Trellis Computations
Axel Heim, Vladimir Sidorenko, Uli Sorger

Abstract—For a certain class of functions, the distribution of the function values can be calculated in the trellis or a sub-trellis. The forward/backward recursion known from the BCJR algorithm [1] is generalized to compute the moments of these distributions. In analogy to the symbol probabilities, by introducing a constraint at a certain depth in the trellis we obtain symbol moments. These moments are required for an efficient implementation of the discriminated belief propagation algorithm in [2], and can furthermore be utilized to compute conditional entropies in the trellis.

The moment computation algorithm has the same asymptotic complexity as the BCJR algorithm. It is applicable to any commutative semi-ring, thus actually providing a generalization of the Viterbi algorithm [3].

Index Terms—Trellis Algorithms, Viterbi Algorithm, BCJR Algorithm, Distributions, Moments, Decoding, Complexity

I. INTRODUCTION

Trellises were introduced into the coding theory literature by Forney [4] as a means of describing the Viterbi algorithm for decoding convolutional codes. Bahl et al. [1] showed that block codes can also be described by a trellis, and Wolf [5] proposed the use of the Viterbi algorithm for trellis-based soft-decision decoding of block codes. Massey [6] gave a graph-theoretic definition of a block trellis and an alternative construction of minimal trellises. Forney’s paper [7] showed that group codes, including linear codes and lattices, have a construction of minimal trellises. Forney [4] as a means of describing the Viterbi algorithm for decoding convolutional codes. Bahl et al. [1] showed that block codes can also be described by a trellis, and Wolf [5] proposed the use of the Viterbi algorithm for trellis-based soft-decision decoding of block codes. Massey [6] gave a graph-theoretic definition of a block trellis and an alternative construction of minimal trellises. Forney’s paper [7] showed that group codes, including linear codes and lattices, have a construction of minimal trellises.

In [8], McEliece investigated the complexity of a generalized Viterbi algorithm which allows efficient computation of flows on a code trellis. These results were further generalized in [9] and [10]. However, the calculation of flows does not fully exploit the capabilities of the trellis (representation): For a certain set of functions it is possible to calculate the moments of these functions in the trellis. These can be scalar or vectorial, as long as they are linear and fulfill a separability criterion.

For iterative decoding of coupled codes, the popular sum-product algorithm is used to calculate the symbol probabilities of the component codes. These probabilities are exchanged between component decoders until a stable solution is found. This iterative algorithm works very well for long “turbo”, low-density parity check (LDPC) and some other codes, obtained by concatenation of simple component codes in a special way. However, performance becomes poor when utilizing short or some good component codes.

Recently, Sorger [2] showed that iterative decoding is improved when discriminating code words $c$ by their correlation $crT$ or $cwT$ with the received word $r$ or a ‘believed’ word $w$, respectively. Not only symbol probabilities are considered, but also the distribution of these probabilities over the correlation value. An efficient algorithm is introduced using the first two moments to approximate these distributions.

In this paper we propose algorithms to compute both such distributions and their moments in the trellis.

Example 1: Consider Figure 1 which shows two distributions of the correlation function $crT$, where $c$ is a code word and $r$ is the noisy version of a code word $\hat{c} \in C$ after transmission over a memory-less binary symmetric channel (BSC). The curves show the distributions for $c \in C_i(+1)$ and $c \in C_i(-1)$, respectively, where $C_i(x) := \{c \in C : c_i = x\}$ denotes the sub-code of $C$ for which the symbol $c_i$ at a given position $i$ of each code word equals $x \in \{-1,+1\}$. The integrals over the distributions equal the symbol probabilities $P(c_i = x | r)$. However, the probability ratio

$$P(crT, c_i = +1 | r)$$

$$P(crT, c_i = -1 | r)$$

(1)

varies significantly over $crT$ which can be exploited when knowledge on the correlation $\hat{c}rT$ with the transmitted code word is available.

The distributions in Figure 1 can be approximated with their moments

$$E_C[(crT)^m | r, c_i] := \sum_{c \in C} (crT)^m \cdot P(c | r, c_i)$$

(2)

up to a certain order $m$, where $E_C[.]$ is the expectation over all code words $c \in C$. The distributions will be GAUSSIAN for sufficiently long codes which can be understood by the law of large numbers. Hence we can expect the first two moments to suffice for a good approximation.

We present generalizations of the methods in [8] which enable us to compute distributions $P(cwT, c_i = +1 | r)$ and expressions like $E_C[(cwT)^m | r, c_i]$ for some word $w$, whereof
(2) is a special case, both for hard and soft decision. The complexity of the algorithm is of the same order as the classically used BCJR algorithm.

The remainder of this paper is structured as follows. The next section contains a review of common terminology in the context of trellises. This is extended in Section III, which deals with the computation of distributions and their moments in a more general frame. In Section IV we will return to the original problem by transferring the results of Section III to linear block codes and calculate the conditional entropy in the trellis.

II. DEFINITIONS

We deliberately follow to a wide extent the notation and style of McEIliee. The first paragraph is an excerpt from [8] with minor modifications. 1

A trellis \( T = (\mathcal{V}, \mathcal{E}) \) of rank \( n \) is a finite-directed graph with vertex set \( \mathcal{V} \) and edge set \( \mathcal{E} \), in which every vertex is assigned a depth in the range \( \{0, 1, \ldots, n\} \). Each edge is connecting a vertex at depth \( i \) to one at depth \( i+1 \), for some \( i \in \{0, 1, \ldots, n\} \). Multiple edges between vertices are allowed. The set of vertices at depth \( i \) is denoted \( \mathcal{V}_i \), so that \( \mathcal{V} = \bigcup_{i=0}^{n} \mathcal{V}_i \). For \( v \in \mathcal{V}_i \) we write \( \text{depth}(v) = i \). The set of edges connecting vertices at depth \( i \) to those at depth \( i+1 \) is denoted \( \mathcal{E}_{i,i+1} \), so that \( \mathcal{E} = \bigcup_{i=0}^{n-1} \mathcal{E}_{i,i+1} \). There is only one vertex at depth 0, called \( A \), and only one at depth \( n \), called \( B \). If \( e \in \mathcal{E} \) is a directed edge connecting the vertices \( u \) and \( v \), which we denote by \( e : u \rightarrow v \), we call \( e \) the initial vertex, and \( v \) the final vertex of \( e \) and write \( \text{init}(e) = u \), \( \text{fin}(e) = v \). We denote the number of edges leaving a vertex \( v \) by \( \rho^-(v) \), and the number of edges entering a vertex \( v \) by \( \rho^+(v) \), i.e.

\[
\rho^+(v) = \{ e : \text{init}(e) = v \} \\
\rho^-(v) = \{ e : \text{fin}(e) = v \}.
\]

If \( u \) and \( v \) are vertices, a path \( P \) of length \( L \) from \( u \) to \( v \) is a sequence of \( L \) edges: \( P = e_1e_2 \cdots e_L \), such that \( \text{init}(e_1) = u \), \( \text{fin}(e_L) = v \), and \( \text{fin}(e_i) = \text{init}(e_{i+1}) \), for \( i = 1, 2, \ldots, L-1 \). If \( P \) is such a path, we sometimes write \( P : u \rightarrow v \) for short, as well as \( \text{init}(P) = \text{init}(e_1) \) and \( \text{fin}(P) = \text{fin}(e_L) \). We denote the set of paths from vertices at depth \( i \) to vertices at depth \( j \) by \( \mathcal{E}_{i,j} \). We assume that for every vertex \( v \), there is at least one path from \( A \) to \( v \), and at least one path from \( v \) to \( B \).

**Example 2 (Trellis):** Figure 2 shows a trellis of rank \( n = 4 \) with edge set \( \mathcal{E} = \{a, b, c, d, e, f, g, h, i, j, k, l\} \) and vertex set \( \mathcal{V} = \{A, 1, 2, 3, 4, 5, 6, B\} \). There are eight paths \( P : A \rightarrow B \) from \( A \) to \( B \). There is \( \rho^-(1) = 1 \) edge entering (edge \( a \)) and \( \rho^+(1) = 2 \) edges (edges \( c \) and \( d \)) leaving vertex \( v = 1 \). We assume each edge in the trellis is labeled. Let \( T = (\mathcal{V}, \mathcal{E}) \) be a trellis of rank \( n \), such that each edge \( e \in \mathcal{E} \) is labeled with a real valued number \( \lambda(e) \in \mathbb{R} \). We now define the label of a path, and the flow between two vertices.

**Definition 1 (Path Labels):** The label \( \lambda(P) \) of a path \( P = e_1 e_2 \cdots e_L \) is defined as the product \( \lambda(P) = \lambda(e_1) \cdot \lambda(e_2) \cdots \lambda(e_L) \) of the labels of all edges in the path. (Note that

the subscript indicates the sequence number rather than the edge’s depth.)

**Definition 2 (Flow):** If \( u \) and \( v \) are vertices in a labeled trellis, we define the flow \( \eta(u, v) \) from \( u \) to \( v \) to be the sum of the labels on all paths from \( u \) to \( v \), i.e.,

\[
\eta(u, v) = \sum_{P : u \rightarrow v} \lambda(P).
\]

In this paper, we only consider operations on the set of real numbers with ordinary addition and multiplication as the authors are not aware of application for other algebraic structures. However, Appendix C briefly shows that the algorithm can be transferred to any commutative semi-ring, thus leading to a generalization of the Viterbi algorithm [3].

**Example 3:** We continue Example 2. The trellis depicted in Figure 2 is the trellis of the \((4, 3, 2)\) single parity check code. In the BCJR algorithm, the edge labels \( \lambda(e) \) are the channel probabilities of the corresponding transitions.

III. TRELLIS-BASED COMPUTATIONS

In this section we consider distributions of the type

\[
D: q \mapsto D(q) = \sum_{P : A \rightarrow B} \frac{\lambda(P)}{f(q)}
\]

for special functions \( f \), i.e., \( q \) is mapped to the sum of the labels of all paths \( P \) with \( f(P) = q \). We present an algorithm to calculate these distributions over all paths of a trellis or a sub-set of these. Before, however, we develop algorithms to calculate the moments

\[
\bar{\theta}^{(m)}(T) := \frac{\sum_{P : A \rightarrow B} (f(P))^m \cdot \lambda(P)}{\sum_{P : A \rightarrow B} \lambda(P)}
\]

and by introducing a constraint on the paths - the symbol moments

\[
\bar{\Omega}^{(m)}_{s}(T,x) := \frac{\sum_{P : A \rightarrow B} (f(P))^m \cdot \lambda(P)}{\sum_{P : A \rightarrow B} \lambda(P)}
\]

of such distributions in the trellis. We show that the complexity of the moment calculation algorithm is \( O(|\mathcal{E}|) \), where \( |\mathcal{E}| \) is the number of edges in the trellis.

To each edge \( e \in \mathcal{E} \) of the trellis \( T \) we introduce a second label \( c(e) \in \mathbb{R} \), which we will refer to as the c-label. For distinction, we will call \( \lambda(e) \) the \( \lambda \)-label.
Example 4: We continue Example 3. Solid lines correspond to the $c$-label $c(e) = 1$, dashed lines correspond to $c(e) = -1$ (bipolar binary notation). E.g., the path $P = \text{addix}$ has the $c$-label $c(P) = +1 - 1 - 1 + 1$ which is a code word. Let

$$g_i(c(e)) : x \mapsto y; x, y \in \mathbb{R}$$

be a common function of $c(e)$ for all edges $e \in E_{i-1,i}$. Further, let

$$f(c(P)) = f(c(e_1), c(e_2), \ldots, c(e_L)) : c \mapsto y; c(e_i), y \in \mathbb{R}$$

be a function of the $c$-labels of the edges of a path $P$ with length $L$. The bold letter indicates that $c$ is a vector. For simplicity, in the following we will abbreviate $g_i(c(e))$ and $f(c(P))$ by $g_i(e)$ and $f(P)$, respectively. The functions $f(P)$ have to fulfill the linearity criterion

$$f(P) = f(e_1 e_2 \cdots e_n) = g_i(e_1) + g_i(e_2) + \cdots + g_i(e_n)$$

for all paths $P : A \rightarrow B$.

**Definition 3 (Forward Numerator):** We define the $m$-th forward numerator of a function $f$ at vertex $v$ of a trellis $T$ as

$$\alpha^{(m)}(v) := \sum_{P: A \rightarrow v} (f(P))^m \cdot \lambda(P)$$

with initial values

$$\alpha^{(m)}(A) := \begin{cases} 1 : & m = 0 \\ 0 : & m > 0 \end{cases}$$

**Theorem 1 (Forward Recursion):** The $m$-th forward numerator $\alpha^{(m)}(v)$ of a vertex $v \in V_i$ on depth $i$ can be recursively calculated on a trellis $T$ by

$$\alpha^{(m)}(v) = \sum_{e: \text{fin}(e) = v} \lambda(e) \cdot \sum_{i=0}^{m} \binom{m}{i} (g_i(e))^i \cdot \alpha^{(m-i)}(\text{init}(e))$$

as in Algorithm 1.

**Algorithm 1 Computation of first $(M+1)$ Forward Numerators**

01: /* initialization */
02: $\alpha^{(0)}(A) = 1$;
03: for (m=1 to m_max) {
04: \hspace{1em} $\alpha^{(m)}(A) = 0$;
05: \hspace{1em} /* recursion */
06: \hspace{2em} for (i=1 to n) {
07: \hspace{3em} for (v \in \mathbb{V}_i) {
08: \hspace{4em} for (m=0 to m_max) {
09: \hspace{5em} $\alpha^{(m)}(v) = \sum_{e: \text{fin}(e) = v} \lambda(e) \cdot \sum_{i=0}^{m} \binom{m}{i} (g_i(e))^i \cdot \alpha^{(m-i)}(\text{init}(e))$;
10: \hspace{4em} }
11: \hspace{3em} }
12: \hspace{2em} }
13: }
14: }

**Proof:** The proof is by induction on $\text{depth}(v)$. For $\text{depth}(v) = 1$, it follows from the definition of a trellis that all paths from $A$ to $v$ must consist of just one edge $e$, with $\text{init}(e) = A$ and $\text{fin}(e) = v$. Thus the true value of $\alpha^{(m)}(v)$ is the sum of the $\lambda$-labels on all edges $e$ joining $A$ to $v$, weighted by $(g_i(e))^m$. On the other hand, when the algorithm computes $\alpha^{(m)}(v)$ on line 9, the value it assigns to it is (because of the initialization $\alpha^{(0)}(A) = 1$, $\alpha^{(m)}(v) = 0$ for $m > 0$)

$$\alpha^{(m)}(v) = \sum_{e: \text{fin}(e) = v} \lambda(e) \cdot \sum_{i=0}^{m} \binom{m}{i} (g_i(e))^i \cdot \alpha^{(m-i)}(\text{init}(e))$$

which is, as required, the sum of the labels on all edges $e$ joining $A$ to $v$, weighted by $(g_i(e))^m$. Thus the algorithm works correctly for all vertices $v$ with depth($v$) = 1 and any $m \geq 0$.

Assuming now that the assertion is true for all vertices at depth $i$ or less and all $m \leq M$, a vertex $v$ at depth $i+1$ is considered. When the algorithm computes $\alpha^{(m)}(v)$ on line 9, the value it assigns to it is

$$\alpha^{(m)}(v) = \sum_{e: \text{fin}(e) = v} \lambda(e) \cdot \sum_{i=0}^{m} \binom{m}{i} (g_i(e))^i \cdot \alpha^{(m-i)}(\text{init}(e)) \cdot \lambda(P) \cdot (f(P))^m \cdot \lambda(P) \cdot (f(P))^m$$

(6)

But depth($\text{init}(e)$) = $i$ and so by the induction hypothesis

$$\alpha^{(m)}(\text{init}(e)) = \sum_{P: A \rightarrow \text{init}(e)} \lambda(P) \cdot (f(P))^m.$$ (7)

Combining (6) and (7), we have

$$\alpha^{(m)}(v) = \sum_{e: \text{fin}(e) = v} \lambda(e) \cdot \sum_{i=0}^{m} \binom{m}{i} (g_i(e))^i \cdot \alpha^{(m-i)}(\text{init}(e)) \cdot \lambda(P) \cdot (f(P))^m \cdot \lambda(P) \cdot (f(P))^m.$$ (8)

Using the binomial theorem we obtain

$$\alpha^{(m)}(v) = \sum_{e: \text{fin}(e) = v} \sum_{P: A \rightarrow \text{init}(e)} \lambda(P) \cdot (f(P) + g_i(e))^m.$$ (9)

But every path from $A$ to $v$ must be of the form $P \cdot e$, where $P$ is a path from $A$ to a vertex $u$ with depth($u$) = $i$, $\text{init}(e) = u$ and $\text{fin}(e) = v$. Thus by (8), $\alpha^{(m)}(v)$ is correctly calculated by the algorithm.

**Remark 1 (Flow):** $\alpha^{(0)}(v)$ in (4) is the flow $\eta(A,v)$ from $A$ to $v$ (cf. Definition 2) as it is calculated by the BCJR algorithm.

**Remark 2:** $f$ and $g_i$ do not necessarily have to be scalars. Theorem 1 holds for all separable linear functions $f$ fulfilling Equation (3).

**Theorem 2 (Complexity):** The proposed moment computing algorithm requires $O(|E|)$ arithmetic operations, i.e. multiplications and additions.

**Proof:** The calculation of the powers of $g_i(e)$ up to a maximum moment $M$ for all edges $e \in E$ requires $|E| \cdot \max(M - 1, 0)$ multiplications and no additions. We do not consider the operations needed for calculating $g_i(e)$ here. The execution of the sum term over $i$ in line 9 of the algorithm requires $m$ additions, $2m + 1$ multiplications for $m > 0$ and no multiplications for $m = 0$. Therefore line 9 requires

$$\rho^{-1}(v) \cdot [1 + 2m + 1] = \rho^{-1}(v) \cdot 2(m + 1)$$

2For $l = 0$, $(g_i(e))^l = 1$ and thus only one multiplication is necessary.
multiplications for \( m > 0 \), \( \rho^-(v) \) multiplications for \( m = 0 \), and \( \rho^-(v) - 1 + \rho^-(v) \cdot m \) additions. Hence, for a vertex \( v \in V_i \),

\[
\rho^-(v) + \sum_{m=1}^{M} \rho^-(v) \cdot 2(m+1) = \rho^-(v) \cdot (M^2 + 3M + 1)
\]

and the total number of additions is

\[
\text{add} = \sum_{i=1}^{n} \sum_{v \in V_i} \left( \rho^-(v) \cdot \left( \frac{1}{2} M^2 + \frac{3}{2} M + 1 \right) - (M-1) \right)
\]

\[
= \left( \frac{1}{2} M^2 + \frac{3}{2} M + 1 \right) \sum_{i=1}^{n} \sum_{v \in V_i} \rho^-(v) - (M-1) \cdot \sum_{i=1}^{n} \sum_{v \in V_i} 1.
\] (10)

Every edge in \( E \) is counted exactly once in the sum in (9), since if \( e : u \rightarrow v \), then \( \text{fin}(e) \in V_i \) for exactly one value of \( i \in \{1, 2, \ldots, n\} \). Thus the sum in (9) is \( |E| \). The second sum in (10) is \( |V| - 1 \), since every vertex except \( A \) is in \( \bigcup_{i=1}^{n} V_i \). Thus from (9) and (10), we have

\[
\text{mult} = (M^2 + 3M + 1) \cdot |E|
\]

\[
\text{add} = \left( \frac{1}{2} M^2 + \frac{3}{2} M + 1 \right) \cdot |E| - (M + 1) \cdot (|V| - 1)
\]

so that the total number of arithmetic operations required by the algorithm is

\[
\left( \frac{3}{2} M^2 + \frac{9}{2} M + 2 \right) \cdot |E| - (M + 1) \cdot |V| + M + 1
\]

\[
\leq \left( \frac{3}{2} M^2 + \frac{9}{2} M + 2 \right) \cdot |E|.
\]

We have \( |V| \geq 1 \), and \( |E| - |V| + 1 \geq 0 \) (since the trellis is connected), so that the total number of operations required is bounded above by \( \left( \frac{3}{2} M^2 + \frac{9}{2} M + 2 \right) \cdot |E| \) and bounded below by \( \left( \frac{3}{2} M^2 + \frac{7}{2} M + 1 \right) \cdot |E| \) (disregarding the complexity of the computation of \( g_i(e) \)).

In analogy to the forward numerator in Definition 3 we can also define a backward numerator.

**Definition 4 (Backward Numerator):** The \( m \)-th backward numerator \( \beta^{(m)}(v) \) of a vertex \( v \in V_i \) is defined as

\[
\beta^{(m)}(v) := \sum_{P : A \rightarrow B} (f(P))^m \cdot \lambda(P)
\]

with initial values

\[
\beta^{(m)}(B) = \begin{cases} 
1 & : m = 0 \\
0 & : m > 0 
\end{cases}
\]

**Theorem 3 (Backward Recursion):** The \( m \)-th backward numerator \( \beta^{(m)}(v) \) of a vertex \( v \in V_i \) can be calculated in a trellis \( T \) by

\[
\beta^{(m)}(v) = \sum_{e : \text{init}(e) = v} \lambda(e) \cdot \sum_{l=0}^{m} \left( \begin{array}{c} m \\ l \end{array} \right) (g_{i+1}(e))^l \cdot \beta^{(m-l)}(\text{fin}(e)).
\]

**Proof:** The proof is analogous to the proof of Theorem 1.

It obviously holds that \( \alpha^{(m)}(B) = \beta^{(m)}(A) = \beta^{(m)}(T) \), providing the \( m \)-th moment

\[
\bar{\beta}^{(m)}(T) := \frac{\beta^{(m)}(T)}{\theta^{(0)}(T)} = \sum_{P : A \rightarrow B} (f(P))^m \cdot \lambda(P)
\]

of the distribution of function \( f \) given \( T \).

In analogy to the BCJR algorithm [1] for calculating symbol probabilities, we next consider the calculation of moments of \( f \) introducing a constraint on the value of the \( c \)-labels at a certain depth \( i \) in the trellis. I.e., the moments are calculated in a sub-trellis of \( T \).

**Definition 5 (Symbol Moment):** We define the \( m \)-th symbol moment \( \Omega_i^{(m)}(T, x) \) at depth \( i \) of a trellis \( T \) as

\[
\Omega_i^{(m)}(T, x) := \sum_{P : A \rightarrow B} \left( \frac{f(P)}{\lambda(P)} \right)^m \cdot \lambda(P)
\]

where \( c_i = c(e_i) \) and \( e_i \in E_{i-1,i} \) is the \( i \)-th edge of path \( P \).

**Theorem 4:** The \( m \)-th symbol moment can be calculated by

\[
\bar{\Omega}_i^{(m)}(T, x) := \frac{\Omega_i^{(m)}(T, x)}{\Omega_i^{(0)}(T, x)}
\]

with

\[
\Omega_i^{(m)}(T, x) := \sum_{e \in E_{i-1,i}} \lambda(e) \cdot \sum_{l=0}^{m} \left( \begin{array}{c} m \\ l \end{array} \right) \beta^{(m-l)}(\text{fin}(e)) \cdot 
\]

\[
\cdot \sum_{k=0}^{l} \left( \begin{array}{c} l \\ k \end{array} \right) (g_i(e))^k \cdot \omega^{(l-k)}(\text{init}(e)).
\] (11)

**Proof:** Let \( P_H \) and \( P_T \) denote the head and tail parts of the paths \( P : A \rightarrow B \) through the trellis \( T \), with an edge \( e \) in between, i.e., \( P = P_H \cup P_T \) with \( \text{init}(P_H) = A, \text{fin}(P_H) = \text{init}(e), \text{fin}(e) = \text{init}(P_T) \) and \( \text{fin}(P_T) = B \), for a given depth \( i \) and \( e \in E_{i-1,i} \). Then we can write

\[
\Omega_i^{(m)}(T, x) = \sum_{P : A \rightarrow B} (f(P))^m \cdot \lambda(P)
\]

\[
= \sum_{e \in E_{i-1,i}} \sum_{\text{fin}(e) \geq x} \left( \sum_{e : \text{init}(e) = x} (f(P_H) + g_i(e) + f(P_T))^m \right) \cdot \lambda(P_H \cup P_T).
\]
Applying Bayes’ rule twice and separating the λ-labels we obtain

\[
\Omega_i^{(m)}(T, x) = \sum_{e \in E_{i-1,i}} \lambda(e) \cdot \sum_{l=0}^{m} \binom{m}{l} (f(P_T))^m - l \cdot \lambda(P_T) \cdot \binom{l}{k} (g_i(e))^k \cdot (f(P_H))^l - k \cdot \lambda(P_H)
\]

and using the definitions of forward and backward numerators finally yields the assertion of the theorem.

**Theorem 5 (Computational Complexity):** Given the forward numerators \( \alpha^{(m)}(v) \) and the backward numerators \( \beta^{(m)}(v) \) up to order \( m \) for all \( v \in \mathcal{V} \), the computation of \( \Omega_i^{(m)}(T, x) \) for all \( i \in 1 \ldots n \) requires \( O(|\mathcal{E}|) \) arithmetic operations.

**Proof:** Consider Equation (11). The sum over \( k \) requires \( 2l \) multiplications and \( l \) additions. The sum over \( l \) requires

\[
\sum_{l=0}^{m} (2l + 2) - 1 = m^2 + 3m + 1
\]

multiplications and

\[
\sum_{l=0}^{m} l + (m - 1) = \frac{1}{2} (m^2 + 3m - 2)
\]

additions. There are at most |\( E_{i-1,i} \) | edges \( e \) for which \( e \in E_{i-1,i} \) and \( e(\mathcal{E}) = x \), thus the sum over these edges requires at most |\( E_{i-1,i} \) | \cdot \((m^2 + 3m + 1) + 1\) multiplications and |\( E_{i-1,i} \) | \cdot \frac{1}{2} (m^2 + 3m - 2) + |\( E_{i-1,i} \) | - 1 additions. As we calculate the symbol moments for all \( i \in 1 \ldots n \), we can finally upper limit the requirements by

\[
\text{mult} \leq |\mathcal{E}| \cdot (m^2 + 3m + 2) \\
\text{add} \leq |\mathcal{E}| \cdot 0.5 (m^2 + 3m) - n.
\]

**Remark 3 (Forward/Backward Moments):** For numeric reasons it may be advantageous to directly compute the forward and backward moments

\[
\tilde{\alpha}^{(m)}(v) := \frac{\alpha^{(m)}(v)}{\alpha^{(0)}(v)} \quad \text{and} \quad \tilde{\beta}^{(m)}(v) := \frac{\beta^{(m)}(v)}{\beta^{(0)}(v)},
\]

respectively, and to calculate and carry the 0-th numerators (flows) in the logarithmic domain.

Finally in this section, we describe the calculation of distributions over all paths \( P : A \rightarrow B \), or a subset of paths, in the trellis in analogy to the calculation of moments and symbol moments, respectively.

**Definition 6 (Forward/Backward Distribution):** We define the forward distribution \( \alpha^D(v) \) and the backward distribution \( \beta^D(v) \) at a vertex \( v \) as the mapping functions

\[
q \mapsto \sum_{P : A \rightarrow v} \lambda(P) \quad \text{and} \quad q \mapsto \sum_{P \rightarrow B} \lambda(P),
\]

respectively.

**Theorem 6:** The forward distribution \( \alpha^D(v) \) at a vertex \( v \in \mathcal{V} \) can be recursively calculated in the trellis by

\[
\alpha^D(v) = \sum_{e : \text{fin}(e) = v} (\alpha^D(\text{init}(e)) \oplus g_i(e)) \cdot \lambda(e),
\]

where \( a(u) \oplus b \) denotes a shift of the domain of the distribution \( a(u) \) by \( b \), and \( \alpha^D(A) \) equals the Dirac function. The calculation of \( \beta^D(v) \) is analog with \( \beta^D(B) \) being the Dirac function. The distribution \( \theta^D(T) \) and the symbol distribution \( \Omega_i^D(x, T) \) can be calculated by

\[
\theta^D(T) = \sum_{v \in \mathcal{V}} \alpha^D(v) \ast \beta^D(v)
\]

and

\[
\Omega_i^D(T, x) = \sum_{e \in E_{i-1,i}} (\alpha^D(\text{init}(e)) \oplus g_i(e)) \ast \beta^D(\text{fin}(e)) \cdot \lambda(e).
\]

respectively. Herby, \( \ast \) denotes the convolution operator, i.e. for two distributions \( a(u) \) and \( b(u) \) it holds

\[
a(u) \ast b(u) = \int_{-\infty}^{\infty} a(v) \cdot b(u - v) \, dv.
\]

**Proof:** Theorem 6 follows directly from Definition 6.

**Remark 4 (Density Distributions):** When normalizing distributions by the corresponding flow, we obtain density distributions.

**Remark 5 (Probability Density Functions):** For \( \lambda(e) \) being probabilities, normalized distributions are probability density functions with the mapping \( f(P) \rightarrow P(f(P)) \) and \( \sum_{f(P)} P(f(P)) = 1 \).

**Remark 6:** By Theorem 6, the complexity due to the calculation on the trellis is in general not reduced (except for the hard decision case) as infinite resolution of the domain of \( \alpha^D(v) \) etc. is required. However, in Appendix B an algorithm is introduced which approximates Theorem 6 and does reduce complexity.

**Remark 7:** We cannot only determine the distribution and its moments of a trellis or sub-trellis, but also of a single edge.

**Remark 8:** The symbol distribution for two sub-trellises of the [7,5,3] convolutional code, namely the sub-codes with the i-th code bit \( c_i = +1 \) and \( c_i = -1 \), respectively, is given in Example 1. The curves obtained by Gaussian approximation almost coincide with the ones plotted in Figure 1.

**Remark 9:** It is straight forward to extend the proposed algorithm to the calculation of joint moments of two or more functions. E.g.,

\[
\tilde{\theta}^{k,m}(y,z) := \frac{\sum_{P} (f_y(P))^k \cdot (f_z(P))^m \cdot \lambda(P)}{\sum_{P} \lambda(P)}
\]

can be calculated using

\[
\alpha^{k,m}(y,z)(v) := \sum_{P : A \rightarrow v} (f_y(P))^k \cdot (f_z(P))^m \cdot \lambda(P)
\]

\[
= \sum_{e : \text{fin}(e) = v} \lambda(e) \cdot \sum_{j=0 \atop f(P) = q}^{m} \binom{k}{j} \binom{m}{l} \cdot g_{k-j}(e) \cdot g_{m-l}(e) \cdot \alpha^{j,l}(\text{init}(e))
\]

with \( i = \text{depth}(v) \).
IV. APPLICATIONS

We will now apply the results of Section III to linear block codes. We compute the moments

\[ E_C[(H(e|w))^m | r, c_i = x] := \sum_{c \in \mathbb{C}} (H(e|w))^m P(c|r, c_i = x) \]

dependent on the distribution

\[ D : q = H(e|w) \rightarrow P(q|r, c_i = x) = \sum_{w \in \mathbb{C}} P(c|r, c_i = x) \]

over all code words \( c \in \mathbb{C} \) given a received word \( r \) and the \( i \)-th code bit being \( c_i = x \in \{-1, 1\} \), where

\[ H(e|w) = -\log P(c|w) \]

is the conditional uncertainty of \( c \) given a word \( w \) and \( P(c|r) \) is the conditional probability of \( c \) given \( r \). These moments are required, e.g., for the discrimination belief propagation algorithm in [2]. As a special case we can calculate the conditional mean uncertainty or entropy

\[ H(C|r) = \sum_{c \in \mathbb{C}} H(c|r) \cdot P(c|r) \]

of a code or sub-code given \( r \).

Both for hard decision (BSC) and soft decision (AWGN channel) the conditional uncertainty is linearly related to the correlation \( cw^T \) (cf. Appendix A),

\[ H(c|w) = K_1 + K_2 \cdot cw^T, \]

with \( K_1 \) and \( K_2 \) being constant functions of error probability and vector \( w \) (assuming equiprobable code words). Therefore, when applying the binomial theorem,

\[
\begin{align*}
\sum_{c \in \mathbb{C}} (H(e|w))^m \cdot P(c|r, c_i = x) & = \sum_{c \in \mathbb{C}} (K_1 + K_2 \cdot cw^T)^m \cdot P(c|r, c_i = x) \\
& = \sum_{c \in \mathbb{C}} \sum_{l=0}^{m} \binom{m}{l} K_1^{m-l} K_2^l (cw^T)^l P(c|r, c_i = x) \\
& = \sum_{l=0}^{m} \binom{m}{l} K_1^{m-l} K_2^l E_C[(cw^T)^l | r, c_i = x],
\end{align*}
\]

it is sufficient to calculate the moments

\[ E_C[(cw^T)^m | r, c_i = x] = \sum_{c \in \mathbb{C}} (cw^T)^m \cdot P(c|r, c_i = x) \]

of the correlation \( cw^T \) on the trellis which will be done in the following.

Consider a binary linear block code \( \mathbb{C} \) of length \( n \) which is representable in a trellis, e.g., a terminated convolutional code. Let the \( c \)-labels \( c(e) = c_i \in \{\pm 1\} \) be the bipolar representation of the code bit labeling edge \( e \). To each path \( P : A \rightarrow B \) it belongs a sequence \( c(P) \) of \( n \) \( c \)-labels representing a code word \( c \in \mathbb{C} \). Let \( r = [r_1 r_2 \cdots r_n] \), \( r_i \in \mathbb{R} \), be the noisy version of a code word \( c \) after transmission over a memory-less channel. Let the \( \lambda \)-label of a path \( P \) be the conditional probability of the received word \( r \) given the code word \( c \), i.e., \( \lambda(P) = P(r|c) \). Let further the function \( f \) of the paths’ \( c \)-labels, i.e., the function of the code words, be the correlation (inner product) of \( w \) and \( c \),

\[ f(P) = f(c(P)) = cw^T = \sum_{i=1}^{n} c_i w_i. \]

Hence, \( g_i(c) = c_i w_i \) and the separability criterion (3) is fulfilled. In the trellis of \( \mathbb{C} \), for each vertex \( v \in \mathcal{V} \) the \( c \)-labels \( c(e) \) of edges \( e : \text{init}(e) = v \) emerging from \( v \) are distinct. Therefore there is a one-to-one mapping of each code word \( c \) to a path \( P \) in the trellis, and we can apply the theorems of Section III replacing \( \sum_{r} \) by \( \sum_{c} \). Applying BAYES’ rule to (13),

\[ E_C[(cw^T)^m | r, c_i = x] = \sum_{c \in \mathbb{C}; c_i = x} (cw^T)^m P(c|r), \]

and comparing with Definition 5 we observe that Theorems 1 and 3 hold, and hence these moments can be calculated in the trellis according to Theorem 4 as the symbol moments

\[ E_C[(cw^T)^m | r, c_i = x] = \bar{\Omega}_i^{(m)}(x). \]

Analogously, when omitting the code bit constraint \( c_i = x \), the moments are given by

\[ E_C[(cw^T)^m | r] = \sum_{c \in \mathbb{C}} (cw^T)^m \cdot P(c|r) = \bar{\theta}^{(m)}(T). \]

For \( w = r \), \( m = 1 \) and \( g_i(e) = c_i r_i \), we can thus calculate the conditional entropies

\[ H(C|r) = \sum_{c \in \mathbb{C}} H(c|r) \cdot P(c|r) = K_1 + K_2 \cdot \bar{\theta}^{(1)}(T) \]

and

\[ H(C_i(x)|r) = \sum_{c \in \mathbb{C}; c_i = x} H(c|r) \cdot P(c|r) = K_1 + K_2 \cdot \bar{\Omega}_i^{(1)}(x) \]

dependent on the code \( \mathbb{C} \) and the sub-code \( C_i(x) = \{c \in \mathbb{C} : c_i = x\} \) given \( r \), respectively. While \( H(C|r) \) can also be calculated with the classical BCJR algorithm as

\[ \sum_{c \in \mathbb{C}} cr^T \cdot P(c|r) = \sum_{i=1}^{n} \sum_{c \in \mathbb{C}} c_i r_i \cdot P(c|r) \]
\[ = \sum_{i=1}^{n} r_i \cdot \left( \sum_{c \in \mathbb{C}; c_i = x} P(c|r) - \sum_{c \in \mathbb{C}; c_i = -x} P(c|r) \right), \]

this does not hold for the conditional entropy of \( C_i(x) \).

Remark 10: For a convolutional code with \( c \) outputs, to each edge in the trellis are assigned \( c \) code symbols. To apply our definition of a single symbol label per edge, each edge \( e \) of the original trellis is replaced by a path \( e_1, e_2, \ldots, e_n \) of \( c \) edges which fulfill

\[ \text{init}(e) = \text{init}(e_1), \quad \text{fin}(e_j) = \text{init}(e_{j+1}), \ldots, \text{fin}(e_n) = \text{fin}(e) \]

and to each edge \( e_j \) one code symbol is assigned.

Example 5: Figure 1 shows the distribution of \( P(cr^T, c_i = \pm 1 | r) \) over \( cr^T \) for the \([5 7]_{oct} \) convolutional.
code of length \( n = 200 \) given a noisy received word \( r \) after transmission over a BSC with bit error probability \( p = 0.35 \). These are the normalized symbol distributions \( \Omega^D_{i=1}(\pm 1) \) weighted by the probability \( P(c_i = \pm 1 | r) \).

Example 6: Figure 3 shows a distribution of the terminated \([7 5]_{oct}\) convolutional code as well the GAUSSian approximation given the first two moments for a BSC.

### V. Conclusions

A trellis represents a general distribution which can be marginalized, e.g. with respect to edge labels. Two algorithms for computations on the trellis were presented: One allowing to calculate distributions, the other to compute their moments, allowing to approximate the distributions. The latter was derived by generalizing the forward/backward recursion as known from the BCJR algorithm. The results were transferred to the concrete problem of computing the moments of the conditional distribution of the correlation between a block code and some given word. The moment calculation algorithm is a requirement for efficient implementation of the discriminated belief propagation algorithm in [2]. It can also be used to calculate the conditional entropy of a code or sub-code. Though not the focus of this paper, in the Appendix it is shown that the algorithm does not restrict to calculation with real numbers, but is valid for any commutative semi-ring, thus providing a generalization of the Viterbi algorithm. The asymptotic complexity of the moment computation algorithm is the same as for the BCJR algorithm.

### APPENDIX

#### A. Relation between Uncertainty and Correlation

The conditional uncertainty of a code word \( c \) given a word \( w \) is defined as

\[
H(c|w) := - \log_2 P(c|w) = - \log_2 P(w|c) + \log_2 \frac{P(w)}{P(c)} \tag{K_{1a}}
\]

where \( K_{1a} \) is a constant assuming equiprobable code words. Assuming further that \( w_i \) is independent of \( c_j \) for \( i \neq j \) it follows that

\[
\log_2 P(w|c) = \log_2 \prod_{i=1}^{n} P(w_i|c_i) = \sum_{i=1}^{n} \log_2 P(w_i|c_i).
\]

- For a binary symmetric channel (BSC) with \( w_i, c_i \in \{ \pm 1 \} \) and error probability \( p \) the Hamming distance between \( c \) and \( w \) is \( \frac{n-cw^T}{2} \) which gives

\[
\log_2 P(w|c) = \frac{n-cw^T}{2} \log_2 p + \frac{n+cw^T}{2} \log_2 (1-p) = \frac{n}{2} \log_2 (p(1-p)) + cw^T \cdot \frac{1}{2} \log_2 \frac{1-p}{p}. \tag{K_{ib}}
\]

- For an AWGN channel with noise variance \( \sigma^2 \) we obtain (note that \( P(w|c) \) actually is the GAUSS probability density)

\[
\log_2 P(w|c) = \sum_{i=1}^{n} \log_2 \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{(w_i - c_i)^2}{2\sigma^2} \right) = \sum_{i=1}^{n} \left( \log_2 \frac{1}{\sqrt{2\pi\sigma}} - \frac{(w_i - c_i)^2}{2\sigma^2 \cdot \ln 2} \right) = n \log_2 \frac{1}{\sqrt{2\pi\sigma}} - \frac{n}{2\sigma^2 \cdot \ln 2} \sum_{i=1}^{n} (c_i^2 + w_i^2 - 2c_i w_i) = n \log_2 \frac{1}{\sqrt{2\pi\sigma}} - \frac{n+cw^T}{2\sigma^2 \cdot \ln 2} + cw^T \cdot \frac{1}{\sigma^2 \cdot \ln 2}. \tag{K_{ib}}
\]

In either case we can thus express the conditional uncertainty as

\[
H(c|w) = (K_{1a} - K_{ib}) - K_{b} \cdot cw^T.
\]

I.e., the uncertainty is linearly related to the correlation.

#### B. Calculating the Actual Distribution

For a trellis of rank \( n \) and \( g_i(e) \in \{ \pm 1 \} \), which is the case for hard decision decoding, the domain of the distributions, i.e., the values that \( f(P) \) can take, is \( \mathbb{D} = \{-n, -n+2, \ldots, n-2, n\} \) with cardinality \( |\mathbb{D}| = n+1 \). In this case the distributions can be directly implemented as vectors of length \( n+1 \). A shift \( \mathbb{D} \) of the domain is simply a shift of the vector contents, and the correlation operation \( * \) is discrete.

In case of soft decision, the domain needs to be quantized. For GAUSSian distributions, an efficient way for uniform mid-tread quantization is to carry along the mean value \( \mu \) of the distribution and to arrange the partitions equally to both sides of it, storing the partition contents in vectors \( d \). When extending a path \( P \) by an edge \( e \in E_{i-1,i} \) in the forward/backward recursion (lengthening), the domain of \( f(P) \) is shifted by \( g_i(e) \), i.e., \( g_i(e) \) is added to the \( \mu \). However, when joining paths in a vertex, the mean values of the incoming path...
distributions do usually not coincide. Hence a new mean value $\mu_{\text{new}}$ has to be determined and the partition contents need to be distributed.

Let the vectors $d$ be of length $(2N + 1)$, each element corresponding to a partition of width $\Delta_w$. The partitions are indexed by $j \in \{-N, -N + 1, \ldots, N\}$, where $j = 0$ denotes the center partition around the mean value. The mean value $\mu_{\text{new}}$ is the weighted sum of the mean values $\mu_{in}$ of the involved distributions in vectors $d_{in}$. E.g., for the forward recursion,

$$\mu_{\text{new}} = \alpha^\mu(v) := \sum_{e: \text{fin}(e) = v} \left(\frac{\alpha^\mu(\text{init}(e)) + g_t(e)}{\mu_{in}}\right) \lambda(e) \cdot \eta(A, \text{init}(e)) \cdot \sum_{e': \text{fin}(e') = v} \lambda(e') \cdot \eta(A, \text{init}(e')),$$

with $\alpha^D(A) = 0$, where $\mu_{in}$ is the mean value of the distribution $\alpha^D(\text{init}(e))$ after lengthening by $g_t(e)$, and $\alpha^\mu(v)$ is the mean of the forward distribution $\alpha^D(v)$. The final distribution vector $d_{\text{new}} = \alpha^D(v)$ is the weighted sum of the vectors $d_{\text{in}} = \alpha^d(\text{init}(e))$ according to the new partition margins with $\Delta_\mu = \mu_{\text{new}} - \mu_{in}$ as follows (cf. Figure 4).

- $e = \Delta_\mu - \left\lfloor \frac{\Delta_\mu}{\Delta_w} \right\rfloor \in [0, 1)$
- $d_{\text{out}} = 0$ (all-zero vector)
- $d_{\text{out}}[-N] = \sum_{j=-N}^{N} d_{\text{in}}[j]$
- $d_{\text{out}}[N] = \sum_{j=N+1}^{N} d_{\text{in}}[j]$
- for $\max \left(-N, -N + \left\lfloor \frac{\Delta_\mu}{\Delta_w} \right\rfloor + 1\right) \leq j \leq \min \left(N, N + \left\lfloor \frac{\Delta_\mu}{\Delta_w} \right\rfloor\right)$
  - $d_{\text{out}}[j] = \frac{\Delta_\mu}{\Delta_w} - 1 = \epsilon = d_{\text{in}}[j]$
  - $d_{\text{out}}[j] = (1 - \epsilon) \cdot d_{\text{in}}[j]$

where $a + = b$ denotes the addition of $b$ to $a$, i.e., $a = a + b$. The forward distribution vector $\alpha^d(A)$ is initialized. The backward distribution is computed analogously.

With the two procedures of lengthening and joining the mean value of the symbol distribution can be calculated by

$$\Omega^\mu_i(x, T) = \sum_{e \in E_{i-1}, c(e) = x} \left(\frac{\alpha^\mu(\text{init}(e)) + g_t(e) + \beta^\mu(\text{fin}(e))}{\mu_{in}}\right) \lambda(A, \text{init}(e)) \cdot \lambda(e) \cdot \eta(\text{fin}(e), B) \cdot \sum_{e' \in E_{i-1}, c(e') = x} \lambda(A, \text{init}(e')) \cdot \lambda(e') \cdot \eta(\text{fin}(e'), B),$$

and the discrete symbol distribution vector $d_{\text{out}} = \Omega^d_i(x, T)$ is obtained by convolving the forward and backward distribution vectors $\alpha^d(\text{init}(e))$ and $\beta^d(\text{fin}(e))$ for each edge $e \in E_{i-1, i}$: $c(e) = x$,

$$d_{\text{in}} = \alpha^d(\text{init}(e)) \ast \beta^d(\text{fin}(e)),$$

followed by a weighted re-distribution of the vector contents of the $d_{\text{in}}$ to $d_{\text{out}}$.

C. Generalization to Calculations on a Semi-ring

In the main part of this paper, the computation of moments in the trellis is introduced for real numbers. However, the algorithm is valid for the more general algebraic structure of commutative semi-rings. The 0-th forward moment then results in the Viterbi algorithm on semi-rings.

Let the $\lambda$-label and the $c$-label come from an algebraic set $S$ which is closed under the two binary operations $\oplus$ and $\odot$, called addition and multiplication, which satisfy the following axioms:

- The operation $\odot$ is associative and commutative, and there is an identity element $1^\odot$ such that $s \odot 1^\odot = 1^\odot \odot s = s$ for all $s \in S$, making $(S, \odot)$ a commutative monoid.
- The operation $\oplus$ is associative and commutative, and there is an identity element $0^\oplus$ such that $s \oplus 0^\oplus = 0^\oplus \oplus s = s$ for all $s \in S$, making $(S, \oplus)$ a commutative monoid.
- The distributive law $(x \oplus y) \odot z = (x \odot z) \oplus (y \odot z)$, for all triples $(x, y, z)$ from $S$.
- The identity element $0^\oplus$ of the addition annihilates $S$, i.e., $0^\oplus \oplus s = 0^\oplus = s$ for all $s \in S$.

The triple $(S, \odot, \oplus)$ is called a commutative semiring.

Let $a, b \in (S, \odot, \oplus)$ be elements of such a commutative semiring. We define the following notation:

$$a^m := \begin{cases} a \odot a \odot \ldots \odot a & m \in \mathbb{N} \\ 1^\odot & m = 0 \end{cases}$$

$$n a := \sum_{i=1}^{n} a$$

with $n a \oplus b = n (a \oplus b)$ and $\mathbb{N}$ being the set of natural numbers. Then the binomial theorem can be written as

$$(a \oplus b)^m = \sum_{l=0}^{m} \binom{m}{l} a^l \odot b^{m-l}, \quad m, l \in \mathbb{N}_0, \quad a, b \in (S, \odot, \oplus)$$

with the binomial coefficient $\binom{m}{l} \in \mathbb{N}_0 = \{0 \cup \mathbb{N}\}$. In analogy to Definition 3 and Theorem 1 we can now define the forward numerator and its calculation on a semi-ring.

Definition 7: We define the $m$-th forward numerator of a function $f \in (S, \odot, \oplus)$ at vertex $v$ of a trellis $T$ as

$$\alpha^{(m)}(v) := \sum_{P; A \rightarrow v} \lambda(P) \odot (f(P))^m$$

with initial values

$$\alpha^{(m)}(A) := \begin{cases} 1^\odot & m = 0 \\ 0^\oplus & m > 0 \end{cases} .$$

Theorem 7: The $m$-th forward moment $\alpha^{(m)}(v)$ of a vertex $v \in V_i$ on depth $i$ can be recursively calculated on a trellis $T$ and a commutative semiring $(S, \odot, \oplus)$ by

$$\alpha^{(m)}(v) = \sum_{e: \text{fin}(e) = v} \lambda(e) \odot \sum_{l=0}^{m} \binom{m}{l} (g_t(e))^l \odot \alpha^{(m-l)}(\text{init}(e))$$

(16)
for all functions $f(P : A \rightarrow v)$ and $g_j$, $j = 1, \ldots, i$, which fulfill

$$f(P) = f(e_1 e_2 \cdots e_i) = g_1(e_1) \oplus g_2(e_2) \oplus \cdots \oplus g_i(e_i). \quad (17)$$

**Proof:** The proof is by induction on $\text{depth}(v)$. For $\text{depth}(v) = 1$ the algorithm computes

$$\alpha^{(m)}(v) = \sum_{e : \text{fin}(e) = v} \lambda(e) \circ (1 \circ (g_1(e))^m \circ 1)$$

which is, as required, the sum of the labels on all edges $e$ joining $A$ to $v$, weighted by $(g_1(e))^m$. For a vertex $v$ at depth $i + 1$ the value assigned to $\alpha^{(m)}(v)$ is by the induction hypothesis

$$\alpha^{(m)}(v) = \sum_{e : \text{fin}(e) = v} \lambda(e) \circ \sum_{l=0}^{m} \binom{m}{l} (g_i(e))^l \circ \sum_{P : A \rightarrow \text{init}(v)} \lambda(P) \circ (f(P))^{m-l}.$$  

Using the axioms of the commutative semiring $(\mathbb{S}, \circ, \oplus)$ we have

$$\alpha^{(m)}(v) = \sum_{e : \text{fin}(e) = v} \sum_{P : A \rightarrow \text{init}(v)} \lambda(e) \circ \lambda(P)$$

$$\circ \sum_{l=0}^{m} \binom{m}{l} (g_i(e))^l \circ (f(P))^{m-l}.$$  

Applying Equation (17) and the binomial theorem we obtain

$$\alpha^{(m)}(v) = \sum_{e : \text{fin}(e) = v} \sum_{P : A \rightarrow \text{init}(v)} \lambda(P) \circ (f(P) \oplus g_i(e))^m.$$  

But every path from $A$ to $v$ must be of the form $Pe$, where $P$ is a path from $A$ to a vertex $u$ with $\text{depth}(u) = i$, $\text{init}(e) = u$ and $\text{fin}(e) = v$. Hence, $\alpha^{(m)}(v)$ is correctly calculated by the theorem.

**Remark 11:** Note that the complexity considerations in Theorems 2 and 5 transfer to the calculation on semi-rings. However, the terminology of “addition” and “multiplication” then refers to the operations $\oplus$ and $\circ$.

---

**REFERENCES**

[1] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *Information Theory, IEEE Transactions on*, Volume 20:Page(s):284 – 287, March 1974.

[2] U. Sorger. Discriminated Belief Propagation. Technical Report TR-CS-07-01, Computer Science and Communications Research Unit, Université du Luxembourg, 2007. http://wiki.uni.lu/csc/Discriminator+Based+Decoding.html.

[3] A.J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, Volume 13:Page(s):260 – 269, April 1967.

[4] G.D. Forney, Jr. Review of random tree codes. Appendix a, final report, contract nas2-3637; nasa cr73176; nasa ames res. ctr. moffett field, ca, Dec. 1967.

[5] J.K. Wolf. Efficient maximum-likelihood decoding of linear block codes using a trellis. *Information Theory, IEEE Transactions on*, vol.24(no.1):pp. 76–80, Jan. 1978.

[6] C.L. Massey. Foundation and methods of channel encoding. In *Proc. Int. Conf. Inform. Theory Syst.*, volume vol.65, Sept. 1978.

[7] G.D. Forney, Jr. Coset codes. ii. binary lattices and related codes. *Information Theory, IEEE Transactions on*, Volume 34:Page(s):1152 – 1187, Sep 1988.

[8] R.J. McEliece. On the BCJR Trellis for Linear Block Codes. *IEEE Transactions on Information Theory*, Vol. 42(No. 4):Pages: 1072 – 1092, July 1996.

[9] S.M. Aji and R.J. McEliece. The generalized distributive law. *Information Theory, IEEE Transactions on*, Volume: 46:On page(s): 325–343, Mar 2000.

[10] F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, Volume: 47:On page(s): 498–519, Feb 2001.

---

3. $\alpha \circ \sum_i B_i = \sum_i A \circ B_i$ requires distributive law (factor into sum)

- $\sum_i \sum_j A_{ij} = \sum_j \sum_i A_{ij}$ requires associativity and commutativity of $\circ$ (change order of sums)

- $A \circ B = B \circ A$ requires commutativity of $\circ$ (change order of factors)