorithm outlined in Section 3.1.3. The computational complexities of SOVA for decoding sectionalized trellises of linear block codes are attained by implementing the analysis of Section 3.2.2 to each section. The quantities are shown to be smaller than the corresponding ones of bit-level trellises. These are compared to the total complexities of Viterbi, Max-Log-MAP, and MAP algorithms when applied to optimally section-

alized trellises of the same codes.

Viterbi, SOVA, and Max-Log-MAP algorithms involve only comparisons and additions. These operations are considered equally complicated, and as such, were added to obtain the total computational complexity of the algorithms. MAP algorithm, on the other hand, involves additions and multiplications. As a multiplication is more complex than an addition, these operations cannot be treated the same. Optimum trellis sectionalizations were obtained that minimize the number of required additions and weighted multiplications. The weighting factor depends on the specific chip implementation. In this thesis, it is assumed that the chip performs five times more operations for a multiplication than for an addition. As such, a weighting factor of 5 was used for a multiplication operation. The total computational complexity of the MAP algorithm is defined as the sum of additions and weighted multiplications.

4.1.1 Bit-Level Trellis

Table 4.1 lists the total computational complexities for decoding bit-level trellises using the algorithms considered for the specified linear block codes. These were obtained by summing all the computations required for the decoding of each section in a bit-level trellis, specified in Section 2.2. Normalized complexity represents the number of operations required to decode one information bit.

As in [39], the more optimal the algorithms are, in terms of BER, the more computations they require. The computational complexity of SOVA, for the decoding of all the specified linear block codes, exceeds the complexity of the Viterbi decoding of the same codes, by about 70%. However, Viterbi algorithm provides only hard outputs that are not sufficient in many error control schemes, including BTCs. SOVA is significantly less complicated than the MAP algorithm, requiring only up to 20% of the computations required by MAP. Requiring an extensive number of operations, MAP algorithm is unfit for implementations in many communication systems. Max-Log-MAP algorithm requires at least 25% more computations than SOVA. This certifies SOVA as the preferred algorithm of the ones examined in this thesis for bit-level trellis decoding. In the next section, it is shown that the computational complexity of SOVA is further reduced by means of sectionalizing a trellis.

Table 4.1 Computational complexities of Viterbi, SOVA, Max-Log-MAP, and MAP algorithms based on a bit-level trellis of RM codes.

	Computational Complexity (# of operations)			
(N, K) RM Codes	Viterbi	SOVA	Max-Log-MAP	MAP
(8, 4)				
Complexity	53	170	226	970
Normalized Complexity	13.3	42.5	56.5	242.5
(16, 5)				
Complexity	193	642	882	3,746
Normalized Complexity	38.6	128.4	176.4	749.2
(16,11)				
Complexity	353	1,082	1,442	5,586
Normalized Complexity	32.1	98.4	131.1	507.8
(32, 6)				
Complexity	729	2,482	3,474	14,674
Normalized Complexity	121.5	413.7	579	2,445.7
(32,16)				
Complexity	7,993	25,578	35,138	137,730
Normalized Complexity	499.6	1,598.6	2,196.1	8,608.1
(32,26)				
Complexity	1,721	5,210	6,946	26,082
Normalized Complexity	66.2	200.4	267.2	1,003.2
(64, 7)				
Complexity	2,825	9,746	13,778	58,034
Normalized Complexity	403.6	1,392.3	1,968.3	8,290.6
(64,22)				
Complexity	425,209	1,412,970	1,975,460	7,976,580
Normalized Complexity	19,327.7	64,225.9	89,793.6	362,571.8
(64,42)				
Complexity	773,881	2,371,820	3,195,810	11,986,300
Normalized Complexity	18,425.7	56,471.9	76,090.7	285,388.1
(64,57)				
Complexity	7,529	22,682	30,242	112,130
Normalized Complexity	132.1	397.9	530.6	1,967.2

4.1.2 Optimally Sectionalized Trellis for SOVA

Table 4.2 compares the total computational complexity required by SOVA to decode a bit-level trellis and an optimally sectionalized one.

It is observed from Table 4.2, that optimum sectionalization for SOVA always results in a mirror symmetry of the trellis. Optimum sectionalization is also uniform for RM (8,4), RM (16,5), and RM (32,16) codes.

As was expected, optimum sectionalization does reduce decoding complexity of SOVA. The greatest reduction, of 68%, is achieved for the RM (64,7) code. The SOVA decoding based on the optimum sectionalized trellis for (32,6) and (64,22) RM codes saves 60% and 54% of computations, respectively, compared to decoding of the same codes based on the bit-level trellis. The total complexities of decoding (8,4), (16,5), and (32,16) RM codes, resulting from the optimum sectionalization with uniform boundary locations, are about 45%, 50%, and 34% less, respectively, than the complexity required for the decoding of the same codes based on a bit-level trellis. For the remaining codes, RM (16,11), RM (32,26), RM (64,42), RM (64,57), savings of 15%, 6%, 12%, and 3%, respectively, are achieved. These results confirm that a SOVA decoder based on a sectionalized trellis is notably more computationally efficient than the decoder based on a bit-level trellis.

4.1.3 Computational Comparisons of Decoders

In this section, the results of optimum sectionalizations of the RM codes, specified in the previous section, for SOVA are compared to the optimum sectionalizations and the corresponding complexities for Viterbi, Max-Log-MAP, and MAP algorithms.

The optimum section boundaries resulting from the computation-wise optimum sectionalizations of bit-level trellises for the linear block codes considered, for the algorithms of interest, and their complexities, are illustrated in Table 4.3.

It is noted that optimum sectionalizations, of RM codes considered, for Max-Log-MAP and MAP algorithms always results in a mirror symmetry. For MAP decoder, uniform sectionalization is optimum for all of the RM codes considered except for (32,26), (64,42), and (64,57) RM codes. For Max-Log-MAP optimum sectionalization is also uniform for (8,4) and (16,5) RM codes. On the other hand, mirror symmetry for Viterbi decoder is obtained only for (8,4), (16,11), (32,16), (32,26), (64,42), and (64,57) RM codes. For all these codes, except for RM (64,57), optimum sectionalization is also uniform.

Table 4.2 Computational complexities of SOVA based on a bit-level trellis and on an optimally sectionalized trellis of RM codes.

	Computational Complexities	
(N, K) RM Codes	Bit-level	Optimally Sectionalized
(8,4)		
Optimum Boundaries		{0,4,8}
Complexity	170	94
Normalized Complexity	42.5	23.5
(16,5)		
Optimum Boundaries		{0,4,8,12,16}
Complexity	642	324
Normalized Complexity	128.4	64.8
(16,11)		
Optimum Boundaries		{0,4,6,8,10,12,16}
Complexity	1,082	922
Normalized Complexity	98.4	83.8
(32,6)		
Optimum Boundaries		{0,4,8,16,24,28,32}
Complexity	2,482	988
Normalized Complexity	413.7	164.7
(32,16)		
Optimum Boundaries		{0,4,8,12,16,20,24,28,32}
Complexity	25,578	16,948
Normalized Complexity	1,598.6	1,059.3
(32,26)		
Optimum Boundaries		{0,4,6,7,8,10,11,12,13,14,16,18,19,20,21,22,24,25,26,28,32}
Complexity	5,210	4,910
Normalized Complexity	200.4	188.9
(64,7)		
Optimum Boundaries		{0,4,8,16,32,48,56,60,64}
Complexity	9,746	3,092
Normalized Complexity	1,392.3	441.7
(64,22)		
Optimum Boundaries		{0,5,9,16,17,24,31,32,33,40,47,48,55,59,64}
Complexity	1,412,970	654,340
Normalized Complexity	64,225.9	29,742.7
(64,42)		
Optimum Boundaries		{0,6,8,10,12,14,16,18,20,22,24,26,28,30,32, 34,36,38,40,42,44,46,48,50,52,54,56,58,64}
Complexity	2,371,820	2,097,140
Normalized Complexity	56,471.9	49,931.9

Table 4.2 Computational complexities of SOVA based on a bit-level trellis and on an optimally sectionalized trellis of RM codes.

	Computational Complexities	
(N, K) RM Codes	Bit-level	Optimally Sectionalized
(64,57)		
Optimum Boundaries		{0,4,6,7,8,10,11,12,13,14,15,16,18,19,20,21,22,23,24,25,26, 27,28,29,30,32,34,35,36,37,38,39,40,41,42,43,44,45,46,48, 49,50,51,52,53,54,56,57,58,60,64}
Complexity	22,682	22,098
Normalized Complexity	397.9	387.7

The optimum section boundaries obtained for SOVA decoding match those for Viterbi decoding of only RM (8,4) code, and those for MAP decoding of RM (32,16) code. While for Max-Log-MAP decoding they match the section boundaries of all the codes considered except for (32,16), (64,22), and (64,42) RM codes.

As was shown in existing literature, [3] and [4], sectionalization of a trellis for Viterbi, Max-Log-MAP, and MAP decoding results in a reduced computational complexity. For example, for the decoding of (32,6) and (64,7) RM codes, the computational complexity required by MAP decoder is reduced by more than 80% and that required by Max-Log-MAP is reduced by more than 60%. For Viterbi decoding of (32,16), (64,7), and (64,22) RM codes, the computational complexity is reduced by more than 70%.

It is observed that the algorithms studied remain in the same order of complexity for sectionalized trellis decoding as for bit-level trellis decoding. SOVA still requires more computations than Viterbi, but less than Max-Log-MAP, and MAP decoding is the most computationally complicated one.

Comparisons with Viterbi Algorithm

The complexity of a SOVA decoder remains higher than that of a ML Viterbi decoder, based on a sectionalized trellis, by at least 71%. The differences in their computational complexities are due to the calculations of backward state probabilities, composite bit probabilities, and soft output, all that are required by SOVA and not by Viterbi.

Table 4.3 Optimum sectionalizations of RM codes and computational complexities of Viterbi, SOVA, Max-Log-MAP, and MAP algorithms.

	Optimum sectionalizations and computational complexities			
(N, K) RM Codes	Viterbi	SOVA	Max-Log-MAP	MAP
(8,4)				
Boundaries	{0,4,8}	{0,4,8}	{0,4,8}	{0,8}
Complexity	23	94	108	257
Normalized	5.75	23.5	27	64.25
(16,5)				
Boundaries	{0,4,8,12,15,16}	{0,4,8,12,16}	{0,4,8,12,16}	{0,16}
Complexity	94	324	414	897
Normalized	18.8	64.8	82.8	179.4
(16,11)				
Boundaries	{0,4,8,12,16}	{0,4,6,8,10,12,16}	{0,4,6,8,10,12,16}	{0,8,16}
Complexity	167	922	1,136	3,092
Normalized	15.2	83.8	103.3	281.1
(32,6)				
Boundaries	{0,4,8,12,16, 20,24,28,31,32}	{0,4,8,16, 24,28,32}	{0,4,8,16, 24,28,32}	{0,8,16, 24,32}
Complexity	278	988	1,326	2,942
Normalized	46.3	164.7	221	490.3
(32,16)				
Boundaries	{0,8,16, 24,32}	{0,4,8,12,16, 20,24,28,32}	{0,3,5,8,12,16, 20,24,27,29,32}	{0,4,8,12,16, 20,24,28,32}
Complexity	2,383	16,948	21,870	59,278
Normalized	148.9	1,059.3	1,366.9	3,704.9
(32,26)				
Boundaries	{0,4,8, 12,16, 20,24, 28,32}	{0,4,6,7,8, 10,11,12,13,14,16, 18,19,20,21,22,24, 25,26,28,32}	{0,4,6,7,8, 10,11,12,13,14,16, 18,19,20,21,22,24, 25,26,28,32}	{0,7,8,10,11, 12,13,14,16, 18,19,20,21, 22,24,25,32}
Complexity	1,255	4,910	6,356	22,154
Normalized	48.27	188.9	244.5	852.1
(64,7)				
Boundaries	{0,4,8,12,16,24,32, 40,48,52,56,60,63,64}	{0,4,8,16,32, 48,56,60,64}	{0,4,8,16,32, 48,56,60,64}	{0,16,32, 48,64}
Complexity	806	3,092	4,430	7,934
Normalized	115.1	441.7	632.9	1,133.4

Table 4.3. Optimum sectionalizations of RM codes and computational complexities of Viterbi, SOVA, Max-Log-MAP, and MAP algorithms

	Optimum sectionalizations and computational complexities			
(N, K) RM Codes	Viterbi	SOVA	Max-Log-MAP	MAP
(64,22)				
Boundaries	{0,8,16,32, 48,56,61,63,64}	{0,5,9,16,17,24,31,32, 33,40,47,48,55,59,64}	{0,3,5,8,10,16,18,24,30,32, 34,40,46,48,54,56,59,61,64}	{0,8,16,24,32, 40,48,56,64}
Complexity	104,370	654,340	903,562	2,055,840
Normalized	4,744.1	29,742.7	41,071	93,447.3
(64,42)				
Boundaries	{0,8,16, 24,32, 40,48, 56,64}	{0,6,8,10,12,14,16, 18,20,22,24,26,28,30,32, 34,36,38,40,42,44,46,48, 50,52,54,56,58,64}	{0,4,6,8,10,12,14,16, 18,20,22,24,26,28,30,32, 34,36,38,40,42,44,46,48, 50,52,54,56,58,60,64}	{0,4,8,12,14,16, 20,22,24,26,28,32, 36,38,40,42,44,48, 50,52,56,60,64}
Complexity	538,799	2,097,140	2,646,470	8,261,180
Normalized	12,828.5	49,931.9	63,011.2	196,694.8
(64,57)				
Boundaries	{0,4,8, 12,13,14,15,16, 20,21,22,23,24, 25,26,27,28,32, 36,37,38,39,40, 41,42,43,44,48, 49,50,51,52,56, 60,64}	{0,4,6,7,8, 10,11,12,13,14,15,16, 18,19,20,21,22,23,24, 25,26,27,28,29,30,32, 34,35,36,37,38,39,40, 41,42,43,44,45,46,48, 49,50,51,52,53,54,56, 57,58,60,64}	{0,4,6,7,8, 10,11,12,13,14,15,16, 18,19,20,21,22,23,24, 25,26,27,28,29,30,32, 34,35,36,37,38,39,40, 41,42,43,44,45,46,48 49,50,51,52,53,54,56, 57,58,60,64}	{0,7,8, 10,11,12,13,14,15,16, 18,19,20,21,22,23,24, 25,26,27,28,29,30,32, 34,35,36,37,38,39,40, 41,42,43,44,45,46,48, 49,50,51,52,53,54,56, 57,64}
Complexity	6,507	22,098	29,080	104,558
Normalized	114.2	387.7	510.2	1,834.4

Comparisons with Max-Log-MAP Algorithm

Similarly, the application of a SOVA decoder, in place of a Max-Log-MAP decoder, to the optimum sectionalized trellises for the linear block codes specified, can save a notable number of computations, up to 30% for RM (64,7) code, and at least 13% for RM (8,4) code. For the remaining codes, the number of required computations is lowered by 28% for (64,22) RM code, by 24% for (32,6), (32,16), (32,26), and (64,57) RM codes, by 22% for (16,5) and (64,42) RM codes, and by 19% for RM (16,11) code with a SOVA decoder.

The differences in computational complexities between Max-Log-MAP and SOVA can be explained from Table 3.1, where it is observed that Max-Log-MAP and SOVA require different number of operations for computing the soft output. As it is explained in Section 3.2.1, Max-Log-MAP considers all branches corresponding to each possible encoder output when computing the LLR of the estimated bits, resulting in the consideration of all the branches for the decoding of one section. Whereas SOVA, for LLR of each bit, considers only branches that are opposite in sign to the ML estimate. Therefore, for the entire section, all of the branches but the ones with the ML label, are considered.

From Table 3.1, it is observed that for all sections in which the size of the composite branch does not exceed one, Max-Log-MAP algorithm requires $(|B_j^c|/2 - 1) \cdot \ell_j$ more comparisons, and in sections other than T_1 and T_v that require $|B_j^c|/|B_j^d|$ additions, $2 \cdot |B_j^c|/|B_j^d|$ more additions are needed than by SOVA. For sections in which there is more than one branch connecting two states, in comparison to SOVA, Max-Log-MAP requires $|B_j^c| - 1$ more comparisons and $|B_j^c|$ more additions. The comparisons of computational complexities required by SOVA and Max-Log-MAP algorithms can be discussed for only those codes for which the optimum section boundaries are matching for the two decoders, including (8,4), (16,5), (16,11), (32,6), and (64,7) RM codes. Considering that optimal sectionalization for these decoders always results in mirror symmetry of the trellis, only half of the trellis needs to be analyzed.

For RM (8,4) code, optimum sectionalization results in {0,4,8} section boundaries. In both sections, there are four composite branches with the size of each one being greater than one. Applying the analysis from above, for Max-Log-MAP, there are $4 - 1 = 3$ more comparisons in each section, and 4 more additions, resulting in a total of $3 \cdot 2 + 4 \cdot 2 = 14$ more operations than for SOVA. As is shown in Table 4.3, SOVA has a computational complexity of 94, and Max-Log-MAP of $94 + 14 = 108$.

For RM (16,5) code, optimum sectionalization results in {0,4,8,12,16} boundaries for both

decoders. In each of the 4 sections, there are 8 distinct composite branches, 8 composite branches in the first and last sections and 16 in the second and third sections, and in all sections the size of the composite branches does not exceed one. From the analysis above, the difference in the number of comparisons between SOVA and Max-Log-MAP for the first two sections are $(8/2-1)\cdot 4 = 12$ and $(16/2-1)\cdot 4 = 28$. The difference in the number of additions for the same sections are $8/8 = 1$ and $2\cdot 16/8 = 4$. Therefore, for the Max-Log-MAP decoding of the entire trellis for RM (16,5) code, and considering that comparisons and additions have the same complexity, $(12+28)\cdot 2+(1+4)\cdot 2 = 90$ more operations are needed than for SOVA. This is illustrated in Table 4.3 above, with SOVA having complexity of 324 and Max-Log-MAP of $324+90 = 414$.

For RM (16,11) code, optimum sectionalization results in {0,4,6,8,10,12,16} section boundaries. In the six sections, the number of distinct composite branches are {8,4,4,4,4,8}, the number of composite branches are {8,32,32,32,32,8}, and the size of the composite branches exceeds one for only the first and last sections. For the first three sections, the number of more comparisons is $\{8-1 = 7, (32/2-1)\cdot 2 = 30, (32/2-1)\cdot 2 = 30\}$, and the number of more additions is $\{8, 2\cdot 32/4 = 16, 2\cdot 32/4 = 16\}$. Max-Log-MAP decoding of the entire trellis will require $(7+30+30)\cdot 2+(8+16+16)\cdot 2 = 214$ more operations than SOVA decoding, as is shown in Table 4.3, with the complexity of SOVA being 922 and of Max-Log-MAP being $922+214 = 1136$.

Similarly, for the RM (32,6), the optimum section boundaries are {0,4,8,16,24,28,32}, the number of distinct branches in the six sections are {8,8,16,16,8,8}, the number of composite branches are {8,16,32,32,16,8}, and the size of composite branches in all sections does not exceed one. The number of more additions required by Max-Log-MAP is $(12+28+120)\cdot 2$, and the number of more comparisons is $(1+4+4)\cdot 2$. This is shown in Table 4.3 with SOVA having a complexity of 988, and Max-Log-MAP of $988+338 = 1326$.

RM (64,7) code also has matching optimum section boundaries {0,4,8,16,32,48,56,60,64} for the two decoders. For the eight sections, the number of distinct composite branches are {8,8,16,32,32,16,8,8}, the number of composite branches are {8,16,32,64,64,32,16,8}, and the size of each composite branch in all sections does not exceed one. Therefore, there are $\{(8/2-1)\cdot 4 = 12, (16/2-1)\cdot 4 = 28, (32/2-1)\cdot 8 = 120, (64/2-1)\cdot 16 = 496\}$ more comparisons and $\{8/8 = 1, 2\cdot 16/8 = 4, 2\cdot 32/16 = 4, 2\cdot 64/32 = 4\}$ more additions for Max-Log-MAP than for SOVA decoding of the first half of the trellis. This reflects the results in Table 4.3, in which SOVA has a complexity of 3092, and Max-Log-MAP of $3092+(12+28+120+496+1+4+4+4)\cdot 2 = 4430$.

Comparisons with MAP Algorithm

When a SOVA decoder, instead of a MAP decoder, is applied to the optimum sectionalized trellis for the linear block codes considered, savings of up to 79%, for (32,26) and (64,57) RM codes, and of at least 64%, for RM (64,7) code, are achieved. The computational operations are reduced by 75% for RM (64,42) code, and by 71% for (16,11) and (32,16) RM codes. For (64,22) and (32,6) RM codes, the complexity is reduced by 68% and 66%, respectively, and for (8,4) and (16,5) RM codes, by 64% with the proposed decoder.

The differences in computational complexities between MAP and SOVA is due to the decoders requiring different types and different number of operations in all of the decoding steps in Table 3.1. For the computation of the branch probabilities, MAP requires ℓ_j multiplications, which is, with the weight factor of 5 for multiplications, $5 \cdot \ell_j$ more operations than SOVA needs for that step. For MAP the composite branch probabilities are computed using the composite bit probabilities, unlike for SOVA, requiring $2 \cdot |B_j^d| - |B_j^d| \cdot |B_j^p|$ more operations, or if the parallel branches are complementary, $2 \cdot |B_j^d| - |B_j^d| \cdot |B_j^p|/2$ more. In the forward and backward recursions, the required comparisons and additions of SOVA are replaced by additions and multiplications, respectively, for MAP. Hence, there are 5 times more operations required by MAP for these decoding steps. For SOVA, only composite bit probability of the complementary ML estimate needs to be computed for each bit, whereas MAP evaluates the composite bit probability corresponding to each encoder output. Therefore, MAP requires double the amount of operations that SOVA requires for this decoding step, which is $(|B_j^p|/2 - 1) \cdot |B_j^d| \cdot \ell_j$ more. For soft output, MAP considers all branches in each section, whereas SOVA considers only branches that are opposite in sign to the ML estimate. It is observed from Table 3.1 that for all sections in which the size of the composite branch does not exceed one, MAP requires additions that are double the amount of comparisons required by SOVA. Also, in sections other than T_1 and T_v that require $|B_j^c|/|B_j^d|$ more, MAP requires $2 \cdot |B_j^c|/|B_j^d|$ more multiplications than SOVA additions. Hence there are $5 \cdot |B_j^c|/|B_j^d|$ for the first and last sections, and $5 \cdot 2 \cdot |B_j^c|/|B_j^d|$ for all other sections, more operations needed by MAP. For the case when the size of the composite branch does exceed one, MAP requires $|B_j^c| - 1 + \ell_j$ more additions than SOVA comparisons. Also $|B_j^c|$ more multiplications are required by MAP than additions required by SOVA, hence, $5 \cdot |B_j^c|$ more operations.

The results in this section establish that the SOVA decoder, introduced in this thesis, is the most computationally efficient SISO decoder examined in this thesis.

Appendix A provides the discussion on the differences with the best results found in the literature. The obtained results of Viterbi are shown to be the same for most linear block codes considered, and the obtained results of Max-Log-MAP and MAP algorithms are shown to be better for all linear block codes considered than those found in the existing literature.

4.2 BER Performance Evaluation

BER represents the ratio of the number of bits in the estimated information sequence that contradict those of the transmitted one to the number of information bits transmitted. The mapping between information sequences and the codewords used by the encoder is obtained from the locations of the K columns of the identity matrix $\mathbf{I}_K$ in the generator matrix $\mathbf{G}$. For example, from the generator matrix $\mathbf{G}$, in (1.1), for the RM (8,4) code, it is observed that the positions of the transmitted information bits $\{m_1, m_2, m_3, m_4\}$ correspond to $\{1,2,3,8\}$ positions in the estimated codeword. The BER performances of the algorithms are evaluated against SNR (E_b/N_o) where E_b denotes the Energy per information bit and $N_o/2$ is the variance of the noise, by simulations under AWGN channel environment with BPSK modulation.

The following section analyzes the BER performances of the algorithms examined in this thesis for a system model shown in Figure 1.1. This is followed by the evaluation of the BER performances of the same algorithms used in a serially concatenated block code scheme. The BER performances of using the proposed SOVA and the conventional SOVA as the component decoders in an iterative decoding of serially concatenated block codes is provided and discussed.

4.2.1 Block Codes

The algorithms are applied to a bit-level trellis and to a sectionalized trellis for the RM (8, 4) code with boundaries at $\{0,4,8\}$. Figure 4.1 shows the BER performances of SOVA applied to a bit-level trellis and to a uniform 2-section trellis for the RM (8,4) code. It is observed that sectionalization of a trellis does not change the error performance of SOVA decoding. The BER performances of MAP, Max-Log-MAP, SOVA, and Viterbi algorithms based on a uniform 2-section trellis for the RM (8,4) code are shown in Figure 4.2. Figure 4.3 also includes the BER performances of the algorithms based on a bit-level trellis for the same code.

From Figure 4.2, it is observed that the performance of SOVA is identical to that of Viterbi algorithm. The difference between Max-Log-MAP and SOVA algorithms is visible at BER above

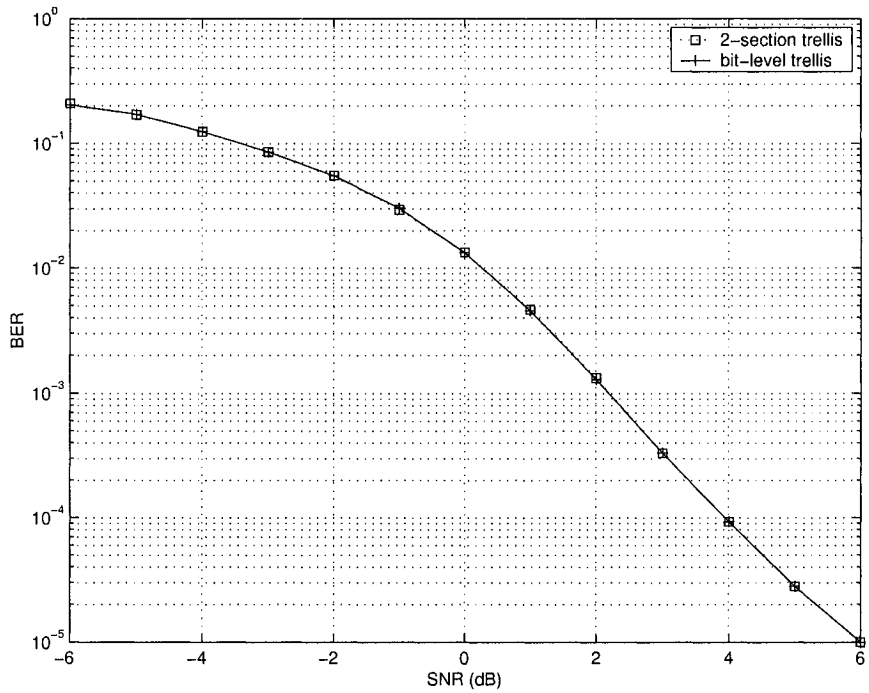


Fig. 4.1 Bit-error performance of SOVA decoding of the RM (8, 4) code.

10^{-2}, at which Max-Log-MAP outperforms SOVA by 0.1 dB. MAP algorithm remains optimum in terms of BER performance. The difference in performance between MAP and ML decoders is more noticeable at high BER, such as above 10^{-3}. The difference between MAP and SOVA is the greatest at BER above 10^{-1} reaching a difference of 1 dB. At BER of 10^{-1}, MAP outperforms SOVA by 0.3 dB, and at BER of 10^{-2} by 0.2 dB. At BER below 10^{-3}, SOVA gives an error performance very close to that of the MAP algorithm. Figure 4.3 reveals that trellis sectionalization does not degrade the error performance of any algorithm examined here.

The BER performances of SOVA, MAP, Max-Log-MAP, and Viterbi algorithms applied to a bit-level trellis and to a sectionalized trellis for RM (32,16), and RM (32,26) codes are provided in the Appendix B.

4.2.2 Serially Concatenated Block Codes

The system model of serially concatenated block codes scheme implemented in this thesis is shown in Figure 4.4.

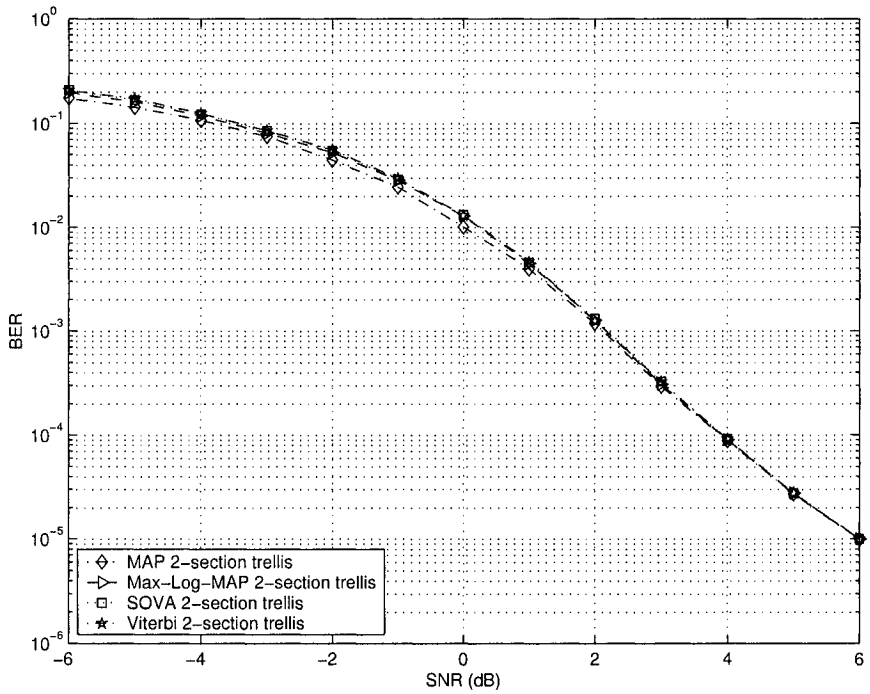


Fig. 4.2 Bit-error performance of MAP, Max-Log-MAP, SOVA, and Viterbi decoding based on a uniform 2-section trellis of the RM (8, 4) code.

The outer encoder encodes an information sequence $\mathbf{m}$ into a code sequence $\mathbf{c1}$ and passes it to the inner encoder. The BPSK modulator maps the encoded sequence $\mathbf{c2}$ from the inner encoder into a bipolar sequence $\mathbf{u}$. This sequence is passed through the AWGN channel and is distorted by noise, $\mathbf{n}$. At the receiving end, the demodulator passes the received signal sequence $\mathbf{r}$ to the inner decoder that outputs the LLR's of the outer code symbols, $S(\mathbf{c1})$. These are passed to the outer decoder to obtain an estimate of the information sequence $\hat{\mathbf{m}}$.

The BER performances of using SOVA, Max-Log-MAP, and MAP as the inner decoders and Viterbi as the outer decoder, obtained from simulations under AWGN channel environment with BPSK modulation, are analyzed next. The outer and inner codes are both formed from the RM (8,4) code. For comparisons, the algorithms are applied to a bit-level trellis and to a sectionalized trellis with boundaries at $\{0,4,8\}$.

Figure 4.5 shows the BER performances of using SOVA as the inner decoder and Viterbi as the outer decoder, both based on a bit-level trellis and on a uniform 2-section trellis.

It is observed from Figure 4.5 that the BER performance curves are identical. As the BER measurement at the output of the outer decoder measures the quality of the reliability estimates

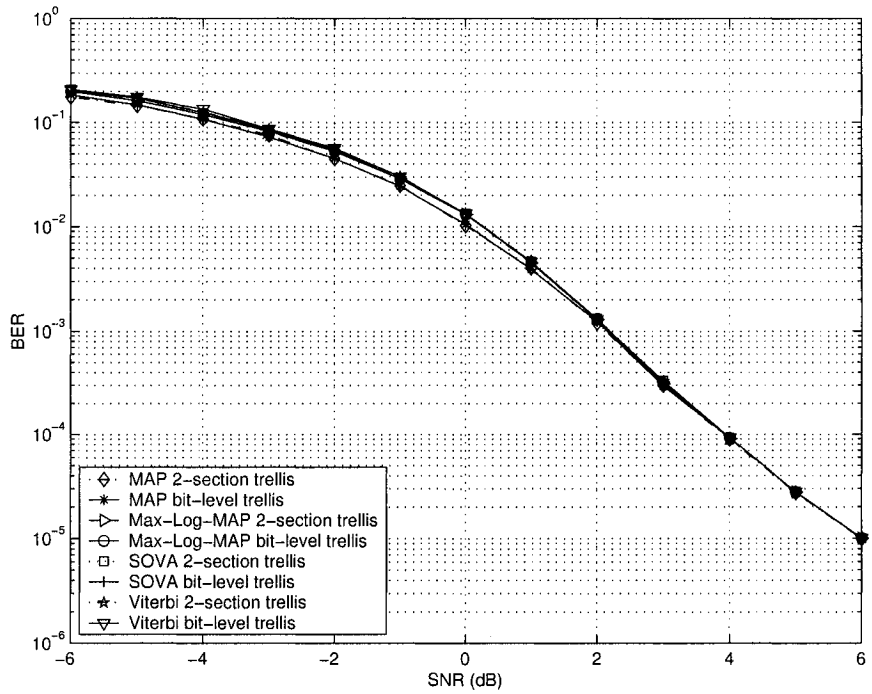


Fig. 4.3 Bit-error performance of MAP, Max-Log-MAP, SOVA, and Viterbi decoding of the RM (8, 4) code.

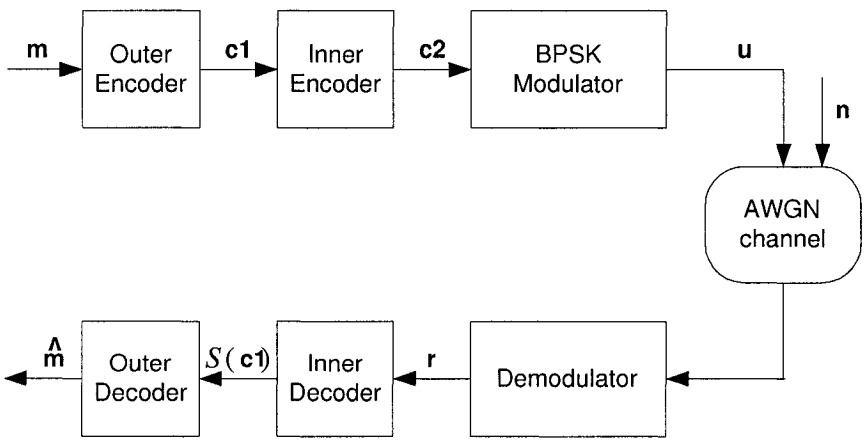


Fig. 4.4 System Model for a Serially Concatenated Block Code Scheme.

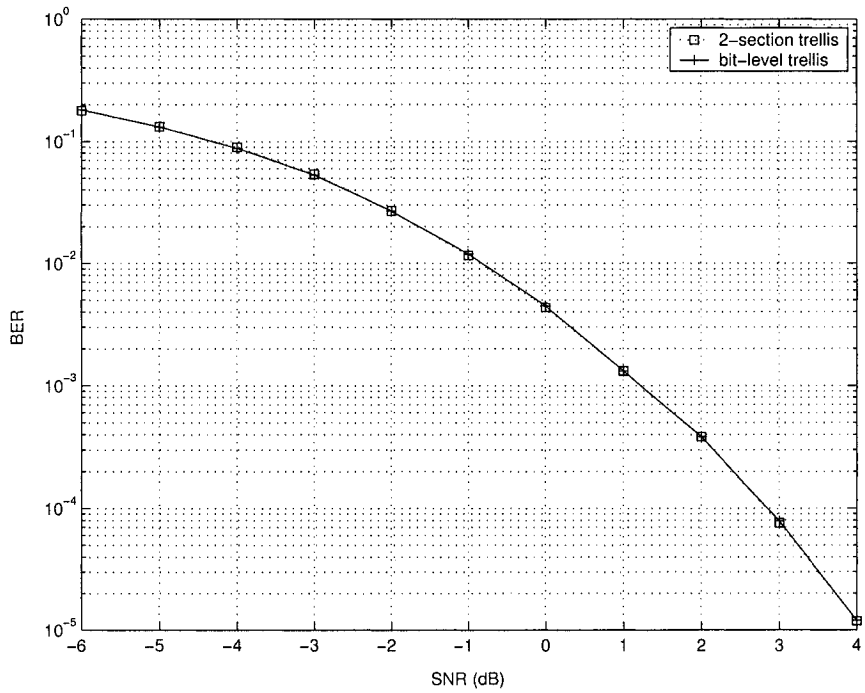


Fig. 4.5 Bit-error performance of using SOVA as the inner decoder in a concatenated scheme formed from the RM (8, 4) code.

of the inner decoder, this observation implies that there is no difference in the quality of the soft outputs of the estimated code bits of the SOVA decoder based on a bit-level trellis and on a sectionalized trellis.

Shown in Figure 4.6 are the BER performances of using MAP, Max-Log-MAP, and SOVA as the inner decoders and Viterbi as the outer decoder, all based on a uniform 2-section trellis for the RM (8,4) code. In Figure 4.7, the BER performances of the decoders based on a bit-level trellis are also included.

Figure 4.6 shows that an improvement in BER performance of using Max-Log-MAP instead of SOVA as the inner decoder, is visible only at BER above 10^{-1}, at which the concatenated scheme with Max-Log-MAP inner decoder outperforms that of SOVA by 0.1 dB. This reveals that the soft outputs of SOVA and Max-Log-MAP algorithms are very similar. It is observed that using MAP as the inner decoder is optimum in terms of BER performance. The difference in performance when MAP is used instead of SOVA as the inner decoder is the greatest at BER above 10^{-1} reaching a difference of 0.5 dB. At BER of 10^{-1}, the concatenated scheme with MAP as the inner decoder outperforms that of SOVA by 0.3 dB, and by 0.2 dB at BER of 10^{-2}. At

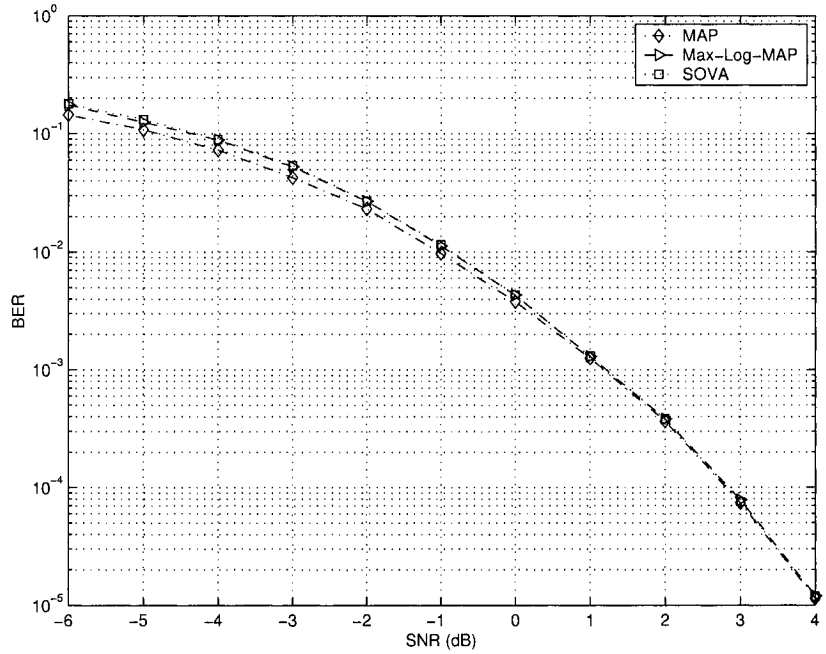


Fig. 4.6 Bit-error performance of using MAP, Max-Log-MAP, and SOVA as the inner decoders in a concatenated scheme based on a uniform 2-section trellis for the RM (8, 4) code.

BER below 10^{-3}, SOVA gives an error performance very close to that of the MAP algorithm. At SNR above 1 dB, the same BER performance is obtained using either as the inner decoder.

From Figure 4.7, it is observed that the same BER performances are obtained using these decoders as the inner decoders in the concatenated scheme based on a bit-level trellis. As these BER curves measure the quality of the reliability estimates of the inner decoder, the fact that trellis sectionalization does not degrade the performance of any decoder examined here in a concatenated scheme indicates that the quality of the soft outputs of the inner decoders based on a bit-level trellis and on a sectionalized trellis is the same.

4.2.3 Iterative Decoding of Serially Concatenated Block Codes

This section presents the performance of iterative SOVA decoding of serially concatenated block codes. The system model implemented in this thesis is shown in Figure 4.8.

An information sequence, $\hat{\mathbf{m}}$, of pk bits is sent to the outer (N_1, k_1) encoder in k_1-bit blocks that generates an outer code sequence $\mathbf{c1}$ of pN_1 bits, where p is an integer. The interleaver, used

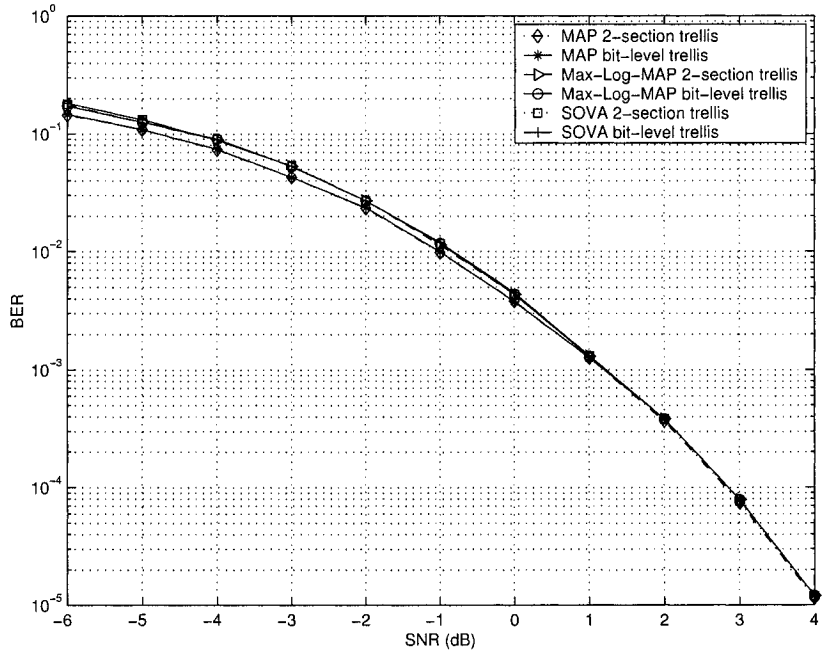


Fig. 4.7 Bit-error performance of using MAP, Max-Log-MAP, and SOVA as the inner decoders in a concatenated scheme based on a bit-level trellis and on a uniform 2-section trellis for the RM (8, 4) code.

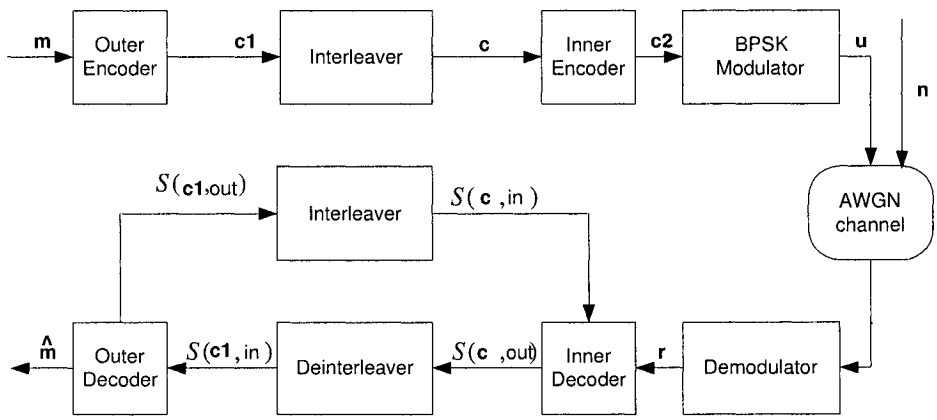


Fig. 4.8 System Model for Iterative Decoding of Serially Concatenated Block Codes.

to separate bursts of errors produced by the inner decoder, receives this sequence, and outputs a sequence, $\mathbf{c}$ that consists of a different order of the code bits of $\mathbf{c1}$, according to a random permutation performed by it. The inner (N_2, k_2) encoder receives $\mathbf{c}$ in k_2-bit blocks and encodes each block, generating an inner code sequence $\mathbf{c2}$. In the receiver, the demodulator passes the received sequence $\mathbf{r}$ to the inner decoder that outputs the LLR's of the inner information symbols, $S(\mathbf{c}, out)$. These are passed through the deinterleaver, that uses the inverse mapping of the interleaver, whose outputs correspond to the LLR's of the outer code symbols, $S(\mathbf{c1}, in)$. The outer decoder processes these and outputs the LLR's of its code symbols, $S(\mathbf{c1}, out)$ that are passed to the interleaver for another iteration. The symbols in the interleaved sequence, $S(\mathbf{c}, in)$, replace the inner information symbols in the received sequence, that is then processed by the inner decoder. At the final iteration, the LLR's of the information symbols are used to obtain an estimated information sequence, $\hat{\mathbf{m}}$.

Figure 4.9 depicts the BER performance of iterative SOVA decoding of serially concatenated block codes formed from the RM (8, 4) code with interleaver of size 256 bits and with 7 iterations over AWGN channel with BPSK modulation. It is observed that the BER performance curves are identical. This observation implies no difference in the soft outputs of the estimated code bits of the SOVA decoder based on a bit-level trellis and on a sectionalized trellis.

4.3 Summary

For the decoding of a bit-level trellis, the computational complexity of SOVA exceeds that of Viterbi, but is lower than that of Max-Log-MAP. MAP is the most computationally complicated decoder.

Trellis sectionalization of all linear block codes considered, significantly reduces the computational complexity of all algorithms presented in this thesis. Savings in computations of up to 79% are achieved when SOVA is used in place of MAP, and up to 30% when SOVA is used instead of Max-Log-MAP. The complexity of SOVA remains higher than that of Viterbi, by at least 71%. Optimum sectionalization of all the SISO algorithms presented always results in the mirror symmetry of the trellis.

The BER performance evaluation under AWGN channel environment reveals that trellis sectionalization does not degrade the performance of the decoders. For the RM (8, 4) code, the BER performance of SOVA is very similar to that of Viterbi. Max-Log-MAP algorithm outperforms SOVA by only 0.1 dB at BER above 10^{-2}. MAP algorithm is optimum with respect to BER,

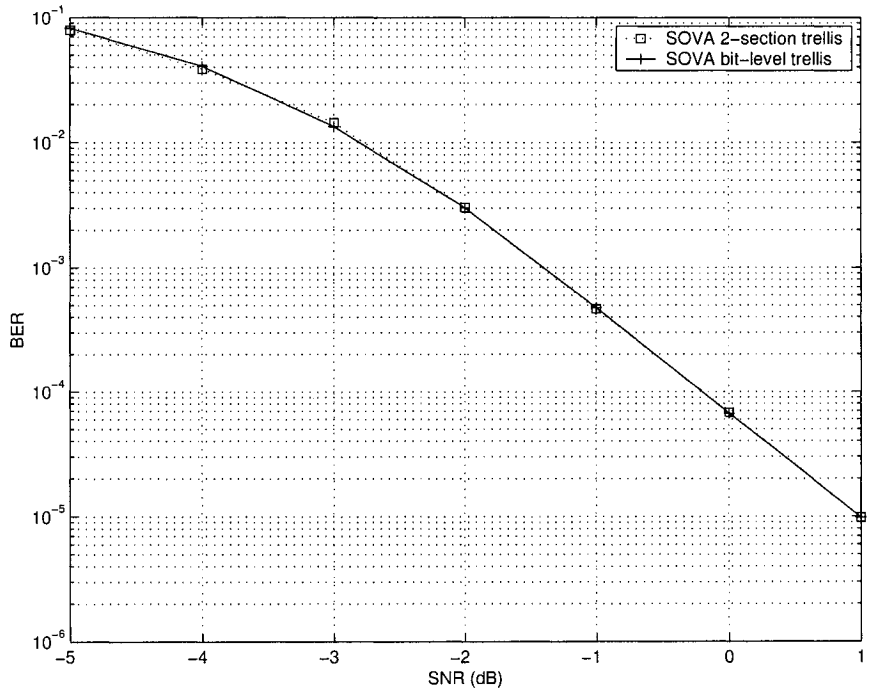


Fig. 4.9 Iterative SOVA Decoding of Serially Concatenated Block Codes.

reaching a difference of 1 dB from SOVA at BER above 10^{-1}.

The BER performance curves obtained for the concatenated scheme based on a bit-level trellis and on a sectionalized one, formed from the RM (8, 4) code are identical for any SISO algorithm, examined in this thesis, used as the inner decoder and Viterbi as the outer decoder. This further indicates that the quality of the soft outputs of the inner decoders based on a bit-level trellis and on a sectionalized trellis is the same. With Viterbi as the outer decoder, using Max-Log-MAP instead of SOVA as the inner decoder, shows a difference of only 0.1 dB at BER above 10^{-1}, indicating that the soft outputs of the two algorithms are very similar. Using MAP as the inner decoder gives the best BER performance.

Iterative decoding of serially concatenated block codes reveals that the quality of reliability estimates of the proposed SOVA decoder is the same as that of the conventional SOVA decoder.

Chapter 5

Conclusion

5.1 Summary

A low-complexity near-optimum soft-input soft-output (SISO) decoding scheme is established based on the soft-output Viterbi algorithm (SOVA) applied to a sectionalized trellis. A minimal bit-level trellis was sectionalized using the section boundaries obtained from the application of the computation-wise optimum sectionalization algorithm for SOVA and other decoders considered, including Viterbi, Max-Log-MAP and MAP.

The advantages of SOVA based on a sectionalized trellis over the conventional bit-level trellis and over other algorithms studied are demonstrated by the analysis of the decoder's required computational complexity and simulation results of its BER performance assuming binary modulation and AWGN channel.

The examination of the optimal sectionalizations obtained for different RM codes, in Section 4.1, reveals:

- The mirror symmetry of a trellis is always obtained for SOVA decoding.
- Contrary to SOVA, for Viterbi decoding, the optimally sectionalized trellis is not always symmetric.
- As for SOVA, mirror symmetry is always attained for MAP and Max-Log-MAP algorithms.
- The required computational complexity of the new SOVA decoding scheme is significantly lower than that of the conventional decoder based on the bit-level trellis.

- In comparison to Viterbi, Max-Log-MAP and MAP algorithms, although maximum-likelihood (ML) Viterbi requires the least amount of computational operations, the proposed SOVA decoding approach is the most computationally efficient SISO algorithm.

A comprehensive analysis of the simulations of the BER performances under AWGN channel environment, in Section 4.2, demonstrates:

- No performance degradation of applying SOVA to a sectionalized trellis with respect to SOVA based on a bit-level trellis.
- The BER performances of Viterbi, Max-Log-MAP and MAP algorithms based on a sectionalized trellis are identical to those obtained for a bit-level trellis.
- The differences in the BER performances of the algorithms are noticeable at low SNR, and become negligible at high SNR.
- The quality of the soft outputs of the SISO decoders based on a bit-level trellis and on a sectionalized trellis are the same.
- The quality of the soft outputs of Max-Log-MAP and SOVA are very similar.
- The optimality of the decoders based on a sectionalized trellis remains the same as for the decoders based on a bit-level trellis. Hence, the new SOVA decoder remains suboptimum.

The results of this thesis justify that the proposed SOVA decoder is sufficient for error control schemes that require a computationally efficient near-optimum SISO decoder.

Some ideas for the potential future work are presented next.

5.2 Future Work

Applications to Block Turbo Codes

In 2001, SOVA based on a bit-level trellis was applied to Block Turbo Codes (BTCs) and proven to be more suitable for hardware implementation than the conventional MAP algorithm used in BTCs, due to its lower complexity [33]. The decoding scheme, studied in this thesis, consisting of SOVA based on a sectionalized trellis for block codes, demonstrated that it requires less computational operations than SOVA based on a bit-level trellis. This would justify the new,

computationally efficient, and practically implementable decoding scheme, to be applied to decoders for block codes in turbo loop, enabling the use of turbo codes in an even wider range of applications.

Storage Requirements

The computational complexity and storage requirement of SOVA decoding depend on the sectionalization of the trellis. A sectionalization that minimizes both, in general, does not exist. In this thesis, it is assumed that there is no severe constraint on the size of memory storage, and the focus is on the sectionalization that minimizes the computational complexity. It would be interesting to investigate how this optimally sectionalized trellis, derived here, affects the storage requirements of the decoder.

Another idea for the potential future work involves deriving a sectionalized trellis for SOVA decoder, that is optimal with respect to the decoder's storage requirements. This topic would determine by how much SOVA decoding based on a sectionalized trellis reduces its storage requirement, how memory efficient it is with comparisons to other trellis based decoders, and if it is adequate for systems with constraints on their memory storage size.

Decoding Time

Considering that optimal sectionalization of linear block codes considered for SOVA always results in a symmetric trellis, it would be interesting to investigate if this structural property can reduce the decoding time.

This mirror symmetry allows bidirectional decoding, the use of two identical circuits for the computation of the forward and backward state probabilities, and hence, simplifying the integrated circuit (IC) implementation. It was applied to MAP and Max-Log-MAP algorithms in [4]. This technique permutes the encoded sequence $\{c_1, c_2, \ldots, c_N\}$ to $\{c_1, c_N, c_2, c_{N-1}, \ldots, c_{N/2}\}$ before transmission. The corresponding received sequence $\{r_1, r_N, r_2, r_{N-1}, \ldots, r_{N/2}\}$ is then permuted into $\mathbf{r1} = \{r_1, r_2, \ldots, r_N\}$ and $\mathbf{r2} = \{r_N, r_{N-1}, \ldots, r_1\}$ and shifted from both ends of the sectionalized trellis. Because of the mirror symmetry, the forward and backward recursions are performed simultaneously and the branch probabilities are computed for the following order of sections $\{T_{h_1}, T_{h_v}, T_{h_2}, T_{h_{v-1}}, \ldots T_{h_{v/2}}\}$. The computation of the soft outputs begins when the two recursions meet in the middle of the trellis. The results in [4] showed that this approach reduces the decoding time by half.

Appendix A

Comparisons with Results in the Literature

The best results of computational complexities for Viterbi, Max-Log-MAP, and MAP algorithms, found in the literature are provided in Table A.1. All the values found in the literature that are not the same as the ones obtained in this thesis are in boldface.

A.1 Bit-Level Trellis Comparisons

The same results were obtained in this thesis as in the referenced literature for Viterbi decoding based on a bit-level trellis for all linear block codes considered, as it is shown in Table A.1. The satisfying results of this thesis for MAP and Max-Log-MAP decoders based on a bit-level trellis were obtained by applying the methods for efficient computation, stated in Section 2.2.

For the MAP decoder, taking into account that the forward and backward path metrics of the initial and final state, respectively, are set to one reduces the required number of multiplications. In the forward recursion, two multiplications are saved, one for each branch, in the first section, and also two multiplications are saved for the last section in the backward recursion. In the computation of the LLR of the first and of the last bit, two multiplications are saved for each computation. With the weight factor of five for a multiplication operation, the number of required multiplications is $5 \cdot 8 = 40$ less than that in the referenced literature, as it is shown in Table A.1.

For the same reasons as for the MAP decoder, the number of computations is lower for the Max-Log-MAP decoder in this thesis than in the referenced literature. Analysis of the bit-level trellis decoding in this thesis, in comparison to [4], reveals that four less additions are required for the forward and backward recursions, one for each branch in the first and last sections. This

Table A.1 Computational complexities of Viterbi, Max-Log-MAP, and MAP decoding of RM codes found in the literature.

(N, K) RM Codes	Obtained Results	Best Known	Reference
(8,4)			
Viterbi			[3], [40], [41]
Bit-level	53	53	
Normalized Complexity	13.3	13.3	
Optimum Boundaries	{0,4,8}	{0,4,8}	
Complexity	23	23	
Normalized Complexity	5.8	5.8	
MAP			[4]
Bit-level	970	**1,010**	
Normalized Complexity	242.5	**252.5**	
Optimum Boundaries	{0,8}	{0,8}	
Complexity	257	**370**	
Normalized Complexity	64.3	**92.5**	
Max-Log-MAP			[4]
Bit-level	226	**230**	
Normalized Complexity	56.5	**57.5**	
Optimum Boundaries	{0,4,8}	{0,4,8}	
Complexity	108	**156**	
Normalized Complexity	27	**39**	
(16,5)			
Viterbi			[3]
Bit-level	193	193	
Normalized Complexity	38.6	38.6	
Optimum Boundaries	{0,4,8,12,15,16}	{0,4,8,12,15,16}	
Complexity	94	94	
Normalized Complexity	18.8	18.8	
MAP			[4]
Bit-level	3,746	**3,786**	
Normalized Complexity	749.2	**757.2**	
Optimum Boundaries	{0,16}	{0,16}	
Complexity	897	**1,235**	
Normalized Complexity	179.4	**247**	
Max-Log-MAP			[4]
Bit-level	882	**886**	
Normalized Complexity	176.4	**177.2**	
Optimum Boundaries	{0,4,8,12,16}	{0,**2**,4,8,12,**14**,16}	
Complexity	414	**486**	
Normalized Complexity	82.8	**97.2**	

Table A.1 Computational complexities of Viterbi, Max-Log-MAP, and MAP decoding of RM codes found in the literature

(N,K) RM Codes	Obtained Results	Best Known	Reference
(16,11)			
Viterbi			[3]
Bit-level	353	353	
Normalized Complexity	32.1	32.1	
Optimum Boundaries	{0,4,8,12,16}	{0,4,8,12,16}	
Complexity	167	167	
Normalized Complexity	15.2	15.2	
MAP			[4]
Bit-level	5,586	**5,625**	
Normalized Complexity	507.8	**511.4**	
Optimum Boundaries	{0,8,16}	{0,**4**,**6**,8,**10**,**12**,16}	
Complexity	3,092	**3,980**	
Normalized Complexity	281.1	**361.8**	
Max-Log-MAP			[4]
Bit-level	1,442	**1,446**	
Normalized Complexity	131.1	**131.5**	
Optimum Boundaries	{0,4,6,8,10,12,16}	{0,**2**,4,6,8,10,12,**14**,16}	
Complexity	1,136	**1,222**	
Normalized Complexity	103.3	**111.1**	
(32,6)			
Viterbi			[3]
Bit-level	729	729	
Normalized Complexity	121.5	121.5	
Optimum Boundaries	{0,4,8,12,16,20,24,28,31,32}	{0,4,8,12,16,20,24,28,31,32}	
Complexity	278	278	
Normalized Complexity	46.3	46.3	
MAP			[4]
Bit-level	14,674	**14,714**	
Normalized Complexity	2,445.7	**2,452.3**	
Optimum Boundaries	{0,8,16,24,32}	{0,8,16,24,32}	
Complexity	2,942	**3,485**	
Normalized Complexity	490.3	**580.8**	
Max-Log-MAP			[4]
Bit-level	3,474	**3,478**	
Normalized Complexity	579	**579.7**	
Optimum Boundaries	{0,4,8,16,24,28,32}	{0,**2**,4,8,16,24,28,**30**,32}	
Complexity	1,326	**1,510**	
Normalized Complexity	221	**251.7**	

Table A.1 Computational complexities of Viterbi, Max-Log-MAP, and MAP decoding of RM codes found in the literature.

(N,K) RM Codes	Obtained Results	Best Known	Reference
(32,16)			
Viterbi			[3]
Bit-level	7,993	7,993	
Normalized Complexity	499.6	499.6	
Optimum Boundaries	{0,8,16,24,32}	{0,8,16,24,32}	
Complexity	2,383	2,383	
Normalized Complexity	148.9	148.9	
MAP			[4]
Bit-level	137,730	**137,770**	
Normalized Complexity	8,608.1	**8,610.6**	
Optimum Boundaries	{0,4,8,12,16,20,24,28,32}	{0,**3**,8,12,16,20,24,**29**,32}	
Complexity	59,278	**59,850**	
Normalized Complexity	3,704.9	**3,740.6**	
Max-Log-MAP			[4]
Bit-level	35,138	**35,142**	
Normalized Complexity	2,196.1	**2,196.4**	
Optimum Boundaries	{0,3,5,8,12,16,20,24,27,29,32}	{0,**1**,3,5,8,12,16,20,24,27,29,**31**,32}	
Complexity	21,870	**22,078**	
Normalized Complexity	1,366.9	**1,379.9**	
(32,26)			
Viterbi			[3]
Bit-level	1,721	1,721	
Normalized Complexity	66.2	66.2	
Optimum Boundaries	{0,4,8,12,16,20,24,28,32}	{0,4,8,12,16,20,24,28,32}	
Complexity	1,255	1,255	
Normalized Complexity	48.3	48.3	
MAP			[4]
Bit-level	26,082	**26,120**	
Normalized Complexity	1,003.2	**1,004.6**	
Optimum Boundaries	{0,7,8,10,11,12,13,14,16, 18,19,20,21,22,24,25,32}	{0,**4**,**6**,7,8,10,11,12,13,14,16, 18,19,20,21,22,24,25,**26**,**28**,32}	
Complexity	22,154	**22,660**	
Normalized Complexity	852.1	**871.5**	
Max-Log-MAP			[4]
Bit-level	6,946	**6,950**	
Normalized Complexity	267.2	**267.3**	
Optimum Boundaries	{0,4,6,7,8,10,11,12,13,14,16, 18,19,20,21,22,24,25,26,28,32}	{0,**2**,4,6,7,8,10,11,12,13,14,16, 18,19,20,21,22,24,25,26,28,**30**,32}	
Complexity	6,356	**6,446**	
Normalized Complexity	244.5	**247.9**	

Table A.1 Computational complexities of Viterbi, Max-Log-MAP, and MAP decoding of RM codes found in the literature.

(N, K) RM Codes	Obtained Results	Best Known	Reference
(64,7)			
Viterbi			[7], [3]
Bit-level	2,825	2,825	
Normalized Complexity	403.6	403.6	
Optimum Boundaries	{0,4,8,12,16,24,32, 40,48,52,56,60,63,64}	{0,4,8,12,16,24,32, 40,48,52,56,60,63,64}	
Complexity	806	806	
Normalized Complexity	115.1	115.1	
MAP			[4]
Bit-level	58,034	**58,074**	
Normalized Complexity	8,290.6	**8,296.3**	
Optimum Boundaries	{0,16,32,48,64}	{0,**8**,16,32,48,**56**,64}	
Complexity	7,934	**9,405**	
Normalized Complexity	1,133.4	**1,343.6**	
Max-Log-MAP			[4]
Bit-level	13,778	**13,782**	
Normalized Complexity	1,968.3	**1,968.9**	
Optimum Boundaries	{0,4,8,16,32, 48,56,60,64}	{0,**2**,4,8,16,24,32, 40,48,56,60,**62**,64}	
Complexity	4,430	**5,094**	
Normalized Complexity	632.9	**727.7**	

Table A.1 Computational complexities of Viterbi, Max-Log-MAP, and MAP decoding of RM codes found in the literature.

(N, K) RM Codes	Obtained Results	Best Known	Reference
(64,22)			
Viterbi			[3]
Bit-level	425,209	425,209	
Normalized Complexity	19,327.7	19,327.7	
Optimum Boundaries	{0,8,16,32,48,56,61,63,64}	{0,8,16,32,48,56,61,63,64}	
Complexity	104,370	**101,786**	
Normalized Complexity	4,744.1	**4,626.6**	
MAP			[4]
Bit-level	7,976,575	**7,976,615**	
Normalized Complexity	362,571.8	**362,573.4**	
Optimum Boundaries	{0,8,16,24,32,40,48,56,64}	{0,**3**,8,16,24,32,40,48,56,**61**,64}	
Complexity	2,055,840	**2,062,990**	
Normalized Complexity	93,447.3	**93,772.3**	
Max-Log-MAP			[4]
Bit-level	1,975,458	**1,975,462**	
Normalized Complexity	89,793.6	**89,793.7**	
Optimum Boundaries	{0,3,5,8,10,16,18,24,30,32, 34,40,46,48,54,56,59,61,64}	{0,**1**,3,5,8,10,16,18,24,30,32, 34,40,46,48,54,56,59,61,**63**,64}	
Complexity	903,562	**905,974**	
Normalized Complexity	41,071	**41,180.6**	

Table A.1 Computational complexities of Viterbi, Max-Log-MAP, and MAP decoding of RM codes found in the literature.

(N, K) RM Codes	Obtained Results	Best Known	Reference
(64,42)			
Viterbi			[3]
Bit-level	773,881	773,881	
Normalized Complexity	18,425.7	18,425.7	
Optimum Boundaries	{0,8,16,24,32,40,48,56,64}	{0,8,16,24,32,40,48,56,64}	
Complexity	538,799	538,799	
Normalized Complexity	12,828.5	12,828.5	
MAP			[4]
Bit-level	11,986,300	**11,986,345**	
Normalized Complexity	285,388.1	**285,389.2**	
Optimum Boundaries	{0,4,8,12,14,16, 20,22,24,26,28,32, 36,38,40,42,44,48, 50,52,56,60,64}	{0,**1**,4,8,12,14,16, 20,22,24,26,28,32, 36,38,40,42,44,48, 50,52,56,60,**63**,64}	
Complexity	8,261,180	**8,261,890**	
Normalized Complexity	196,694.8	**196,711.7**	
Max-Log-MAP			[4]
Bit-level	3,195,810	**3,195,814**	
Normalized Complexity	76,090.7	**76,090.8**	
Optimum Boundaries	{0,4,6,8,10,12,14,16, 18,20,22,24,26,28,30,32, 34,36,38,40,42,44,46,48, 50,52,54,56,58,60,64}	{0,**2**,4,6,8,10,12,14,16, 18,20,22,24,26,28,30,32, 34,36,38,40,42,44,46,48, 50,52,54,56,58,60,**62**,64}	
Complexity	2,646,470	**2,646,566**	
Normalized Complexity	63,011.2	**63,013.5**	

Table A.1 Computational complexities of Viterbi, Max-Log-MAP, and MAP decoding of RM codes found in the literature.

(N, K) RM Codes	Obtained Results	Best Known	Reference
(64,57)			
Viterbi			[3]
Bit-level	7,529	7,529	
Normalized Complexity	132.1	132.1	
Optimum Boundaries	{0,4,8,12,13,14,15,16, 20,21,22,23,24,25,26, 27,28,32,36,37,38,39, 40,41,42,43,44,48,49, 50,51,52,56,60,64}	{0,4,8,12,13,14,15,16, 20,21,22,23,24,25,26, 27,28,32,36,37,38,39, 40,41,42,43,44,48,49, 50,51,52,56,60,64}	
Complexity	6,507	6,507	
Normalized Complexity	114.2	114.2	
MAP			[4]
Bit-level	112,130	**112,170**	
Normalized Complexity	1,967.2	**1,967.9**	
Optimum Boundaries	{0,7,8, 10,11,12,13,14,15,16, 18,19,20,21,22,23,24, 25,26,27,28,29,30,32, 34,35,36,37,38,39,40, 41,42,43,44,45,46,48, 49,50,51,52,53,54,56, 57,64}	{0,**4**,**6**,7,8, 10,11,12,13,14,15,16, 18,19,20,21,22,23,24, 25,26,27,28,29,30,32, 34,35,36,37,38,39,40, 41,42,43,44,45,46,48, 49,50,51,52,53,54,56, 57,**58**,**60**,64}	
Complexity	104,558	**105,065**	
Normalized Complexity	1,834.4	**1,843.2**	
Max-Log-MAP			[4]
Bit-level	30,242	**30,246**	
Normalized Complexity	530.6	530.6	
Optimum Boundaries	{0,4,6,7,8, 10,11,12,13,14,15,16, 18,19,20,21,22,23,24, 25,26,27,28,29,30,32, 34,35,36,37,38,39,40, 41,42,43,44,45,46,48, 49,50,51,52,53,54,56, 57,58,60,64}	{0,**2**,4,6,7,8, 10,11,12,13,14,15,16, 18,19,20,21,22,23,24, 25,26,27,28,29,30,32, 34,35,36,37,38,39,40, 41,42,43,44,45,46,48, 49,50,51,52,53,54,56, 57,58,60,**62**,64}	
Complexity	29,080	**29,174**	
Normalized Complexity	510.2	**511.8**	

is reflected in Table A.1.

A.2 Optimally Sectionalized Trellis Comparisons

For Viterbi decoding of a sectionalized trellis, the optimum section boundaries are the same for all linear block codes considered, and the computational complexities are the same for all, except for RM (64,22) code. The optimum sectionalization of this code results in {0,8,16,32,48,56,61,63,64} section boundaries. However, the obtained computational complexity in this thesis is 104,370, whereas it is 101,786 in [3]. The difference in the results is due to the different methods used for computing the branch probabilities in the third and fourth sections, in which the size of the composite branch is two. In this thesis, the expression $|B_j^d| \cdot |B_j^p| \cdot (\ell_j - 1)/2$ was applied, considering that it provides the minimum number of additions. However, in [3], that expression was divided by 4, resulting in 2,584 less computations for the two sections, as is shown in Table A.1.

Significantly higher computational complexities for Max-Log-MAP and MAP decoding of all RM codes considered, and different optimum section boundaries of most RM codes, were attained in the referenced literature, as the methods for efficient computation of the decoding steps for a sectionalized trellis, stated in Section 3.2.1, were not applied.

For MAP decoding, the number of required multiplications is reduced by taking into account that the forward and backward probabilities of the initial and final states, respectively, are set to one. The number of required additions for computing the branch probabilities is reduced by taking into account the following: the possible encoder outputs in each section are complementary for RM codes, and therefore, only half of the branch probabilities need to be computed; if the code bits have even weight and the section length is even, the number of additions required could be saved even more; depending on the length of a section, gray code ordering of the code bits could also reduce the number of required additions. It is apparent that the number of multiplication and addition operations required depends on the number of parallel, distinct, and composite branches, the length of each section, and whether the branch labels have even weight. Because these values differ for each section, it is possible to analyze the difference in the computational complexities between the results in this thesis and in the referenced literature only for linear block codes that resulted with the same optimum section boundaries. These codes are: RM (8,4), RM (16,5), and RM (32,6).

For RM (8,4) code, optimum section boundaries obtained in this thesis and in the referenced literature are $\{0,8\}$. The one section trellis has 16 parallel branches. For the computation of the branch probabilities, considering that only half of the branch probabilities need to be computed, the expression $(\ell_j - 1) \cdot |B_j^d| \cdot |B_j^p|/2$ was applied resulting in 56 additions. However, in [4], $(\ell_j - 1) \cdot |B_j^d| \cdot |B_j^p|$ expression was used, resulting in 56 more additions. The computations of the forward and backward state probabilities, in this thesis, required no operations, because of the initialization of the state probabilities. However in [4], 1 multiplication was required in each recursion, resulting in $5 \cdot 1 \cdot 2 = 10$ more operations. In the computation of the soft output, considering that there is only one section, no operations are necessary to compute (3.16) or the numerator of the LLR. Obtaining the denominator of the LLR, requires 8 subtractions, 1 for each bit. Final evaluation of the LLR requires 8 divisions, 1 for each bit. However in [4] additional computations resulted from the evaluation of (3.16), requiring 2 multiplications, and from the evaluation of the numerator of the LLR, requiring 8 multiplications. Therefore, computation of the soft output in [4], requires $5 \cdot 10 = 50$ additional operations. In total, the complexity of the MAP decoder for RM (8,4) code is by $56 + 10 + 50 = 116$ operations higher than that of this thesis, as it is shown in Table A.1 with 0.8% error.

The optimum sectionalization of RM (16,5) code, resulted in the $\{0,16\}$ section boundaries in this thesis and in the referenced literature. This one section trellis has 32 parallel branches. The same procedure was followed to compute the complexity for this code, as for RM (8,4) code analyzed above. Therefore, the computation of the branch probabilities, results in $(16 - 1) \cdot 32/2 = 240$ additions, whereas in [4], 240 more additions are required. No operations are required for the computation of the forward and backward state probabilities, whereas in [4], one multiplication is required for each, resulting in $5 \cdot 2 = 10$ operations more. The computation of the soft output, requires 16 subtractions for obtaining the denominator of the LLR, and 16 divisions for the evaluation of the LLR. However in [4], also 2 multiplications are required for the evaluation of (3.16), and also 16 multiplications are required for the evaluation of the LLR numerator, resulting in the total of $5 \cdot 18 = 90$ additional operations. Hence, the complexity of the MAP decoder being 897 in this thesis, should be $897 + (240 + 10 + 90) = 1237$ in [4], as it is shown in Table A.1, with the percentage of error of $2/1237 = 0.16\%$.

The optimum sectionalization of RM (32,6) code, in this thesis and in [4], resulted in $\{0,8,16,24,32\}$ section boundaries. With the examination of the TOGM for this code, it is observed that in each of the four sections there are 16 distinct composite branches, 16 composite branches in the first and last sections and 32 in the second and third sections, and the size of each

is one. The same procedure was followed to compute the branch probabilities, the forward and the backward state probabilities, as for the RM codes analyzed above. Therefore, $(8-1)\cdot 16/2 = 56$ additions are needed in each section for the computation of the branch probabilities, and no operations are required for the computation of the forward and backward state probabilities in the first and last sections, respectively. However in [4], $56 \cdot 4 = 224$ more additions are needed for the branch probabilities, and 16 more multiplications are needed in each recursion, resulting in $5 \cdot 16 \cdot 2 = 160$ additional operations. In the computation of the soft output for the first and last sections, using expressions from the Table 3.1, $16 + 8 = 24$ multiplications, and in the rest of the sections, $2 \cdot 32 + 8 = 72$ multiplications are required. In [4], as the first and last sections are treated the same way as the rest, $2 \cdot 16 + 8 = 40$ multiplications are required. The difference of $40 - 24 = 16$ multiplications for each section, results in total of $16 \cdot 2 \cdot 5 = 160$ additional operations in [4]. Therefore, in total, the complexity of this thesis is $224 + 160 + 160 = 544$ operations lower, as it is shown in Table A.1 with 0.03% error.

For Max-Log-MAP, the number of addition and comparison operations obtained in this thesis is less than the reported results of the referenced literature for the same reasons as for the MAP decoder, in addition to the method for computing the composite branch probabilities by which the number of comparisons is lowered if the parallel branches within each composite branch are complementary.

For RM (8,4) code, optimum section boundaries obtained in this thesis and in the referenced literature are $\{0,4,8\}$. With the examination of the TOGM for this code, it is observed that in each section there are 4 distinct composite branches, 4 composite branches, and the size of each is two. For the computation of the branch probabilities, considering that the section lengths are even and that the weights of the generated codewords for each section is even, using the applicable expression from Table 3.1, $2^{4/2}+4-4+2^{4-2} = 8$ addition operations are needed for each section. However, in [4], $|B_j^d| \cdot |B_j^p| \cdot (\ell_j - 1)$ expression was used, resulting in 16 additions more for each section. For the computation of the composite branch probabilities, taking into account that the parallel branches are self complementary in each section, from Table 3.1, $4 \cdot (2/2 - 1) = 0$ additions are needed. However, in [4], that was not considered, and $|B_j^d| \cdot (|B_j^p| - 1)$ expression was applied instead, resulting in 4 more additions for each section. For the forward and backward recursions, no additions are needed for the first and last sections, respectively, unlike in [4], resulting in the total of 8 more additions. In total, there are $16 \cdot 2 + 4 \cdot 2 + 8 = 48$ more operations required for Max-Log-MAP decoding in [4] than in this thesis. This is illustrated in Table A.1.

Appendix B

BER Performance

The BER performances of SOVA, MAP, Max-Log-MAP, and Viterbi algorithms applied to a bit-level trellis and to a sectionalized trellis for RM (32,16) and RM (32,26) codes are examined in this section. For a sectionalized trellis for RM (32,16) code, the section boundaries used are $\{0,8,16,24,32\}$, and for RM (32,26) code, the section boundaries used are $\{0,4,8,12,16,20,24,28,32\}$.

Figures B.1 and B.2 show BER performances of SOVA applied to a bit-level trellis and to a uniformly sectionalized trellis for RM (32,16) and RM (32,26) codes, respectively. From both figures, it is observed that the performance of SOVA applied to a sectionalized trellis does not change from its performance based on a bit-level trellis.

Figures B.3 and B.4 depict bit-error performances of MAP, Max-Log-MAP, SOVA, and Viterbi algorithms based on a bit-level trellis and on a sectionalized trellis for the (32, 16) and (32, 26) RM codes. Both figures reveal that sectionalization does not degrade the performance of any algorithm. Viterbi gives an error performance very close to that of the SOVA algorithm. Max-Log-MAP algorithm yields better performance than SOVA at BER above 10^{-1}, reaching a difference of 0.1 dB. At BER below 10^{-1}, the two algorithms become identical. It is observed from both figures that MAP algorithm is optimum in terms of BER. The difference between MAP and SOVA is the greatest at BER above 10^{-1} reaching a difference of 1 dB for RM (32,26) code, and of 0.5 dB for RM (32,16). From Figure B.3, MAP outperforms SOVA by 0.1 dB at BER of 10^{-2}. At BER below 10^{-3}, the two algorithms perform similarly. Similarly, from Figure B.4, MAP outperforms SOVA by 0.20 dB at BER of 10^{-1}, and by 0.1 dB at BER of 10^{-2}. At BER below 10^{-2}, the difference in the performance curves of the two algorithms is negligible.

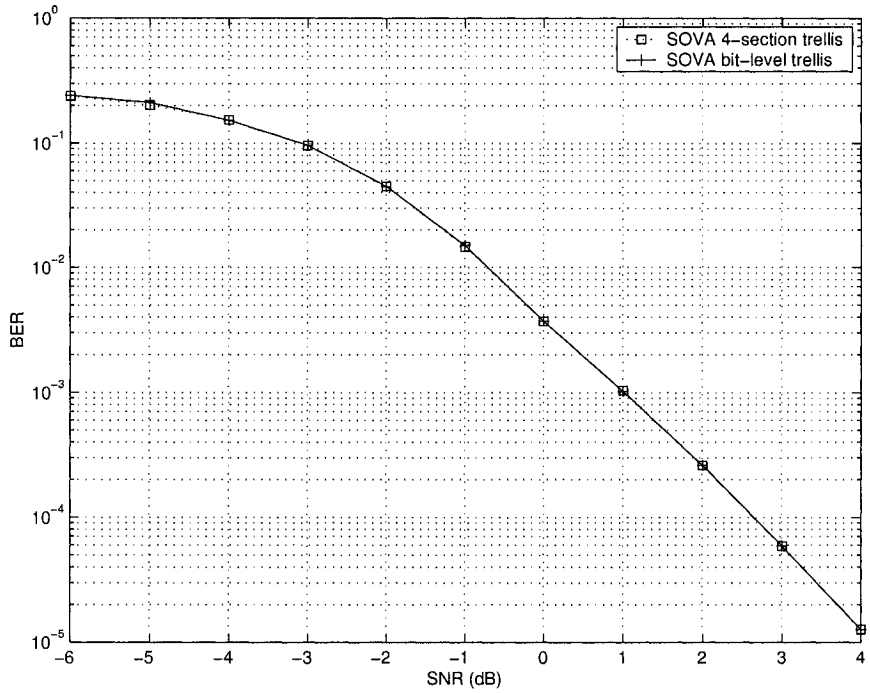


Fig. B.1 Bit-error performance of SOVA decoding of the (32, 16) RM code.

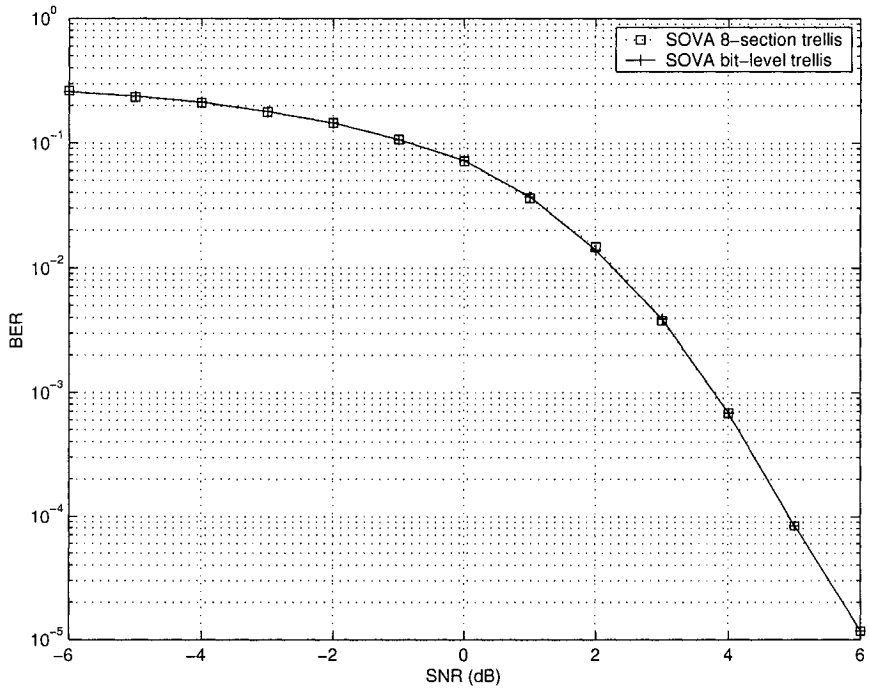


Fig. B.2 Bit-error performance of SOVA decoding of the (32, 26) RM code.

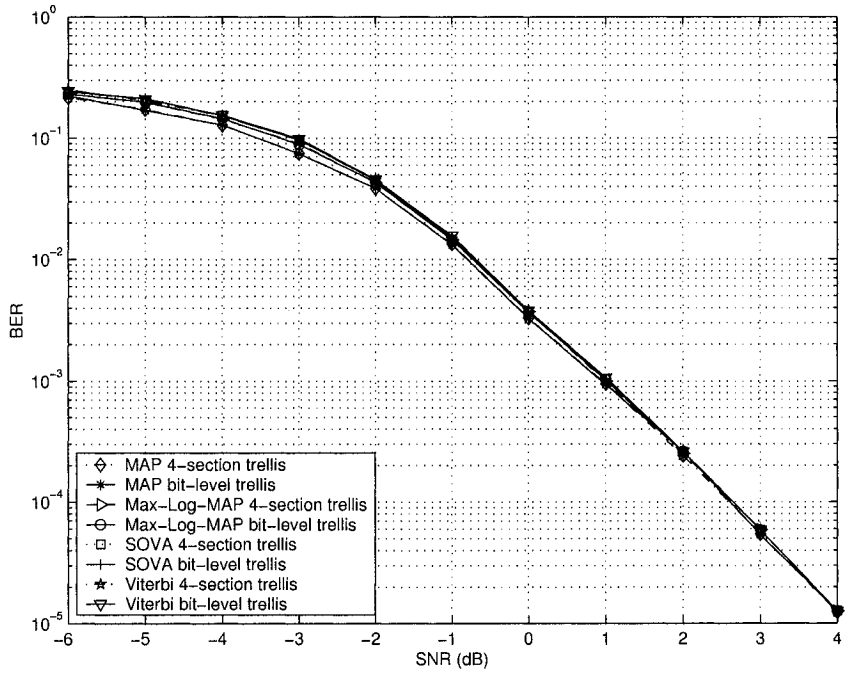


Fig. B.3 Bit-error performance of MAP, Max-Log-MAP, SOVA, and Viterbi decoding of the (32, 16) RM code.

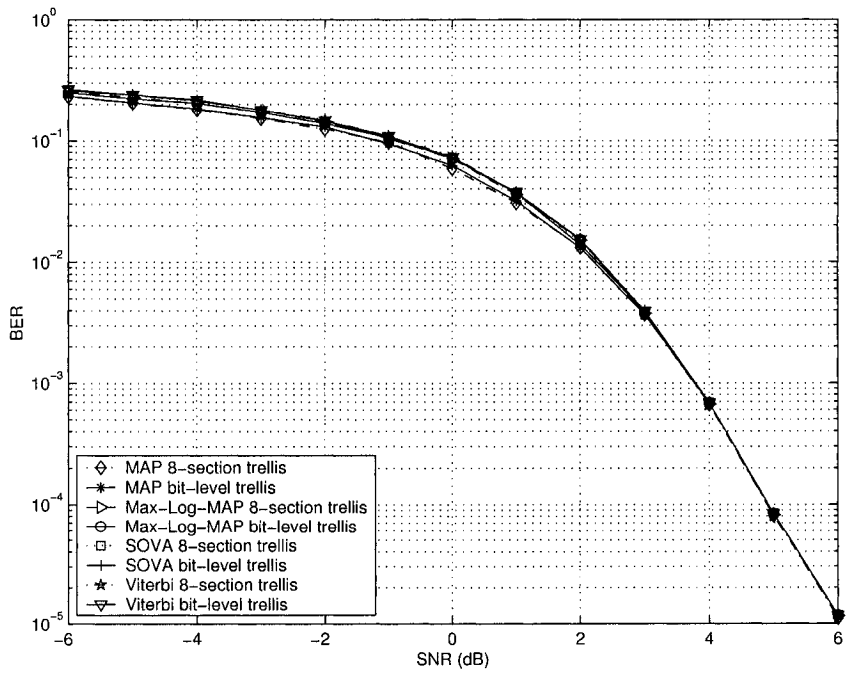


Fig. B.4 Bit-error performance of MAP, Max-Log-MAP, SOVA, and Viterbi decoding of the (32, 26) RM code.

References

[1] C. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423, July 1949.

[2] R. Pyndiah, "Near-optimum decoding of product codes: Block Turbo Codes," *IEEE Trans. Communications*, vol. 46, pp. 1003–1010, Aug. 1998.

[3] A. Lafourcade and A. Vardy, "Optimal sectionalization of a trellis," *IEEE Trans. Inform. Theory*, vol. 42, pp. 689–703, May 1996.

[4] Y. Liu, S. Lin, and M. Fossorier, "MAP algorithms for decoding linear block codes based on sectionalized trellis diagrams," *IEEE Trans. Communications*, vol. 48, pp. 577–587, Apr. 2000.

[5] D. Muller, "Application of Boolean algebra to switching circuit design and to error detection," *IEEE Trans. Computers*, vol. 3, pp. 6–12, 1954.

[6] I. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *IEEE Trans. Inform. Theory*, vol. 4, pp. 38–49, Sept. 1954.

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

[8] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, Mar. 1974.

[9] J. Wolf, "Efficient maximum-likelihood decoding of linear block codes using a trellis," *IEEE Trans. Inform. Theory*, vol. 24, pp. 76–80, Jan. 1978.

[10] G. Forney, "Coset Codes II: Binary lattices and related codes," *IEEE Trans. Inform. Theory*, vol. 34, pp. 1152–1187, Sept. 1988.

[11] J. Omura, "On the Viterbi decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 15, pp. 177–179, Jan. 1969.

[12] G. Forney, "The Viterbi algorithm," in *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.

[13] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 13, pp. 260–269, Apr. 1967.

[14] M. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the equivalence between SOVA and Max-Log-MAP decoding," *IEEE Commun. Lett.*, vol. 2, pp. 137–139, May 1998.

[15] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE Globecom Conference*, (Dallas, TX), pp. 1680–1686, Nov. 1989.

[16] C. Berrou, P. Adde, E. Angui, and S. Faudeil, "A low complexity soft-output Viterbi decoder architecture," in *Proc. IEEE Int. Conf. Communications*, pp. 737–740, May 1993.

[17] B. Vucetic and J. Yuan, *Turbo Codes: Principles and Applications*. Boston, MA: Kluwer Academic Publishers, 2000.

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

[19] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-Codes," *IEEE Trans. Communications*, vol. 44, pp. 1261–1271, Oct. 1996.

[20] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, "Near optimum decoding of product codes," in *Proc. IEEE Globecom '94 Conference*, vol. 1, (San Francisco, CA), pp. 339–343, Nov. 1994.

[21] P. Adde, R. Pyndiah, O. Raoul, and J. Inisan, "Block turbo decoder design," in *Proc. IEEE Int. Symposium on Turbo Codes*, (Brest, France), pp. 166–169, Sept. 1997.

[22] P. Elias, "Error-free coding," *IRE Trans. Inform. Theory*, vol. 4, pp. 29–37, Sept. 1954.

[23] D. Divsalar and F. Pollara, "Turbo codes for deep-space communications," Tech. Rep. 42-120, TDA, Feb. 1995.

[24] C. Edwards, C. Steilzvied, L. Deutsch, and L.Swanson, "NASA's deep-space telecommunications road map," Tech. Rep. 42-126, TMO, Feb. 1999.

[25] S. Barbulescu, W. Farrell, P. Gray, and M. Rice, "Bandwidth efficient turbo coding for high speed mobile satellite communications," in *Proc. IEEE Int. Symposium on Turbo Codes*, (Brest, France), pp. 119–126, Sept. 1997.

[26] D. Wang, "High-performance SOVA decoding for Turbo Codes over cdma2000 mobile radio," *IEEE Trans. Inform. Theory*, vol. 42, pp. 189–193, May 2000.

[27] R. Pyndiah, "Iterative decoding of product codes: Block Turbo-Codes," in *Proc. IEEE Int. Symposium on Turbo Codes*, (Brest, France), pp. 71–79, Sept. 1997.

[28] U. Vilaipornsawai and M. R. Soleymani, "Trellis-based iterative decoding of block codes for satellite ATM," *IEEE Trans. Inform. Theory*, vol. 42, pp. 2947–2951, May 2002.

[29] D. Chase, "A class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inform. Theory*, vol. 18, pp. 170–182, Jan. 1972.

[30] S. Dave, J. Kim, and S. Kwatra, "Turbo Block Codes using modified Kaneko's algorithm," *IEEE Trans. Inform. Theory*, pp. 181–183, Oct. 2000.

[31] S. Shin, S. Lee, and S. Lee, "Evaluation of block turbo code performance with the reduced search trellis decoding method," in *Proc. IEEE Int. Conf. Communications*, vol. 148, pp. 125–131, June 2001.

[32] P. Luukkanen and P. Zhang, "Comparison of optimum and sub-optimum Turbo Decoding schemes in 3rd generation cdma2000 mobile system," *IEEE Trans. Inform. Theory*, vol. 42, pp. 437–441, May 1999.

[33] L. Lin and R. Cheng, "Improvements in SOVA-based decoding for Turbo Codes," *IEEE Trans. Inform. Theory*, pp. 1473–1478, June 1997.

[34] R. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1072–1092, July 1996.

[35] F. Kschischang and V. Sorokine, "On the trellis structure of block codes," *IEEE Trans. Inform. Theory*, vol. 41, pp. 1924–1937, Nov. 1995.

[36] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On structural complexity of the L-section minimum trellis diagrams for binary linear block codes," *IEICE Trans. Fundamentals*, vol. 76, pp. 1411–1421, Sept. 1993.

[37] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On complexity of trellis structure of linear block codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1057–1064, May 1993.

[38] A. Kiely, S. Dolinar, R. McEliece, L. Ekroot, and W. Lin, "Trellis decoding complexity of linear block codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1687–1697, Nov. 1996.

[39] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc. IEEE Int. Conf. Communications*, (Seattle), pp. 1009–1013, June 1995.

[40] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On the optimum bit orders with respect to the state complexity of trellis diagrams for binary linear codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 242–245, Jan. 1993.

[41] A. Vardy and Y. Be'ery, "Maximum-likelihood soft decision decoding of BCH codes," *IEEE Trans. Inform. Theory*, vol. 40, pp. 546–554, Mar. 1994.