# MAP ALGORITHMS FOR DECODING LINEAR BLOCK CODES BASED ON SECTIONALIZED TRELLIS DIAGRAMS

**Ye Liu, Marc Fossorier**
**and**
**Shu Lin**

**January 26, 1999**

# MAP ALGORITHMS FOR DECODING LINEAR BLOCK CODES BASED ON SECTIONALIZED TRELLIS DIAGRAMS


## Abstract

The MAP algorithm is a trellis-based maximum a posteriori probability decoding algorithm. It is the heart of the turbo (or iterative) decoding which achieves an error performance near the Shannon limit. Unfortunately, the implementation of this algorithm requires large computation and storage. Furthermore, its forward and backward recursions result in long decoding delay. For practical applications, this decoding algorithm must be simplified and its decoding complexity and delay must be reduced. In this paper, the MAP algorithm and its variations, such as Log-MAP and Max-Log-MAP algorithms, are first applied to sectionalized trellises for linear block codes and carried out as two-stage decodings. Using the structural properties of properly sectionalized trellises, the decoding complexity and delay of the MAP algorithms can be reduced. Computation-wise optimum sectionalizations of a trellis for MAP algorithms are investigated. Also presented in this paper are bi-directional and parallel MAP decodings.

# 1 Introduction

Maximum likelihood decoding (MLD) minimizes the block ( or sequence) error probability, however, it does not necessarily minimize the bit (or symbol) error probability. It only delivers hard decoded bits, called hard outputs, without providing their reliability measures. In many error control coding schemes, it is desirable to provide both decoded bits and their reliability values, called soft outputs, for further processing to improve the system error performance. A decoding algorithm that allows to process soft-decision inputs and produces soft-decision outputs is called a soft-in/soft-out (SISO) decoding algorithm. The most well known SISO decoding algorithm is the MAP (maximum a posteriori probability) decoding algorithm that was devised by Bahl, Cocke, Jelinek and Raviv in 1974 [1]. This algorithm is a trellis-based decoding algorithm for both block and terminated convolutional codes. It is devised to minimize the symbol error probability. Unfortunately, the implementation of this algorithm requires large computation and storage. Furthermore, its forward and backward recursions result in long decoding delay. As a result, the potential of the MAP algorithm has not been studied until most recently when this algorithm was used in turbo (or iterative) decoding to achieve an error performance near the Shannon limit [2]. In the study of turbo decoding, a focal point is the MAP algorithm which is actually the heart of turbo decoding. The concern is how to simplify this decoding algorithm and reduce both the decoding complexity and delay.

This paper investigates decoding complexity and delay of the MAP algorithm and its variations, such as Log-MAP [3] and Max-Log-MAP [4] algorithms, for linear block codes. For decoding linear block codes, conventional MAP and Max-Log-MAP algorithms have been devised based on bit-level code trellises [1, 4]. In this paper, it is shown that these decoding algorithms can be modified and carried out based on sectionalized code trellises. Proper sectionalization of a code trellis results in a significant reduction of decoding computational complexity. Computation-wise optimum sectionalizations of a code trellis for MAP, Log-MAP and Max-Log-MAP algorithms are investigated. Also investigated in this paper are bi-directional and parallel MAP decodings based on the structural properties of code trellises. Bi-directional decoding can reduce the decoding delay by a factor close to two. Parallel decoding not only simplifies decoding complexity but also speeds up the decoding process.

The organization of this paper is as follows. Section 2 gives a brief review of sectionalized

trellises for linear block codes. In Section 3, a modified MAP algorithm based on a sectionalized code trellis is first described and its computational complexity and storage requirement are analyzed. Then optimum sectionalization of a code trellis to minimize the computational complexity is considered. Modified Max-Log-MAP and Log-MAP algorithms based on a sectionalized code trellis and their computational complexities are presented in Section 4. Bi-directional and parallel MAP and Max-log-MAP algorithms are described in Section 5. Section 6 summarizes the results and concludes the paper.

# 2 A Brief Review of Sectionalized Trellises

Consider an $(n, k)$ linear block code $C$ with a minimal $n$-section bit-level trellis $T$ in which each branch represents a single code bit [5, 6]. For any positive integer $\nu$ such that $1 \leq \nu \leq n$, the bit-level trellis $T$ can be sectionalized into $\nu$ sections with section boundary locations in $U = \{h_0, h_1, \cdots, h_\nu\}$ where $0 = h_0 < h_1 < \cdots < h_\nu = n$. This sectionalization results in an $\nu$-section trellis $T(U)$. At boundary location $h_j$, the state space (the set of states) is denoted $\Sigma_{h_j}(C)$. The $j$-th section of $T(U)$, denoted $T_j$, consists of the state space $\Sigma_{h_{j-1}}(C)$ at time-$h_{j-1}$, the state space $\Sigma_{h_j}(C)$ at time-$h_j$ and the branches that connect the states in $\Sigma_{h_{j-1}}(C)$ and the states in $\Sigma_{h_j}(C)$. A branch in this section represents $m_j = h_j - h_{j-1}$ code bits. Two adjacent states $\sigma'$ and $\sigma$ with $\sigma' \in \Sigma_{h_{j-1}}(C)$ and $\sigma \in \Sigma_{h_j}(C)$ may be connected by multiple branches, called *parallel branches*. For convenience, we say that these parallel branches form a *composite branch*, denoted $L(\sigma', \sigma)$. Each branch $\mathbf{b}(\sigma', \sigma) \in L(\sigma', \sigma)$ is labeled by a $m_j$-tuple $(v_{h_{j-1}+1}, v_{h_{j-1}+2}, \cdots, v_{h_j})$, where $v_{h_{j-1}+i} = \pm 1$ for BPSK signaling with unit energy.

A codeword in $C$ is denoted by $\mathbf{v} = (v_1, v_2, \cdots, v_n)$. Let $C_{h_{j-1},h_j}$ denote the linear subcode of $C$ which consists of those codewords in $C$ whose components are all zero except for the $h_j - h_{j-1}$ components from the $(h_{j-1}+1)$-th bit position to the $h_j$-th bit position. Let $p_{h_{j-1},h_j}(C)$ be the punctured code of length $h_j - h_{j-1}$ obtained from $C$ by removing the first $h_{j-1}$ and last $n - h_j$ components from each codeword in $C$. For a linear code $A$, let $k(A)$ denotes its dimension. Then the number of states at time-$h_j$ in $T(U)$ is given by [5]:

$$|\Sigma_{h_j}(C)| = 2^{k(C)-k(C_{0,h_j})-k(C_{h_j,n})}. \tag{1}$$

The number of composite branches converging into (or entering) a state $\sigma_{h_j} \in \Sigma_{h_j}(C)$ at

time-$h_j$, called the incoming degree of $\sigma_{h_j}$, is given by

$$\deg(\sigma_{h_j})_{\text{in}} \triangleq 2^{k(C_{0,h_j})-k(C_{0,h_{j-1}})-k(C_{h_{j-1},h_j})}. \tag{2}$$

The number of composite branches diverging from (or leaving) a state $\sigma_{h_{j-1}} \in \Sigma_{h_{j-1}}(C)$ at time-$h_{j-1}$, called the outgoing degree of $\sigma_{h_{j-1}}$, is given by

$$\deg(\sigma_{h_{j-1}})_{\text{out}} \triangleq 2^{k(C_{h_{j-1},n})-k(C_{h_j,n})-k(C_{h_{j-1},h_j})}. \tag{3}$$

The branch complexity of the trellis section $T_j$ is measured by: (1) the size of a composite branch, denote $|B_j^p|$; (2) the number of distinct composite branches in the trellis section, denoted $|B_j^d|$; (3) the total number of composite branches in the trellis section, denoted $|B_j^c|$. These branch complexity parameters are given below [5]:

$$\begin{aligned} |B_j^p| &= 2^{k(C_{h_{j-1},h_j})}, \\ |B_j^d| &= 2^{k(p_{h_{j-1},h_j}(C))-k(C_{h_{j-1},h_j})}, \\ |B_j^c| &= |\Sigma_{h_j}(C)| \cdot \deg(\sigma_{h_j})_{\text{in}} = 2^{k(C)-k(C_{h_j,n})-k(C_{0,h_{j-1}})-k(C_{h_{j-1},h_j})}. \end{aligned} \tag{4}$$

In the trellis section $T_j$, each distinct composite branch appears

$$2^{k(C)-k(C_{0,h_{j-1}})-k(C_{h_j,n})-k(p_{h_{j-1},h_j}(C))}$$

times. In general, $|B_j^c|$ is much larger than $|B_j^d|$. This fact will be used in reducing the computational complexity of MAP algorithms.

# 3 A Modified MAP Algorithm Based on a Sectionalized Trellis

For decoding block codes, the conventional MAP algorithm has been devised based on bit-level trellises [1][4]. In this section, we show that the MAP algorithm can be carried out in two stages based on sectionalized trellises. Proper trellis sectionalization results in a significant reduction in storage requirement and computational complexity.

## 3.1 The algorithm

Consider an $\nu$-section trellis $T(U)$ for an $(n,k)$ linear block code $C$ with section boundary locations in $U = \{h_0, h_1, \cdots, h_\nu\}$. Let $L(\sigma', \sigma)$ be a composite branch in the $j$-th section $T_j$ of $T(U)$ which connects state $\sigma'$ to state $\sigma$ with $\sigma' \in \Sigma_{h_{j-1}}(C)$ and $\sigma \in \Sigma_{h_j}(C)$. Each branch $\mathbf{b}(\sigma', \sigma) \in L(\sigma', \sigma)$ consists of $m_j = h_j - h_{j-1}$ code bits, denoted

$$\mathbf{b}(\sigma', \sigma) = (v_{h_{j-1}+1}, v_{h_{j-1}+2}, \cdots, v_{h_j}).$$

Let $\mathbf{r} = (r_1, r_2, \cdots, r_n)$ be the received sequence and $\mathbf{r}_j = (r_{h_{j-1}+1}, r_{h_{j-1}+2}, \cdots, r_{h_j})$ denote the $j$-th section of $\mathbf{r}$. For each branch $\mathbf{b}(\sigma', \sigma) \in L(\sigma', \sigma)$, define the following probability:

$$\begin{aligned} \gamma_{h_j}(\mathbf{b}(\sigma', \sigma)) &\triangleq p(\sigma, \mathbf{b}(\sigma', \sigma), \mathbf{r}_j | \sigma') \\ &= p(\sigma, \mathbf{b}(\sigma', \sigma) | \sigma') p(\mathbf{r}_j | (\sigma', \sigma), \mathbf{b}(\sigma', \sigma)). \end{aligned} \tag{5}$$

The value $\gamma_{h_j}(\mathbf{b}(\sigma', \sigma))$ represents the transition probability from state $\sigma'$ to state $\sigma$ through the branch $\mathbf{b}(\sigma', \sigma)$ that produces $\mathbf{r}_j$. For simplicity, we call it branch (transition) probability. Then the probability of the composite branch $L(\sigma', \sigma)$ connecting state $\sigma'$ to state $\sigma$ that results in the received vector $\mathbf{r}_j$ is given by

$$\gamma_{h_j}(L(\sigma', \sigma)) \triangleq \sum_{\mathbf{b}(\sigma', \sigma) \in L(\sigma', \sigma)} \gamma_{h_j}(\mathbf{b}(\sigma', \sigma)). \tag{6}$$

For the MAP algorithm based on the sectionalized trellis $T(U)$, the forward and backward recursions are to compute the following state probabilities:

$$\alpha_{h_j}(\sigma) = \sum_{\sigma' \in \Omega_{h_{j-1}}(\sigma)} \gamma_{h_j}(L(\sigma', \sigma)) \alpha_{h_{j-1}}(\sigma'), \tag{7}$$

$$\beta_{h_{j-1}}(\sigma') = \sum_{\sigma \in \Omega_{h_j}(\sigma')} \gamma_{h_j}(L(\sigma', \sigma)) \beta_{h_j}(\sigma), \tag{8}$$

where $\Omega_{h_{j-1}}(\sigma)$ denotes the set of states in $\Sigma_{h_{j-1}}(C)$ that are adjacent to state $\sigma$ and $\Omega_{h_j}(\sigma')$ denotes the set of states in $\Sigma_{h_j}(C)$ that are adjacent to state $\sigma'$. The forward and backward recursions are initialized with $\alpha_{h_0}(\sigma_0) = 1$ and $\beta_{h_\nu}(\sigma_f) = 1$, where $\sigma_0$ and $\sigma_f$ are the initial state and final state of the trellis $T(U)$ at time-0 and time-$n$, respectively.

For computing the log-likelihood ratio (LLR) of the code bit $v_l$, $h_{j-1} + 1 \leq l \leq h_j$, define

$$\gamma_{h_j}(L^+_{v_l}(\sigma', \sigma)) \triangleq \sum_{\substack{\mathbf{b}(\sigma', \sigma) \in L(\sigma', \sigma) \\ v_l = +1}} \gamma_{h_j}(\mathbf{b}(\sigma', \sigma)), \tag{9}$$

$$\gamma_{h_j}(L_{v_l}^{-}(\sigma',\sigma)) \triangleq \sum_{\substack{\mathbf{b}(\sigma',\sigma)\in L(\sigma',\sigma) \\ v_l=-1}} \gamma_{h_j}(\mathbf{b}(\sigma',\sigma)). \tag{10}$$

It follows from the definitions of (6), (9) and (10) that

$$\gamma_{h_j}(L(\sigma',\sigma)) = \gamma_{h_j}(L_{v_l}^{+}(\sigma',\sigma)) + \gamma_{h_j}(L_{v_l}^{-}(\sigma',\sigma)), \tag{11}$$

for every code bit $v_l$ in $T_j$. Then the LLR of $v_l$ is given by

$$L(\hat{v}_l) = \log \frac{\sum_{\substack{(\sigma',\sigma) \\ v_l=+1}} \alpha_{h_{j-1}}(\sigma')\gamma_{h_j}(L_{v_l}^{+}(\sigma',\sigma))\beta_{h_j}(\sigma)}{\sum_{\substack{(\sigma',\sigma) \\ v_l=-1}} \alpha_{h_{j-1}}(\sigma')\gamma_{h_j}(L_{v_l}^{-}(\sigma',\sigma))\beta_{h_j}(\sigma)}, \tag{12}$$

where the summations are over all the adjacent state pairs $(\sigma',\sigma)$ with $\sigma' \in \Sigma_{h_{j-1}}(C)$ and $\sigma \in \Sigma_{h_j}(C)$.

To carry out the decoding process, we must first compute the composite branch probabilities, $\gamma_{h_j}(L(\sigma',\sigma))$'s. In a sectionalized trellis, a section may consists of many composite branches, however, the number of distinct composite branches is relatively small. In computing $\alpha$'s, $\beta$'s and LLR's, we only need to compute the probabilities of the distinct composite branches. Based on this fact, we may perform a preprocessing step to compute $\gamma_{h_j}(L(\sigma',\sigma))$, $\gamma_{h_j}(L_{v_l}^{-}(\sigma',\sigma))$, and $\gamma_{h_j}(L_{v_l}^{+}(\sigma',\sigma))$ for each distinct composite branch $L(\sigma',\sigma)$ and each code bit $v_l$ and store them in a table called the $\gamma_{h_j}$-table. This preprocessing step reduces computational complexity of the MAP algorithm significantly.

The MAP algorithm based on a sectionalized trellis $T(U)$ can be carried out in two stages. At the first stage, the parallel branches of each distinct composite branch are preprocessed to form the $\gamma$-tables. Then, the MAP decoding is performed with parallel branches in a composite branch viewed as a single branch.

Consider the computation of a branch transition probability $\gamma_{h_j}(\mathbf{b}(\sigma',\sigma))$ given by (5). For equally likely signaling and a linear code, all the states at any section boundary are equiprobable. Then

$$p(\sigma,\mathbf{b}(\sigma',\sigma)|\sigma') = \frac{1}{\sum_{\sigma''\in\Omega_{h_j}(\sigma')}|L(\sigma',\sigma'')|}. \tag{13}$$

As $|\Omega_{h_j}(\sigma')|$ and $|L(\sigma',\sigma'')|$ remain the same for all states $\sigma' \in \Sigma_{h_{j-1}}(C)$ [5], $p(\sigma,\mathbf{b}(\sigma',\sigma)|\sigma')$ is constant in the trellis section $T_j$. Since we are only interested in the ratio given by (12), we can

scale the branch transition probability $\gamma_{h_j}(\mathbf{b}(\sigma',\sigma))$ by any factor without changing the LLR of an estimated code bit. Therefore, in computing $\alpha_{h_j}(\sigma)$, $\beta_{h_{j-1}}(\sigma')$ and $L(v_l)$, we can use

$$w_{h_j}(\mathbf{b}(\sigma',\sigma)) \triangleq p(\mathbf{r}_j|(\sigma',\sigma),\mathbf{b}(\sigma',\sigma)) \tag{14}$$

instead of $\gamma_{h_j}(\mathbf{b}(\sigma',\sigma))$ to simplify computations. For an AWGN channel with zero mean and variance $N_0/2$,

$$w_{h_j}(\mathbf{b}(\sigma',\sigma)) = \left(\frac{1}{\pi N_0}\right)^{m_j/2} \exp\{-\sum_{l=h_{j-1}+1}^{h_j} (r_l - v_l)^2/N_0\}. \tag{15}$$

To construct the $\gamma$-tables for the $\nu$-section trellis $T(U)$, we use the following procedure: For $1 \leq j \leq \nu$,

(1) Compute $w_{h_j}(\mathbf{b}(\sigma',\sigma))$ using (15) for all distinct branches in trellis section $T_j$.

(2) Compute $\gamma_{h_j}(L^+_{v_l}(\sigma',\sigma))$ for each code bit $v_l$ for $h_{j-1}+1 \leq l \leq h_j$.

(3) Compute $\gamma_{h_j}(L^-_{v_{h_{j-1}+1}}(\sigma',\sigma))$ for the code bit $v_{h_{j-1}+1}$. Then, it follows from from (11) that

$$\gamma_{h_j}(L(\sigma',\sigma)) = \gamma_{h_j}(L^-_{v_{h_{j-1}+1}}(\sigma',\sigma)) + \gamma_{h_j}(L^+_{v_{h_{j-1}+1}}(\sigma',\sigma)).$$

(4) Finally, using (11), we obtain $\gamma_{h_j}(L^-_{v_l}(\sigma',\sigma))$ for $h_{j-1}+2 \leq l \leq h_j$, by computing the difference, $\gamma_{h_j}(L(\sigma',\sigma)) - \gamma_{h_j}(L^+_{v_l}(\sigma',\sigma))$.

From (7), we see that $\alpha_{h_j}(\sigma)$'s can be computed along with the construction of $\gamma$-tables from the initial state $\sigma_0$ to the final state $\sigma_f$ of the code trellis in forward direction. If bi-directional decoding (see Section 5) is performed, it follows from (8) that $\beta_{h_{j-1}}(\sigma')$'s can be computed along with the construction of $\gamma$-tables from the final state $\sigma_f$ to the initial state $\sigma_0$ of the code trellis in backward direction. As soon as $\alpha_{h_j}(\sigma')$'s and $\beta_{h_{j-1}}(\sigma)$'s at the boundaries of the $j$-th trellis section $T_j$ have been computed, the LLR's of the code bits $v_l$ with $h_{j-1}+1 \leq l \leq h_j$ are evaluated from (12).

## 3.2 Computational complexity and storage requirement

The computational complexity of the MAP algorithm is measured in terms of the number of real operations required to decode a received sequence. It can be analyzed by enumerating

the numbers of real operations required for constructing the $\gamma$-tables and computing $\alpha$'s, $\beta$'s and LLR's of the estimated code bits. We assume that computations of $\exp(\cdot)$ and $\log(\cdot)$ are accomplished by using a read-only memory (ROM) (i.e., table look-up).

To construct the $\gamma_{h_j}$-table for the $j$-th trellis section $T_j$ based on the procedure presented in the previous subsection, the computations required at each step are given below:

(1) step-1 requires $|B_j^d| \cdot |B_j^p| \cdot m_j$ additions and $|B_j^d| \cdot |B_j^p| \cdot (m_j - 1)$ multiplications.

(2) step-2 to step-4 require $|B_j^d| \cdot (|B_j^p|/2 - 1) \cdot m_j$, $|B_j^d| \cdot (|B_j^p|/2 - 1) + 1$ and $m_j - 1$ additions, respectively.

Therefore, construction of $\gamma_{h_j}$-table for the trellis section $T_j$ requires a total of

$$N_a^j(\gamma) = |B_j^d| \cdot |B_j^p| \cdot m_j + |B_j^d| \cdot (|B_j^p|/2 - 1) \cdot m_j + |B_j^d| \cdot (|B_j^p|/2 - 1) + m_j \tag{16}$$

additions and a total of

$$N_m^j(\gamma) = |B_j^d| \cdot |B_j^p| \cdot (m_j - 1) \tag{17}$$

multiplications.

Computations of $\alpha$'s and $\beta$'s from (7) and (8) require a total of $N_a^j(\alpha) + N_a^j(\beta)$ additions and $N_m^j(\alpha) + N_m^j(\beta)$ multiplications, where

$$N_a^j(\alpha) = (\deg(\sigma_{h_j})_{\text{in}} - 1) \cdot |\Sigma_{h_j}(C)| = |B_j^c| - |\Sigma_{h_j}(C)|, \tag{18}$$

$$N_a^j(\beta) = (\deg(\sigma_{h_{j-1}})_{\text{out}} - 1) \cdot |\Sigma_{h_{j-1}}(C)| = |B_j^c| - |\Sigma_{h_{j-1}}(C)|, \tag{19}$$

$$N_m^j(\alpha) = N_m^j(\beta) = |B_j^c|. \tag{20}$$

The last step of MAP decoding is to compute the LLR's of the estimated code bits $v_l$ for $h_{j-1} + 1 \le l \le h_j$ with $1 \le j \le \nu$. Define

$$S^{(j)} \triangleq \sum_{(\sigma', \sigma)} \alpha_{h_{j-1}}(\sigma') \gamma_{h_j}(L(\sigma', \sigma)) \beta_{h_j}(\sigma), \tag{21}$$

$$S_{+1}^{(j)}(v_l) \triangleq \sum_{\substack{(\sigma', \sigma) \\ v_l = +1}} \alpha_{h_{j-1}}(\sigma') \gamma_{h_j}(L_{v_l}^{+}(\sigma', \sigma)) \beta_{h_j}(\sigma), \tag{22}$$

$$S_{-1}^{(j)}(v_l) \triangleq \sum_{\substack{(\sigma', \sigma) \\ v_l = -1}} \alpha_{h_{j-1}}(\sigma') \gamma_{h_j}(L_{v_l}^{-}(\sigma', \sigma)) \beta_{h_j}(\sigma). \tag{23}$$

It follows from (11) and (21) to (23) that

$$S^{(j)} = S^{(j)}_{+1}(v_l) + S^{(j)}_{-1}(v_l) \tag{24}$$

for $h_{j-1}+1 \leq l \leq h_j$. To compute the LLR $L(v_l)$'s from (12), we need to compute $S^{(j)}_{+1}(v_l)$'s and $S^{(j)}_{-1}(v_l)$'s. Computations of $S^{(j)}_{+1}(v_l)$'s and $S^{(j)}_{-1}(v_l)$'s can be done efficiently by using the following procedure:

(1) Compute $S^{(j)}$ from (21), which requires $|B^c_j| - 1$ additions and $2|B^c_j|$ multiplications.

(2) Compute $S^{(j)}_{+1}(v_l)$'s from (22) for $h_{j-1}+1 \leq l \leq h_j$, which requires $(|B^c_j|-1)\cdot m_j$ additions and $|B^c_j| \cdot m_j$ multiplications (using partial results from step-1).

(3) Compute $S^{(j)}_{-1}(v_l)$'s from (24) by taking the differences, $S^{(j)} - S^{(j)}_{+1}(v_l)$, for $h_{j-1}+1 \leq l \leq h_j$. This step requires $m_j$ subtractions ( equivalent to additions).

Once $S^{(j)}_{+1}(v_l)$'s and $S^{(j)}_{-1}(v_l)$'s have been computed, the LLR's of estimated code bits $v_l$ for $h_{j-1}+1 \leq l \leq h_j$ can be evaluated, which requires $m_j$ divisions (a division operation is assumed to have the same complexity as a multiplication operation). Therefore, computation of the LLR's of estimated code bits corresponding to the trellis section $T_j$ requires a total of

$$N^j_a(L) = |B^c_j| - 1 + |B^c_j| \cdot m_j \tag{25}$$

additions and a total of

$$N^j_m(L) = 2|B^c_j| + (|B^c_j| + 1) \cdot m_j \tag{26}$$

multiplications (including $m_j$ divisions).

Summarizing the above results, execution of the MAP algorithm based on the sectionalized trellis $T(U)$ requires a total of

$$N_a(U) = \sum_{j=0}^{\nu-1}(N^j_a(\gamma) + N^j_a(L)) + \sum_{j=1}^{\nu-1} N^j_a(\alpha) + \sum_{j=0}^{\nu-2} N^j_a(\beta) \tag{27}$$

additions and a total of

$$N_m(U) = \sum_{j=0}^{\nu-1}(N^j_m(\gamma) + N^j_m(L)) + \sum_{j=1}^{\nu-1} N^j_m(\alpha) + \sum_{j=0}^{\nu-2} N^j_m(\beta) \tag{28}$$

multiplications. The numbers $N_a(U)$ and $N_m(U)$ together give a measure of computational complexity of the MAP algorithm based on the sectionalized trellis $T(U)$.

During the decoding process, the $\gamma$-tables must be stored for the computations of $\alpha$'s, $\beta$'s and LLR's of the estimated code bits. This requires

$$M(\gamma) = \sum_{j=0}^{\nu-1} |B_j^d| \cdot (2m_j + 1) \tag{29}$$

storage locations (or unites). If bi-directional decoding is performed, we also need to store $\alpha$'s and $\beta$'s for the computations of LLR's of the estimated code bits before the forward and backward recursions meet at the middle of the trellis. This requires $\frac{M(\alpha)+M(\beta)}{2}$ storage locations, where

$$M(\alpha) = \sum_{j=1}^{\nu-1} |\Sigma_{h_j}(C)|, \tag{30}$$

$$M(\beta) = \sum_{j=0}^{\nu-2} |\Sigma_{h_{j-1}}(C)|. \tag{31}$$

If the LLR's of estimated code bits are to be used for further decoding process, they must be also stored. This requires another

$$M(L) = n \tag{32}$$

storage locations. Therefore, the total storage requirement for MAP decoding based on sectionalized trellis $T(U)$ is

$$M(U) = M(\gamma) + \frac{M(\alpha) + M(\beta)}{2} + M(L). \tag{33}$$

## 3.3 Optimum sectionalizations

A code trellis can be sectionalized in many ways. Sectionalizations that give the smallest number of computation operations and memory storages are called computation-wise optimum sectionalizations and memory-wise optimum sectionalizations, respectively. An algorithm for finding computation-wise optimal sectionalizations for Viterbi decoding of block codes has been devised by Lafourcade and Vardy [7]. This algorithm can be applied to MAP algorithms.

The Lafourcade and Vardy algorithm is based on the following simple structure: For any two integers $x$ and $y$ with $0 \leq x < y \leq n$, a section from time-$x$ to time-$y$ in any sectionalized

trellis $T(U)$ with $x, y \in U$ and $x+1, x+2, \cdots, y-1 \notin U$ is identical [5]. Let $\varphi(x,y)$ denote the number of computations (or storage unites) required to compute $\gamma$'s, $\alpha$'s, $\beta$'s and LLR's of the MAP algorithm to process the trellis section from time-$x$ to time-$y$. This number $\varphi(x,y)$ is solely determined by the choices of $x$ and $y$. Let $\varphi_{\min}(x,y)$ denote the smallest number of computations (or storage unites) required in MAP decoding to process the trellis section(s) from time-$x$ to time-$y$ in any sectionalized trellis $T(U)$ with $x, y \in U$. The value $\varphi_{\min}(0,n)$ gives the smallest total number of computations (or storage unites) of the MAP algorithm for processing the code trellis with an optimum sectionalization. It follows from the definitions of $\varphi(x,y)$ and $\varphi_{\min}(x,y)$ that

$$\varphi_{\min}(0,y) = \begin{cases} \min\{\varphi(0,y), \min_{0<x<y}\{\varphi_{\min}(0,x) + \varphi(x,y)\}\}, & \text{for } 1 < y \leq n \\ \varphi(0,1), & \text{for } y = 1. \end{cases} \tag{34}$$

For every $y \in \{1, 2, \cdots, n\}$, $\varphi_{\min}(0,y)$ can be computed as follows. The values of $\varphi(x,y)$ for $0 \leq x < y \leq n$ are computed using the structure of the trellis section from time-$x$ to time-$y$. First, $\varphi_{\min}(0,x)$ is computed. The value $\varphi_{\min}(0,y)$ can be computed from $\varphi_{\min}(0,x)$ and $\varphi(x,y)$ with $0 < x < y$ using (34). By storing the information when the minimum value occurs in the right-hand side of (34), an optimum sectionalization is found from the computation of $\varphi_{\min}(0,n)$.

The computational complexity and storage requirement of MAP decoding of a linear block code very much depend on the sectionalization of the code trellis. A sectionalization that minimizes both is desirable. However, such a sectionalization in general does not exist. If there is no severe constraint on the size of memory storage, we may choose a sectionalization that minimizes the computational complexity. Based on the above analysis, decoding computation of the MAP algorithm involves two kinds of real number operations, additions and multiplications, in every decoding step. A multiplication operation is more complex than an addition operation and they can not be treated the same (to have the same weight) in the minimization of computational complexity. This makes it hard (if not impossible) to find a sectionalization that minimizes both the number of additions and the number of multiplications. Since the number of multiplications required in the MAP decoding is much larger than the number of additions required and a multiplication operation is much more complex than an addition operation, we may just find a trellis sectionalization to minimize the total number of multiplications. Alternatively we can weight an addition operation as a fraction of a multiplication operations, say 1/20 of a multiplication. Then we find a trellis sectionalization to minimize the total number of multiplications

and weighted additions. Optimum trellis sectionalizations (in terms of minimizing the number of multiplication operations) of some RM codes for MAP decoding are given in Table 1. For comparison purpose, the computational complexity and storage requirement based on the bit-level trellis for each code are also included. From this table, we see that bit-level trellis requires much larger computational complexity and storage requirement than an optimum sectionalized trellis. We also see that the optimum trellis sectionalization in terms of minimizing the number of multiplication operations may not reduce the number of addition operations. So there is a trade-off between numbers of additions and multiplications.

From Table 1, we see that optimum sectionalization is more efficient for low-rate codes than for high-rate codes. This is because that $|B_j^d|$ tends to be large in high-rate codes [5].

From Table 1, we also see that optimum sectionalization of a code trellis, in general, results in a non-uniform sectionalized trellis (trellis sections are not equal in length). For IC implementation of a MAP decoder, since using the same hardware to process all the trellis sections is highly desirable, a uniformly sectionalized trellis seems a better choice. Table 2 gives the best uniform sectionalizations for RM codes with the MAP decoding. Uniform sectionalizations marked with "*" are also optimal sectionalizations. From Tables 1 and 2, we observe that the best uniform sectionalization only requires slightly more operations than the optimum sectionalization.

# 4 The Max-Log-MAP Algorithm Based on Sectionalized Trellises

The Max-Log-MAP algorithm is a suboptimum version of the MAP algorithm [3, 4]. It provides an efficient trade-off between error performance and decoding complexity. This algorithm is based on a very simple approximation for logarithm of a sum of real numbers. For a finite set of real numbers, $\{\delta_1, \delta_2, \cdots, \delta_q\}$, the following approximation holds:

$$\log(\sum_{i=1}^{q} e^{\delta_i}) \simeq (\max_{1 \le i \le q} \{\delta_i\}). \tag{35}$$

This approximation is called *maximum logarithm (max-log) approximation.*

## 4.1 The algorithm

Using the max-log approximation, the LLR of an estimated code bit given by (12) based on a sectionalized code trellis $T(U)$ can be can be approximated by

$$\tilde{L}(\hat{v}_l) = \max_{\substack{(\sigma',\sigma) \\ v_l=+1}} \{\log \alpha_{h_{j-1}}(\sigma') + \log \gamma_{h_j}(L_{v_l}^{+}(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)\} - \max_{\substack{(\sigma',\sigma) \\ v_l=-1}} \{\log \alpha_{h_{j-1}}(\sigma') + \log \gamma_{h_j}(L_{v_l}^{-}(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)\}, \tag{36}$$

for $h_{j-1}+1 \leq l \leq h_j$. It follows from (6) to (10) and the max-log approximation of (35) that the state and composite branch metrics in (36) are given below:

$$\log \alpha_{h_j}(\sigma) = \max_{\sigma' \in \Omega_{h_{j-1}}(\sigma)} \{\log \gamma_{h_j}(L(\sigma',\sigma)) + \log \alpha_{h_{j-1}}(\sigma')\}, \tag{37}$$

$$\log \beta_{h_{j-1}}(\sigma') = \max_{\sigma \in \Omega_{h_j}(\sigma')} \{\log \gamma_{h_j}(L(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)\}, \tag{38}$$

$$\log \gamma_{h_j}(L(\sigma',\sigma)) \triangleq \max_{\mathbf{b}(\sigma',\sigma) \in L(\sigma',\sigma)} \{\log \gamma_{h_j}(\mathbf{b}(\sigma',\sigma))\}, \tag{39}$$

$$\log \gamma_{h_j}(L_{v_l}^{+}(\sigma',\sigma)) \triangleq \max_{\substack{\mathbf{b}(\sigma',\sigma) \in L(\sigma',\sigma) \\ v_l=+1}} \{\log \gamma_{h_j}(\mathbf{b}(\sigma',\sigma))\}, \tag{40}$$

$$\log \gamma_{h_j}(L_{v_l}^{-}(\sigma',\sigma)) \triangleq \max_{\substack{\mathbf{b}(\sigma',\sigma) \in L(\sigma',\sigma) \\ v_l=-1}} \{\log \gamma_{h_j}(\mathbf{b}(\sigma',\sigma))\}, \tag{41}$$

for $h_{j-1}+1 \leq l \leq h_j$.

The metrics, $\log \alpha_{h_j}(\sigma)$ and $\log \beta_{h_j}(\sigma)$, are simply the forward and backward metrics of state $\sigma$, respectively, and they can be computed recursively with initial conditions, $\log \alpha_{h_0}(\sigma_0) = 0$ and $\log \beta_{h_\nu}(\sigma_f) = 0$. The metric $\log \gamma_{h_j}(L(\sigma',\sigma))$ is simply the metric of the composite branch $L(\sigma',\sigma)$. The sum

$$\log \alpha_{h_{j-1}}(\sigma') + \log \gamma_{h_j}(L(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)$$

evaluated from (37) to (39) represents the metric of a path in $T(U)$ that passes through state $\sigma'$ at time-$h_{j-1}$, state $\sigma$ at time-$h_j$ and a branch in $L(\sigma',\sigma)$.

For the AWGN with BPSK transmission, we use (15) to compute $\log \gamma_{h_j}(\mathbf{b}(\sigma',\sigma))$. Since $N_0$ and $(\frac{1}{\pi N_0})^{m_j/2}$ are constants in $T_j$, we can use

$$\log \gamma_{h_j}(\mathbf{b}(\sigma',\sigma)) = \sum_{l=h_{j-1}+1}^{h_j} r_l v_l, \tag{42}$$

as the metric of branch $\mathbf{b}(\sigma', \sigma)$ in computing $\log \gamma_{h_j}(L(\sigma', \sigma))$'s.

Similar to the MAP algorithm based on a sectionalized trellis $T(U)$, the Max-Log-MAP algorithm can also be carried out in two stages. At first stage, the parallel branches of each distinct composite branch $L(\sigma', \sigma)$ are preprocessed to obtain the composite branch metric $\log \gamma_{h_j}(L(\sigma', \sigma))$ and the bit metrics, $\log \gamma_{h_j}(L^+_{v_l}(\sigma', \sigma))$ and $\log \gamma_{h_j}(L^-_{v_l}(\sigma', \sigma))$ for $h_{j-1} + 1 \leq l \leq h_j$. These metrics are stored in a table, called the *branch metric table* for the trellis section $T_j$ from time-$h_{j-1}$ to time-$h_j$. Let $\mathbf{b}^*(\sigma', \sigma) \triangleq (v^*_{h_{j-1}+1}, v^*_{h_{j-1}+2}, \cdots, v^*_{h_j})$ be the branch that has the largest branch metric among parallel branches in $L(\sigma', \sigma)$. From (39), we have

$$\log \gamma(L(\sigma', \sigma)) = \log \gamma(\mathbf{b}^*(\sigma', \sigma)). \tag{43}$$

It follows from (39) to (41) and (43) that

$$\log \gamma_{h_j}(L(\sigma', \sigma)) = \max\{\log \gamma_{h_j}(L^+_{v_l}(\sigma', \sigma)), \log \gamma_{h_j}(L^-_{v_l}(\sigma', \sigma))\} \tag{44}$$

$$= \begin{cases} \log \gamma_{h_j}(L^+_{v_l}(\sigma', \sigma)), & \text{if } v^*_l = 1, \\ \log \gamma_{h_j}(L^-_{v_l}(\sigma', \sigma)), & \text{if } v^*_l = -1, \end{cases} \tag{45}$$

for $h_{j-1} + 1 \leq l \leq h_j$. Based on (45), we can construct the branch metric table for the trellis section $T_j$ using the following procedure A:

(1) Compute $\log \gamma_{h_j}(L(\sigma', \sigma))$ from (39) for each distinct composite branch in $T_j$.

(2) For each code bit $v_l$ in $T_j$, based on $\mathbf{b}^*(\sigma', \sigma)$ and from (45), we first check whether $v^*_l = 1$ or $-1$ (a logic operation) to determine which one between $\log \gamma_{h_j}(L^+_{v_l}(\sigma', \sigma))$ and $\log \gamma_{h_j}(L^-_{v_l}(\sigma', \sigma))$ is equal to $\log \gamma_{h_j}(L(\sigma', \sigma))$. Then we only need to compute the one between $\log \gamma_{h_j}(L^+_{v_l}(\sigma', \sigma))$ and $\log \gamma_{h_j}(L^-_{v_l}(\sigma', \sigma))$ that is not equal to $\log \gamma_{h_j}(L(\sigma', \sigma))$.

At the second stage of the Max-Log-MAP decoding, the state metrics $\log \alpha_{h_j}(\sigma)$ and $\log \beta_{h_{j-1}}(\sigma')$ are computed from (37) and (38) recursively with initial conditions, $\log \alpha_{h_0}(\sigma_0) = 0$, and $\log \beta_{h_\nu}(\sigma_f) = 0$. Assuming bi-directional decoding, $\log \alpha_{h_j}(\sigma)$ and $\log \beta_{h_{j-1}}(\sigma')$ are computed simultaneously from both directions of the trellis $T(U)$ along with the construction of the branch metric tables, section by section in serial manner. Once the state metrics, $\log \alpha_{h_{j-1}}(\sigma')$ and $\log \beta_{h_j}(\sigma)$, at the section boundary locations $h_{j-1}$ and $h_j$, have been computed and the branch metric table for trellis section $T_j$ has been constructed, the LLR's of the estimated code bits $v_l$

for $h_{j-1}+1 \le l \le h_j$ can be computed from (36) by executing the *add-compare-select-subtract* (ACSS) process (similar to the Viterbi algorithm).

To compute the LLR $\tilde{L}(\hat{v}_l)$ efficiently, define

$$R^{(j)} \triangleq \max_{(\sigma',\sigma)} \{\log \alpha_{h_{j-1}}(\sigma') + \log \gamma_{h_j}(L(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)\}, \tag{46}$$

$$R^{(j)}_{+1}(v_l) \triangleq \max_{\substack{(\sigma',\sigma) \\ v_l=+1}} \{\log \alpha_{h_{j-1}}(\sigma') + \log \gamma_{h_j}(L^+_{v_l}(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)\}, \tag{47}$$

$$R^{(j)}_{-1}(v_l) \triangleq \max_{\substack{(\sigma',\sigma) \\ v_l=-1}} \{\log \alpha_{h_{j-1}}(\sigma') + \log \gamma_{h_j}(L^+_{v_l}(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)\}\}, \tag{48}$$

for $1 \le j \le \nu$. From (11), (46) to (48) and using the max-log approximation, we readily see that for $h_{j-1}+1 \le l \le h_j$,

$$R^{(j)} = \max\{R^{(j)}_{+}(v_l), R^{(j)}_{-}(v_l)\}. \tag{49}$$

It follows from (45) to (48) that

$$R^{(j)} = \begin{cases} R^{(j)}_{+1}(v_l), & \text{if } v^*_l = 1, \\ R^{(j)}_{-1}(v_l), & \text{if } v^*_l = -1. \end{cases} \tag{50}$$

Then an efficient procedure for computing $R^{(j)}_{+1}(v_l)$, $R^{(j)}_{-1}(v_l)$ and $\tilde{L}(v_l)$, called procedure B, is given below:

(1) Compute $R^{(j)}$ based on (46).

(2) For each code bit $v_l$ in $T_j$, based on $\mathbf{b}^*(\sigma',\sigma)$ and from (50), we first check whether $v^*_l = 1$ or $-1$ to determine which one between $R^{(j)}_{+1}(v_l)$ and $R^{(j)}_{-1}(v_l)$ is equal to $R^{(j)}$. Then, compute the one between $R^{(j)}_{+1}(v_l)$ and $R^{(j)}_{-1}(v_l)$ that is not equal to $R^{(j)}$.

(3) Compute $\tilde{L}(v_l)$ by taking the difference, $R^{(j)}_{+1}(v_l) - R^{(j)}_{-1}(v_l)$, for $h_{j-1}+1 \le l \le h_j$.

## 4.2 Computational complexity and storage requirement

The computational complexity for constructing the branch metric tables for the trellis sections can be analyzed based on procedure A. From (39) and (42), we find that step-1 of procedure A requires $|B^d_j| \cdot |B^p_j| \cdot (m_j - 1)$ additions and $|B^d_j| \cdot (|B^p_j| - 1)$ comparisons to compute the branch

metrics for all the distinct composite branches in the trellis section $T_j$. From (45), (40) and (41), we find that step-2 of procedure A requires logic operations and $|B_j^d| \cdot (|B_j^p|/2-1) \cdot m_j$ comparisons to compute metrics, $\log \gamma_{h_j}(L_{v_l}^+(\sigma', \sigma))$ and $\log \gamma_{h_j}(L_{v_l}^-(\sigma', \sigma))$. Therefore, construction of the branch metric table for the trellis section $T_j$ requires a total of

$$N_c^j(\gamma) = |B_j^d| \cdot (|B_j^p| - 1 + (|B_j^p|/2 - 1) \cdot m_j) \tag{51}$$

comparisons and a total of

$$N_a^j(\gamma) = |B_j^d| \cdot |B_j^p| \cdot (m_j - 1) \tag{52}$$

additions.

From (37) and (38), we find that the computation of metrics of states at the boundaries of $T_j$ requires a total of $N_a^j(\alpha) + N_a^j(\beta)$ additions and $N_c^j(\alpha) + N_c^j(\beta)$ comparisons, where

$$N_c^j(\alpha) = |B_j^c| - |\Sigma_{h_j}(C)|, \tag{53}$$

$$N_c^j(\beta) = |B_j^c| - |\Sigma_{h_{j-1}}(C)|, \tag{54}$$

$$N_a^j(\alpha) = N_a^j(\beta) = |B_j^c|. \tag{55}$$

To analyze the complexity of computing the LLR's given by (36), we follow procedure B. Based on (46), step-1 requires $2|B_j^c|$ additions and $|B_j^c| - 1$ comparisons to compute $R^{(j)}$'s in the trellis section $T_j$. Based on (50), (47) and (48), step (2) requires logic operations, $|B_j^c| \cdot m_j$ additions and $(|B_j^c| - 1) \cdot m_j$ comparisons to compute $R_{+1}^{(j)}(v_l)$'s and $R_{-1}^{(j)}(v_l)$'s. Step-3 requires $m_j$ subtractions (a subtraction operation is assumed to be equivalent to an addition) to compute LLR's in the trellis section $T_j$. Therefore, a total of

$$N_c^j(\tilde{L}) = (|B_j^c| - 1) \cdot (m_j + 1) \tag{56}$$

comparisons and a total of

$$N_a^j(\tilde{L}) = (2 + m_j) \cdot |B_j^c| + m_j \tag{57}$$

additions (including subtractions) are required to compute LLR's in the trellis section $T_j$.

Since a comparison operation has the same complexity as an addition, it is regarded as an addition-equivalent operation. Therefore, to decode a received sequence, the Max-log-MAP

algorithm based on the sectionalized trellis $T(U)$ requires a total of

$$N_{ae}(U) = \sum_{j=0}^{\nu-1}(N_a^j(\gamma) + N_c^j(\gamma) + N_a^j(\tilde{L}) + N_c^j(\tilde{L})) + \sum_{j=1}^{\nu-1}(N_a^j(\alpha) + N_c^j(\alpha)) + \sum_{j=0}^{\nu-2}(N_a^j(\beta) + N_c^j(\beta)) \tag{58}$$

addition-equivalent operations. $N_{ae}(U)$ is used as a measure of the computational complexity of the Max-log-MAP decoding algorithm based on a sectionalized trellis $T(U)$.

The storage requirement for the Max-log-MAP algorithm is the same as that for the MAP algorithm.

Table 3 gives optimum sectionalizations (in terms of minimizing the number of addition-equivalent operations) of trellises for some RM codes with the Max-Log-MAP decoding. For comparison purpose, the computational complexities and storage requirements of these codes based on bit-level trellises are also included. We see that proper sectionalization reduces computational complexity and storage requirement significantly for the Max-Log-MAP algorithm.

## 4.3 Log-MAP algorithm

The Max-Log-MAP algorithm is a suboptimum realization of the MAP algorithm. Even through it gives an error performance very close to that of the MAP algorithm as shown in Figure 1, it produces soft-output values inferior to that of the MAP algorithm, due to the approximation of (35). Hence, when we use the Max-Log-MAP algorithm in turbo decoding, the inferior reliability value (the soft-output of the Max-Log-MAP decoder) results in performance degradation compared with the optimum MAP decoder. Figure 2 shows bit error performances of turbo decoding of the parallel concatenated code with block interleaver of size 256 and the (32,16) RM component codes. We see that at bit error rate of $10^{-3}$, the Max-Log-MAP decoder results in 0.2 to 0.4 dB performance degradation compared with the MAP decoder.

To overcome this problem, the Jacobian logarithm

$$\log(e^{\delta_1} + e^{\delta_2}) = \max\{\delta_1, \delta_2\} + \log(1 + e^{-|\delta_2 - \delta_1|}) = \max\{\delta_1, \delta_2\} + f_c(|\delta_2 - \delta_1|) \tag{59}$$

can be used, where $f_c(\cdot)$ is a correction function [3]. Then, for a finite set of real numbers, $\{\delta_1, \cdots, \delta_q\}$, $\log(e^{\delta_1} + \cdots + e^{\delta_q})$ can be computed recursively. The recursion is initialized with

two terms given by (59). Suppose that $\log(e^{\delta_1} + \cdots + e^{\delta_{i-1}})$ with $1 < i \leq q$ is known. Hence,

$$\begin{aligned} \log(e^{\delta_1} + \cdots + e^{\delta_i}) &= \log(\triangle + e^{\delta_i}) \\ &= \max\{\log \triangle, \delta_i\} + f_c(|\log \triangle - \delta_i|), \end{aligned} \tag{60}$$

with $\triangle = e^{\delta_1} + \cdots + e^{\delta_{i-1}}$. Based on this recursion, we modify the Max-log-MAP algorithm through the use of simple correction functions. This algorithm, called the *Log-MAP algorithm* [3], gives the same error performance as the MAP algorithm, but is easier to implement. Each correction term needs an additional one-dimensional look-up table, and two additions based on (59). Consequently, the Log-MAP algorithm requires only additions and comparisons for computing the LLR's.

The storage requirement for the Log-MAP algorithm is the same as those for the MAP and Max-log-MAP algorithms, assuming the storage of the look-up tables is negligible.

Consider the computational complexity of the Log-MAP algorithm. Since two extra additions are required per comparison to calculate $f_c(\cdot)$ in (59), a total of $N_a^j(\gamma) + 3N_c^j(\gamma)$, $N_a^j(\alpha) + 3N_c^j(\alpha)$, $N_a^j(\beta)+3N_a^j(\beta)$ and $N_a^j(\tilde{L})+3N_c^j(\tilde{L})$ addition-equivalent operations required to compute $\log\gamma$'s, $\log\alpha$'s, $\log\beta$'s and LLR's in $T_j$, respectively, where $N_a^j()$'s and $N_c^j()$'s are the numbers of additions and comparisons evaluated for the Max-Log-MAP algorithm in Section 4.2.

Table 4 gives optimum sectionalizations (in terms of minimizing the number of addition-equivalent operations) of trellises for some RM codes with the Log-MAP decoding. For comparison purpose, the computational complexities and storage requirements of these codes based on bit-level trellises are also included. We also see that proper sectionalization reduces computational complexity and storage requirement for the Log-MAP algorithm.

# 5 Bi-directional and Parallel MAP Decoding

To reduce the decoding delay and speed up the decoding process of the MAP algorithm, a bi-directional MAP algorithm can be devised. This bi-directional MAP algorithm allows to compute the forward and backward recursions simultaneously.

To achieve bi-directional decoding, we can permute the encoded sequence $(v_1, v_2, \cdots, v_n)$ to $(v_1, v_n, v_2, v_{n-1}, \cdots)$ before transmission. Let $\mathbf{r} = (r_1, r_n, r_2, r_{n-1}, \cdots)$ be its corresponding re-

ceived sequence. Based on the interleaved received sequence $\mathbf{r}$, we compute the $\gamma$ probabilities at the boundary locations in the following order: $\gamma_{h_1} \rightarrow \gamma_{h_\nu} \rightarrow \gamma_{h_2} \rightarrow \gamma_{h_{\nu-1}} \rightarrow \cdots$. A permutation circuit permutes the received sequence $\mathbf{r}$ into $\mathbf{r}_1 = (r_1, r_2, \cdots, r_n)$ and $\mathbf{r}_2 = (r_n, r_{n-1}, \cdots, r_1)$ and shift them from both ends of the sectionalized trellis $T(U)$ to perform forward and backward recursions simultaneously and compute the $\alpha$ and $\beta$ probabilities along with computation of $\gamma$'s in both directions. When the recursions meet at the middle of the trellis, the computation of the log-likelihood ratios of the code bits begins based on (12).

Even though the same number of computations is required, this approach roughly doubles the decoding speed (or reduces the decoding time by half). If the code trellis has mirror image symmetry [5], we use two identical circuits to compute $\alpha$ and $\beta$. This also simplifies the IC implementation. Trellises for RM codes and all cyclic codes have mirror symmetry [5]. This bi-directional decoding approach can also be applied to the Log-MAP and Max-Log-MAP algorithms.

An $\nu$-section trellis for a linear block code can be decomposed into parallel and structurally identical subtrellises without cross connections among them [5]. Each subtrellis has much smaller state complexity and connectivity than the full code trellis. This parallel decomposition allows us to devise identical smaller MAP (or Max-Log-MAP) decoders to process all the subtrellises in parallel independently without communication between them. This also simplifies the IC implementation and speeds up the decoding process [8].

Suppose an $\nu$-section trellis for an $(n, k)$ linear block code is decomposed into $Q$ subtrellises, $T(1), T(2), \cdots, T(Q)$. The MAP decoder for the subtrellis $T(q)$, $1 \leq q \leq Q$, finds a pair of values, $G^q_{+1}(v_i)$ and $G^q_{-1}(v_i)$ for each code bit $v_i$ in $j$-th section with $0 \leq i < n$, as follows,

$$G^q_{\pm 1}(v_i) \triangleq \sum_{\substack{(\sigma', \sigma) \in T(q) \\ v_i = \pm 1}} (\alpha_{h_{j-1}}(\sigma') \cdot \gamma_{h_j}(L^{\pm}_{v_i}(\sigma', \sigma)) \cdot \beta_{h_j}(\sigma)). \tag{61}$$

Then we determine the final pair of values of the entire trellis, denoted $(G_{+1}(v_i), G_{-1}(v_i))$ as follows,

$$G_{+1}(v_i) \triangleq \sum_{1 \leq q \leq Q} G^q_{+1}(v_i) \text{ and } G_{-1}(v_i) \triangleq \sum_{1 \leq q \leq Q} G^q_{-1}(v_i). \tag{62}$$

Finally, the log-likelihood ratio of $v_i$ is given by

$$L(\hat{v}_i) = \log \frac{G_{+1}(v_i)}{G_{-1}(v_i)}. \tag{63}$$

Similarly, the Max-log-MAP decoding can also be performed in parallel. The Max-log-MAP decoder for $T(q)$ obtains a pair of surviving metrics, $M^q_{+1}(v_i)$ and $M^q_{-1}(v_i)$, for $v_i$ which are given by

$$M^q_{\pm 1}(v_i) = \max_{\substack{(\sigma',\sigma)\in T(q) \\ v_i=\pm 1}} \{\log \alpha_{h_{j-1}}(\sigma') + \log \gamma_{h_j}(L^{\pm}_{v_i}(\sigma',\sigma)) + \log \beta_{h_j}(\sigma)\}. \tag{64}$$

Then the final surviving pair of the entire trellis, $(M_{+1}(v_i),\ M_{-1}(v_i))$ are given by

$$M_{+1}(v_i) = \max\{M^q_{+1}(v_i) : 1 \le q \le Q\} \text{ and } M_{-1}(v_i) = \max\{M^q_{-1}(v_i) : 1 \le q \le Q\}. \tag{65}$$

Finally, the log-likelihood ratio of $v_i$ is given by the difference, $M_{+1}(v_i) - M_{-1}(v_i)$.

From the stand-point of speed, the effective computational complexity of decoding a received sequence is defined as the computational complexity of a single parallel subtrellis plus the cost of the final summations (or comparisons) among the survivors generated by each of the subtrellis decoders. The time required for final summation (or comparison) is generally small relative to the time required for processing a subtrellis. Furthermore, since all the subtrellises are processed in parallel, the speed of decoding is therefore limited only by the time required to process one subtrellis. Consequently, this approach not only simplifies the decoding complexity but also gains speed. For example, the 4-section trellis diagram of the (32,16) RM code is shown in Figure 3. It consists of 8 parallel and structurally identical subtrellises without communication between them. From Table 5, we observe that a single subtrellis has much less computational complexity than the entire trellis with the Max-log-MAP decoding. The system level architecture utilizes 8 low complexity Max-Log-MAP decoders to decode 8 subtrellises in parallel. Furthermore, the parallel subdecoders are identical, so that the IC implementation is easy. Also it is clear that both bi-directional and parallel decoding processes can be incorporated in a single decoder.

# 6 Conclusion

In this paper, we have modified the MAP, Log-MAP and Max-Log-MAP algorithms for decoding linear block codes based on sectionalized trellises in order to reduce both the decoding complexity and delay. Example results show that properly sectionalized trellises require less real operations and less memory than the bit level trellis. Furthermore, we have taken advantage of the trellis structure of linear block codes, such as mirror image symmetry and parallel structure to perform

bi-directional and parallel decodings. The bi-directional decoding is a simple approach to reduce the decoding delay by a factor close to two. The parallel decoding uses identical subdecoders to process subtrellises in parallel. This approach reduces the number of computations required for each subdecoder and speeds up the decoding process.

# References


[1] L. R. Bahl. J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," IEEE Trans. on Information Theory, 20, 2, pp. 284-287 (1974).

[2] C. Berrou, A. Glavieux and P. Thitimajshima, " Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," Proc. of IEEE Intl. Conf. on Communications, Geneva, Switzerland, pp. 1064-1070 (May 1993).

[3] P. Robertson, E. Villebrun, P. Hoeher, "A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms Operating in the Log Domain," in Proc. ICC'95, Seattle, WA, 1995. pp. 1009-1013.

[4] J. Hagenauer, E. Offer and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," IEEE Trans. on Information Theory, 42, 2, pp. 429-445 (Mar. 1996).

[5] S. Lin, T. Kasami, T. Fujiwara and M. Fossorier, Trellis and Trellis-based Decoding Algorithms, Kluwer Academic Publishers, Boston, MA., 1998.

[6] A. Vardy, Trellis Structure of Codes, Handbook of Coding Theory, (edited by V. S. Pless, W. C. Huffman, and R. A. Brualdi), Elsevier Science Publishers, 1998.

[7] A. Lafourcade and A. Vardy, "Optimal Sectionalization of a Trellis," IEEE Trans. on Information Theory, 42,3, pp. 689-703 (May 1996).

[8] H. T. Moorthy, S. Lin and G. T. Uehara, "Good Trellises for IC Implementation of Viterbi Decoders for Linear Block Codes," IEEE Trans. on Communications, 45, 1, pp. 52-63 (Jan. 1997).

Table 1: Optimum trellis sectionalizations of some RM codes for MAP decoding

| Codes | Bit-level trellis | | | Optimum sectionalization | | | |
|---|---|---|---|---|---|---|---|
| | mult. | add. | memory | boundary location | mult. | add. | memory |
| RM(8,4) | 180 | 66 | 55 | {0,2,4,6,8} | 112 | 86 | 33 |
| RM(16,5) | 700 | 218 | 195 | {0,4,8,12,16} | 288 | 334 | 65 |
| RM(16,11) | 1,020 | 458 | 195 | {0,2,4,6,8,10,12,14,16} | 712 | 574 | 109 |
| RM(32,6) | 2,764 | 778 | 731 | {0,4,8,16,24,28,32} | 784 | 1,118 | 153 |
| RM(32,16) | 25,612 | 9,594 | 4,891 | {0,2,4,8,12,16,20,24,28, 30,32} | 9,704 | 12,846 | 885 |
| RM(32,26) | 4,748 | 2,266 | 731 | {0,2,4,6,7,8,10,11,12, 13,14,16,18,19,20,21, 22,24,25,26,28,30,32} | 4,064 | 2,486 | 549 |
| RM(64,7) | 10,988 | 2,922 | 2,827 | {0,4,8,16,24,32,40,48, 56,60,64} | 2,288 | 3,710 | 457 |
| RM(64,22) | 1,500,204 | 475,386 | 325,051 | {0,2,4,8,16,24,32,40,48, 56,60,62,64} | 285,256 | 664,686 | 20,629 |
| RM(64,42) | 2,197,548 | 998,394 | 325,051 | {0,2,4,8,12,14,16,20,22, 24,26,28,32,36,38,40,42, 44,48,50,52,56,60,62,64} | 1,395,352 | 1,395,352 | 124,453 |
| RM(64,57) | 20,396 | 9,978 | 2,827 | {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} | 18,952 | 10,398 | 2,453 |

Table 2: Best uniform sectionalizations of the MAP algorithm for Reed-Muller Codes

| Codes | trellis | multiplication |
|---|---|---|
| RM(8,4) | 2-section* | 112 |
| RM(16,5) | 4-section* | 288 |
| RM(16,11) | 8-section* | 712 |
| RM(32,6) | 4-section | 832 |
| RM(32,16) | 8-section | 9,728 |
| RM(32,26) | 8-section | 4,088 |
| RM(64,7) | 8-section | 2,336 |
| RM(64,22) | 8-section | 286,528 |
| RM(64,42) | 16-section | 1,395,616 |
| RM(64,57) | 32-section | 19,032 |

Table 3: Optimum trellis sectionalizations of some RM codes for Max-Log-MAP decoding

| Codes | Bit-level trellis | | Optimum sectionalization | | |
|---|---|---|---|---|---|
| | operation | memory | boundary location | operation | memory |
| RM(8,4) | 230 | 55 | {0,,4,8} | 156 | 81 |
| RM(16,5) | 886 | 195 | {0,2,4,8,12,14,16} | 486 | 77 |
| RM(16,11) | 1,446 | 195 | {0,2,4,6,8,10,12,14,16} | 1,222 | 109 |
| RM(32,6) | 3,478 | 731 | {0,2,4,8,16,24,28,30,32} | 1,510 | 165 |
| RM(32,16) | 35,142 | 4,891 | {0,1,3,5,8,12,16,20,24,27,29,31,32} | 22,078 | 919 |
| RM(32,26) | 6,950 | 731 | {0,2,4,6,7,8,10,11,12,13,14,16, 18,19,20,21,22,24,25,26,28,30,32} | 6,446 | 549 |
| RM(64,7) | 13,782 | 2,827 | {0,2,4,8,16,24,32,40,48,56,60,62,64} | 5,094 | 469 |
| RM(64,22) | 1,975,462 | 325,051 | {0,1,3,5,8,10,16,18,24,30, 32,34,40,46,48,54,56,59,61,63,64} | 905,974 | 37,839 |
| RM(64,42) | 3,195,814 | 325,051 | {0,2,4,8,10,12,14,16,18,20,22,24,26, 28,30,32,34,36,38,40,42,44,46,48,50,52,54, 56,58,60,62,64} | 2,646,566 | 141,925 |
| RM(64,57) | 30,246 | 2,827 | {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} | 29,174 | 2,453 |

Table 4: Optimum trellis sectionalizations of some RM codes for Log-MAP decoding

| Codes | Bit-level trellis | | Optimum sectionalization | | |
|---|---|---|---|---|---|
| | operation | memory | boundary location | operation | memory |
| RM(8,4) | 330 | 55 | {0,4,8} | 244 | 81 |
| RM(16,5) | 1,258 | 195 | {0,1,2,4,8,12,14,15,16} | 866 | 83 |
| RM(16,11) | 2,298 | 195 | {0,1,2,3,4,6,8,10,12,13,14,15,16} | 2,242 | 115 |
| RM(32,6) | 4,906 | 731 | {0,1,2,4,8,16,24,28,30,31,32} | 2,946 | 171 |
| RM(32,16) | 54,202 | 4,891 | {0,1,2,3,5,8,9,12,15,16,17, 20,23,24,27,29,30,31,32} | 43,906 | 1,415 |
| RM(32,26) | 11,354 | 731 | {0,1,2,3,4,6,7,8,10,11,12,13,14,16, 18,19,20,21,22,24,25,26,28,29,30, 31,32} | 11,210 | 555 |
| RM(64,7) | 19,370 | 2,827 | {0,1,2,4,8,16,24,32,40,48,56,60, 62,63,64} | 10,690 | 475 |
| RM(64,22) | 2,925,978 | 325,051 | {0,1,2,3,5,8,9,12,16,17,20,24,28, 31,32,33,36,40,44,47,48,52,55,56, 59,61,62,63,64} | 1,933,682 | 63,819 |
| RM(64,42) | 5,192,346 | 325,051 | {0,1,2,3,4,6,8,10,12,14,16,18,20,22,24,26, 28,30,32,34,36,38,40,42,44,46,48,50,52, 54,56,58,60,61,62,63,64} | 5,009,346 | 141,931 |
| RM(64,57) | 49,946 | 2,827 | {0,1,2,3,4,6,7,8,10,11,12,13,14,15,16,18, 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, 48,49,50,51,52,53,54,56,57,58,60,61,62, 63,64} | 49,618 | 2,459 |

Table 5: Complexities for the Max-Log-MAP decoding of the (32,16) RM code with 4-section trellis

| operations | entire trellis | subtrellis |
|---|---|---|
| memory | 2,690 | 340 |
| operations | 30,842 | 3,865 |

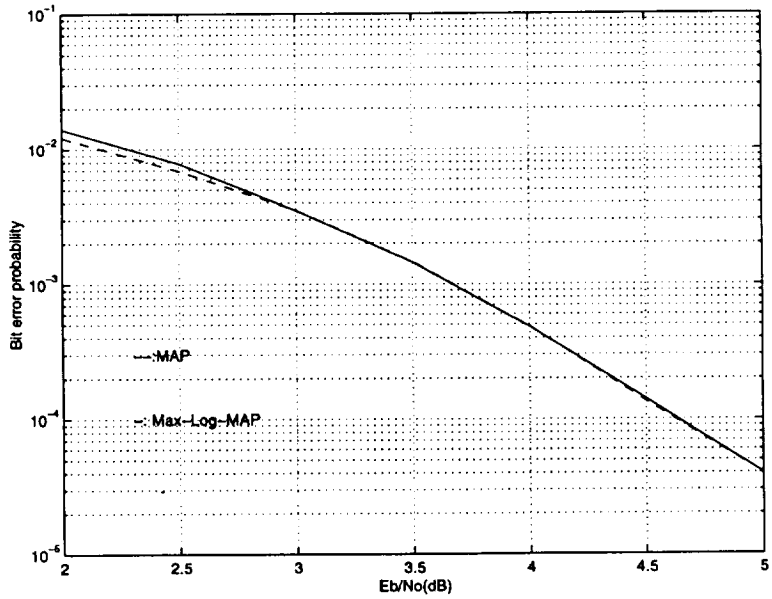


Figure 1: Bit-error performances of the (32,16) RM code with MAP and Max-Log-MAP decoding algorithms

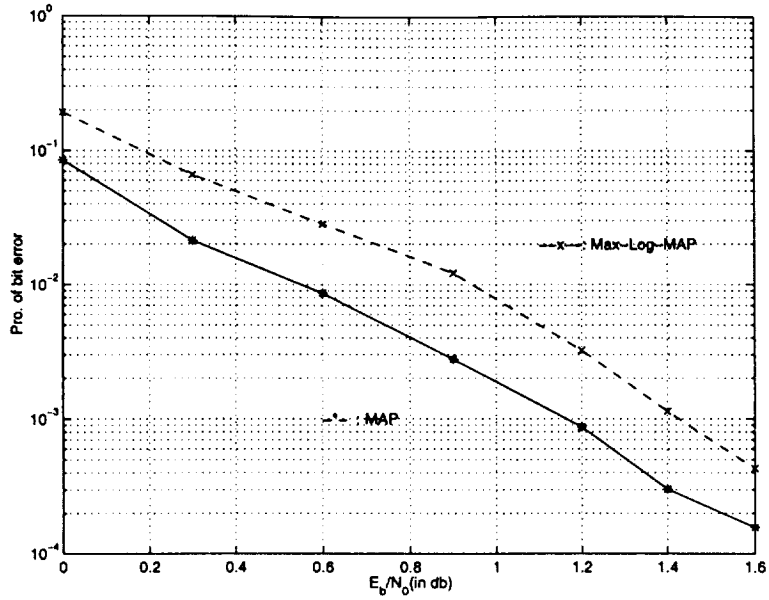


Figure 2: Bit-error performances of turbo decoding of the (32,16) RM code with MAP and Max-Log-MAP decoding algorithms

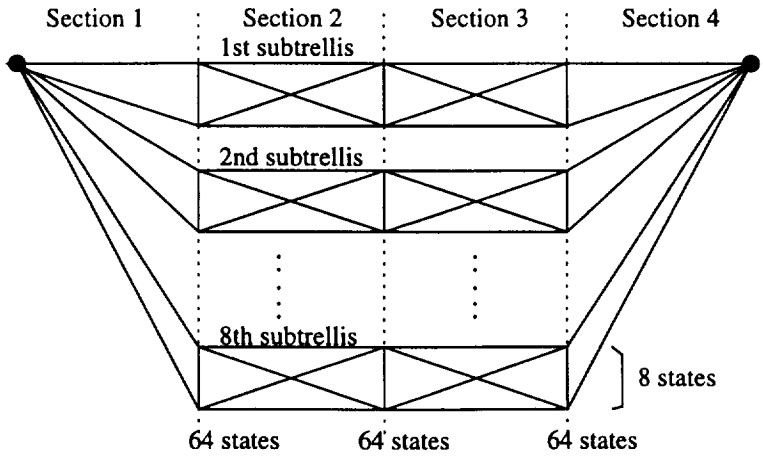


Figure 3: 4-section trellis diagram of the (32,16) RM code