Path Partitions and Forward-Only Trellis Algorithms

Xiao Ma and Aleksandar Kavčić, *Member, IEEE*

***Abstract*— This is a semi-tutorial paper on trellis-based algorithms. We argue that most decoding/detection algorithms described on trellises can be formulated as path-partitioning algorithms, with proper definitions of mappings from subsets of paths to metrics of subsets. Thereby, the only two operations needed are path-concatenation and path-collection, which play the roles of multiplication and addition, respectively. Furthermore, we show that the trellis structure permits the path-partitioning algorithms to be formulated as forward-only algorithms (with structures resembling the Viterbi algorithm), thus eliminating the need for backward computations regardless of what task needs to be performed on the trellis. While all of the actual decoding/detection algorithms presented here are rederivations of variations of previously known methods, we believe that the exposition of the algorithms in a unified manner as forward-only path-partitioning algorithms is the most intuitive manner in which to generalize the Viterbi algorithm. We also believe that this approach may in fact influence the practical implementation of the algorithms as well as influence the construction of other forward-only algorithms (e.g., byte-wise forward-only detection algorithms).**

***Index Terms*— BCJR algorithm, forward-backward algorithm, forward-only algorithm, Markov processes, sum-product algorithm, trellis graph, Viterbi algorithm**

I. Introduction

The Viterbi algorithm was proposed in 1967 [1] as a method for decoding convolutional codes. It has been shown that the Viterbi algorithm is a decoding/detection method which minimizes the sequence error probability for convolutional codes [2] and intersymbol interference (ISI) channels [3]. To minimize the symbol error probability, Bahl, Cocke, Jelinek, and Raviv [4] derived a forward-backward *a posteriori* probability (APP) decoding algorithm, the BCJR algorithm, which is similar in concept to the methods previously published in [5][6]. Unlike the Viterbi algorithm which has been widely utilized in different fields [2], the BCJR algorithm was rarely utilized in the coding field[1] until the invention of turbo codes [8].

Forney first mentioned the close relationship between the BCJR algorithm and the Viterbi algorithm in the Appendix of [2]. After the advent of the turbo codes, the similarities between these two algorithms have been disclosed by several authors. Based on the ideas of Tanner [9], Wiberg, Loeliger and Kötter [10] (see also [11]) developed a general framework for the description of codes and iterative decoding, where two general algorithms, the min-sum algorithm and the sum-product algorithm, were proposed. The min-sum algorithm is a generalized Viterbi algorithm (it can also be considered as a generalized version of the soft-output Viterbi algorithm (SOVA) developed by Hagenauer *et al.* [12] [13]). The sum-product algorithm can be considered as a generalized version of the BCJR algorithm. It has been pointed out in Section 3.6 of [11] that these two algorithms can be unified by using the "universal algebra" approach (the semiring view). The general description of the similarities between the Viterbi algorithm and the forward-backward algorithm can also be found in [14][15][16][17][18][19]. These available references are invaluable for grasping these two algorithms and the related algorithms in other fields. For the related algorithms, see [16][18][19][20] and the references therein.

In this paper, we focus on *forward-only* APP algorithms. Although we do not have a compelling argument that forward-only APP algorithms are absolutely required for practical systems, we should note that the forward-only APP algorithms are interesting at least from a theoretical point of view; see the Appendix of [2]. Different versions of the forward-only APP algorithm were developed by several authors who, in some cases, were not aware of similar work prior to theirs. In 1974, Lee [21] developed a forward-only APP algorithm for convolutional codes, which has a structure resembling the Viterbi algorithm. In 1982, a similar algorithm was derived in the context of detection of symbols transmitted over intersymbol interference (ISI) channels by Hayes, Cover and Riera [22], who also noticed the similarities between the Viterbi algorithm and the forward-only APP algorithm. (It should be noted that the algorithm in [22] makes connections to the work by Abend and Fritchman [23], who derived a non-trellis-based fixed-delay lookahead version of the forward-only algorithm in 1970.) In 1995, Li, Vucetic and Sato [24] proposed another version of the forward-only APP algorithm (named optimum soft-output algorithm, OSA) for intersymbol interference (ISI) channels.

The algorithms presented in [21], [22], [24] apply only to *non-recursive* finite-state machines, and as such, do not apply to, for example, the turbo codes in [8]. In this paper, we develop a general construction of forward-only algorithms for both recursive and non-recursive finite-state machines. (We should also point out that a special construction for turbo codes was proposed in [25].) Beyond this subtle novelty, there is very little that we add in terms of new algorithms. However, by virtue of studying forward-only path partitions, we present a general theory for designing forward-only trellis algorithms that could

Authors are with the Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138.

This work was supported by the National Science Foundation under Grant No. CCR-9984297 and by the National Storage Industry Consortium.

[1] This algorithm was however frequently utilized in signal processing [7], where it is called the Baum-Welch algorithm.

be used in various decoding/detection applications.

The main goal of this paper is to demonstrate that most decoding/detection problems defined on trellises can be solved by forward-only methods. Obviously, any decoding/detection problem can be solved by an exhaustive search method. For a problem defined on a trellis, the exhaustive method is inefficient (even unfeasible) unless the length of the trellis is small. Generally, the longer the trellis is, the less efficient is the exhaustive method. This is a consequence of the graph-structure of the trellis. So, it is important to find a method that can recursively extend the solutions in a step-by-step manner. Our aim is to show that by constructing proper path-partitioning algorithms on the trellis graph, we can formulate most known trellis-based algorithms as forward-only computations with structures that resemble the structure of the Viterbi algorithm. That is, forward-only trellis-based algorithms are formulated using the concept of the *survivor memory* [2].

We formulate the path-partitioning algorithms on sets whose elements are trellis paths. Defined on these sets of paths are two operations: *path-collection* and *path-concatenation*. Respectively, they play the roles of addition and multiplication over the trellis semiring, with the distributive law preserved. Therefore, as we will see, the path-partitioning algorithms are results of applying the sum-product algorithm [18] (or the generalized distributive law [16]) to the trellis semiring. By introducing symbolic graph representations, the descriptions of the path-partitioning algorithms become very simple and intuitive. Consequently, most trellis-based algorithms are obtained from the path-partitioning algorithms by defining proper mappings 1) from the subsets to their metrics, 2) from the path-collection operation to a suitably defined metric addition operation, and 3) from the path-concatenation operation to a suitably defined metric multiplication operation. That is, most trellis-based algorithms may be obtained by means of a mapping from the trellis semiring to a space of path-set metrics. This mapping has some properties that resemble a homomorphism, but we will see that it is not a strict-sense homomorphism.

We begin the paper in Section II with the concept of a trellis. The basic notation is also given in Section II. In particular, subsets of interest are represented by symbolic graphs. When the partitions are described by these symbolic graphs, the problems and the algorithms become simple and intuitive. In Section II-C, the forward-backward path-partitioning algorithm and the forward-only path-partitioning algorithm are described using both mathematical expressions and symbolic graphs. In Section III, by applying the path-partitioning algorithms to decoding/detection problems, we rederive two APP algorithms: the forward-backward algorithm and the forward-only algorithm. In Section IV, we show how to reduce the computational complexity and the delay. The forward-only APP algorithm with the fixed-delay constraint is illustrated using a decoding example. We also point out the relationships among several existing algorithms. Section V concludes the paper. For the sake of completeness, we include an Appendix in which we show how to rederive both the well-known Viterbi algorithm [2], [3] and the less familiar list Viterbi algorithm [2], [26] from the exposed path-partitioning trellis algorithms.

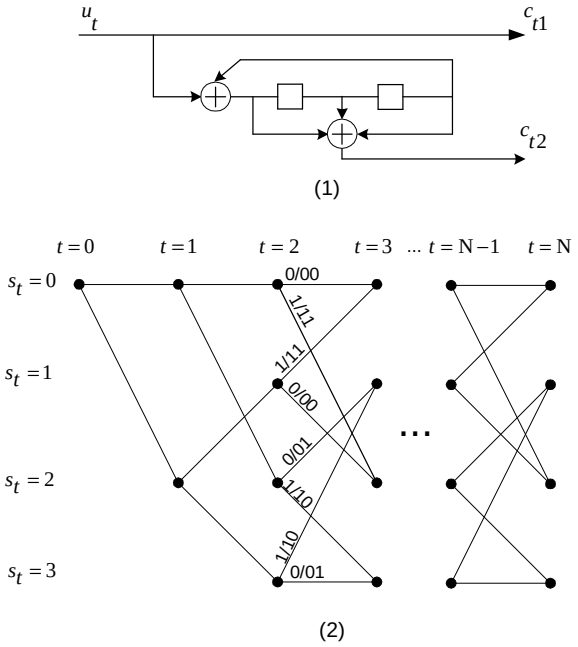

Fig. 1. (1) A binary convolutional encoder. (2) The trellis representation of the encoder.

II. Trellis, Semiring and Path-Partitioning Algorithms

A. Labelled trellis

Figure 1-(2) is an example of a labelled trellis that represents the binary convolutional encoder shown in Figure 1-(1). We will frequently use this example to illustrate the notation and algorithms in this paper. In general, a *labelled trellis* is a directed graph that defines certain relationships between four nonempty sets[2] $\mathcal{I}$, $\mathcal{S}$, $\mathcal{B}$ and $\mathcal{L}$:

1) The index set is $\mathcal{I} = \{0, 1, \ldots, N\}$. The elements of $\mathcal{I}$ are called *time*. $N = |\mathcal{I}| - 1$ is called the *length* of the trellis. Hereafter, the cardinality of a finite set $\mathcal{X}$ is denoted by $|\mathcal{X}|$.
2) The state set $\mathcal{S}$, whose elements are called *states*, is partitioned into $N+1$ disjoint subsets, that is, $\mathcal{S} = \bigcup_{t=0}^{N} \mathcal{S}_t$. In a trellis graph, each state (also called *a vertex*) is represented by one black dot $\bullet$, see Figure 1-(2). These black dots $\bullet$ are arranged into an array of $|\mathcal{I}|$ columns that are indexed by $t \in \mathcal{I}$. From left to right, the columns represent $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_N$, respectively; see Figure 1-(2). The column corresponding to time t contains $|\mathcal{S}_t|$ dots, which are indexed, in general, by $\{0, 1, \ldots, |\mathcal{S}_t| - 1\}$ from top to bottom[3].
3) The branch set $\mathcal{B}$, whose elements are called *branches*, is partitioned into N disjoint subsets. That is, $\mathcal{B} = \bigcup_{t=1}^{N} \mathcal{B}_t$. A branch $b \in \mathcal{B}$ is represented by a 3-tuple $(\sigma^-(b), l(b), \sigma^+(b))$, where $\sigma^-(b) \in \mathcal{S}$ and $\sigma^+(b) \in \mathcal{S}$ represent the two states connected by b, and $l(b) \in \mathcal{L}$

[2] All of these four sets are assumed to be finite in this paper.

[3] Sometimes we use more convenient notation. For example, in Figure 1-(2), $\mathcal{S}_1$ is indexed by $\{0, 2\}$, not by $\{0, 1\}$.

represents the label assigned to b. The important character of a trellis is that if $b \in \mathcal{B}_t$ then $\sigma^-(b) \in \mathcal{S}_{t-1}$ and $\sigma^+(b) \in \mathcal{S}_t$. In a trellis graph, each branch (also called *an edge*) connects two dots $\bullet$ in the adjacent columns. For $b \in \mathcal{B}$, we call $\sigma^-(b)$, $l(b)$ and $\sigma^+(b)$ the starting state, the label and the ending state of b, respectively. In Figure 1-(2), the 8 branches (edges) that connect states at time $t = 2$ to states at time $t = 3$ constitute the set $\mathcal{B}_3$.

4) The label set is denoted by $\mathcal{L}$. The elements of $\mathcal{L}$ are called *branch labels* (or simply *labels*). Every branch $b \in \mathcal{B}$ is assigned a label $l(b) \in \mathcal{L}$. In principle, any branch labeling convention is possible, but in practice the labeling is chosen to be descriptive of the input/output process that is modeled by the trellis. For example, a typical labeling convention is to label the branches as in Figure 1-(2): the branch corresponding to a pair of states $(s_{t-1}, s_t) \in \mathcal{S}_{t-1} \times \mathcal{S}_t$ is labeled by $x_t/c_{t1}c_{t2}$ (here, x_t is the input to the convolutional encoder and the pair (c_{t1}, c_{t2}) is the output of the encoder). We may represent the label set as the union of N label sets $\mathcal{L}_t$, that is, $\mathcal{L} = \bigcup_{t=1}^{N} \mathcal{L}_t$. Thereby, if $b \in \mathcal{B}_t$, then $l(b) \in \mathcal{L}_t$. Note that there may *not* be a 1-to-1 correspondence between the elements of $\mathcal{B}_t$ and the elements of $\mathcal{L}_t$. Indeed, from Figure 1-(2), we observe that $\mathcal{B}_3$ contains 8 elements (8 branches), but $\mathcal{L}_3 = \{0/00, 0/01, 1/10, 1/11\}$ contains only 4 labels.

For the purpose of this paper, we also assume the following:

5) For any time $1 \le t \le N$, the set $\mathcal{L}_t$ is partitioned into M disjoint subsets, $\mathcal{L}_t = \bigcup_{m=0}^{M-1} \mathcal{L}_t^{(m)}$. This partitioning may be chosen arbitrarily, but typically we assign a meaning to the partitioning. For example, consider Figure 1-(2), where the labels correspond to the input/output values of the encoder. At time $t = 3$, we have that the set of labels is $\mathcal{L}_3 = \{0/00, 0/01, 1/10, 1/11\}$. Typically, we partition the set $\mathcal{L}_3$ into disjoint subsets based on the value of the input symbol. Since in the example in Figure 1 we have $M = 2$ possible input symbols, 0 and 1, we partition $\mathcal{L}_3$ into $M = 2$ disjoint subsets $\mathcal{L}_3^{(0)} = \{0/00, 0/01\}$ and $\mathcal{L}_3^{(1)} = \{1/10, 1/11\}$.

Throughout this paper, when we refer to a state at time t, we mean a state $s \in \mathcal{S}_t$. Similarly, a branch at time t is a branch $b \in \mathcal{B}_t$ and a label at time t is a label $l \in \mathcal{L}_t$. A *path* through a trellis from time i to time j is defined as a sequence of concatenated branches $b_{i+1}, b_{i+2}, \ldots, b_j$ (denoted by $\underline{p}_i^j$), where $b_t = (s_{t-1}, l_t, s_t) \in \mathcal{B}_t$, and $i < t \le j$. In words, the ending state of b_t is the same as the starting state of b_{t+1} for $i < t < j$. We call s_i and s_j the starting state (denoted by $\sigma^-(\underline{p}_i^j)$) and the ending state (denoted by $\sigma^+(\underline{p}_i^j)$) of the given path $\underline{p}_i^j$, respectively. The *length* of the given path $\underline{p}_i^j$ is defined as $j - i$. In particular, a branch $b \in \mathcal{B}$ is considered to be a path of length one. A state $s \in \mathcal{S}$ is considered to be a path (denoted by $1_s = s$) of length zero with the *null* label and the same starting and ending state s.

A *trellis section* from time i to time j is defined as the collection of all the paths $\underline{p}_i^j$ of length $j - i$ from time i to time j, which is denoted by $T_i^j(*, *, *)$. In this paper, we are interested in the following subsets of $T_i^j(*, *, *)$:

1) $T_i^j(*, *, s_j)$ is the subset in which every path ends with the same state $s_j \in \mathcal{S}_j$.
2) $T_i^j(s_i, *, *)$ is the subset in which every path starts with the same state $s_i \in \mathcal{S}_i$.
3) Given $i < t \le j$, we denote by $T_i^j(*, \mathcal{L}_t^{(m)}, *)$ the subset of paths whose branches at time t take on labels from the subset $\mathcal{L}_t^{(m)}$.
4) Given $i < t \le j$, we denote by $T_i^j(*, \mathcal{L}_t^{(m)}, s_j)$ the subset of paths whose branches at time t take on labels from the subset $\mathcal{L}_t^{(m)}$ and whose ending states are the same state $s_j \in \mathcal{S}_j$.

We also introduce symbolic graphs to represent these subsets, see Figure 2. Note that in Figure 2, in the symbolic graphs for $T_i^j(*, \mathcal{L}_t^{(m)}, *)$ and $T_i^j(*, \mathcal{L}_t^{(m)}, s_j)$, we substitute the symbol $\mathcal{L}_t^{(m)}$ with just m. This is justified in the context of the symbolic graph because in the symbolic graph (see Figure 2) the symbol m is sandwiched between the identifiers $t - 1$ and t, which uniquely identifies the subset $\mathcal{L}_t^{(m)}$.

B. Trellis semiring

A *semiring* is represented by a 3-tuple $(\mathcal{R}, \oplus, \otimes)$, where $\mathcal{R}$ is a nonempty set, and $\oplus$ and $\otimes$ are two binary operations over $\mathcal{R}$. That is, $\oplus$ and $\otimes$ are two mappings from $\mathcal{R} \times \mathcal{R}$ to $\mathcal{R}$. The semiring $(\mathcal{R}, \oplus, \otimes)$ satisfies the following axioms[4]:

- $(\mathcal{R}, \oplus)$ is a *commutative monoid*; that is, the following properties hold:
  1) **Associativity**: $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ for all $a, b, c \in \mathcal{R}$.
  2) **Identity element**: there exists an element (denoted by $1_\oplus$) in $\mathcal{R}$ such that $a \oplus 1_\oplus = 1_\oplus \oplus a = a$ for all $a \in \mathcal{R}$.
  3) **Commutativity**: $a \oplus b = b \oplus a$ for all $a, b \in \mathcal{R}$.
- $(\mathcal{R}, \otimes)$ is a *monoid*; that is, the following properties hold:
  4) **Associativity**: $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ for all $a, b, c \in \mathcal{R}$.
  5) **Identity element**: there exists an element (denoted by $1_\otimes$) in $\mathcal{R}$ such that $a \otimes 1_\otimes = 1_\otimes \otimes a = a$ for all $a \in \mathcal{R}$.
- The operation $\otimes$ (called *multiplication* or *product*) has the *distributive* property with respect to the operation $\oplus$ (called *addition* or *sum*); that is, the following property holds[5]:
  6) **Distributivity**: $(a \oplus b) \otimes c = a \otimes c \oplus b \otimes c$ and $c \otimes (a \oplus b) = c \otimes a \oplus c \otimes b$ for all $a, b, c \in \mathcal{R}$.

Similar to [14] [15] [16] [28], we will define a *trellis semiring* as follows. Let $\mathcal{P} = \bigcup_{0 \le i \le j \le N} T_i^j(*, *, *)$. In words, $\mathcal{P}$ consists of all paths (in general, these paths have different lengths) in the time interval $[0, N]$. Let $\mathcal{R} = 2^{\mathcal{P}}$, where $2^{\mathcal{P}}$ represents the set of all the subsets of $\mathcal{P}$. In other words, P is an element in $\mathcal{R}$ if and only if P is a subset of $\mathcal{P}$. For any two path subsets

[4] This definition is slightly different from that in [27], where a commutative semiring is defined, and the existence of the identities are not required.

[5] As usual, we make an assumption that the priority of the multiplication $\otimes$ is higher than that of the addition $\oplus$.

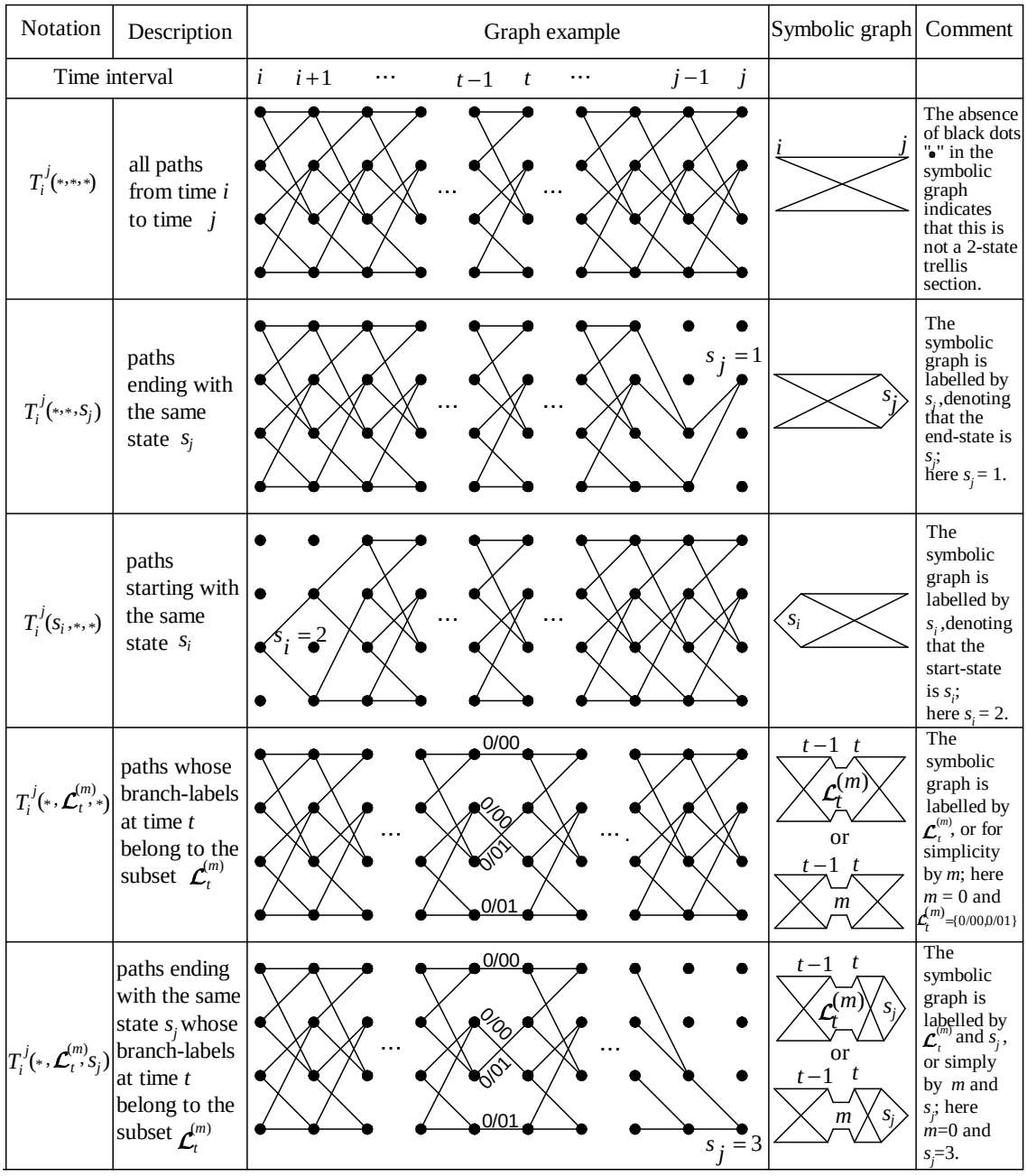

Fig. 2. Some basic notation and symbolic graph representations.

$P \in \mathcal{R}$ and $Q \in \mathcal{R}$, their sum is defined as their *collection* (union)

$$P \oplus Q \stackrel{\text{def}}{=} P \cup Q. \tag{1}$$

To define the product operation, we first define the product of two paths $\underline{p}$ and $\underline{q}$ as their *concatenation*. Let $\underline{p} = \underline{p}_i^j$ be a path of concatenated branches $b_{i+1}, b_{i+2}, \ldots, b_j$, and let $\underline{q} = \underline{q}_k^\ell$ be a path of concatenated branches $b_{k+1}, b_{k+2}, \ldots, b_\ell$. Then, we define the product of $\underline{p}$ and $\underline{q}$ as

$$\underline{p} \odot \underline{q} \stackrel{\text{def}}{=} \begin{cases} b_{i+1}, b_{i+2}, \ldots, b_j, b_{k+1}, b_{k+2}, \ldots, b_\ell; \\ \qquad \text{if } j = k \text{ and } \sigma^+(\underline{p}_i^j) = \sigma^-(\underline{q}_k^\ell) \\ \text{empty set } \emptyset; \text{ otherwise} \end{cases} . \tag{2}$$

The product of two path subsets $P \in \mathcal{R}$ and $Q \in \mathcal{R}$ is now defined as the set of concatenated paths $\underline{p} \odot \underline{q}$, where $p \in P$ and $q \in Q$, i.e.,

$$P \otimes Q \stackrel{\text{def}}{=} \{\underline{p} \odot \underline{q} \mid \underline{p} \in P, \underline{q} \in Q\}. \tag{3}$$

Note that the multiplication $\otimes$ is not commutative and that the

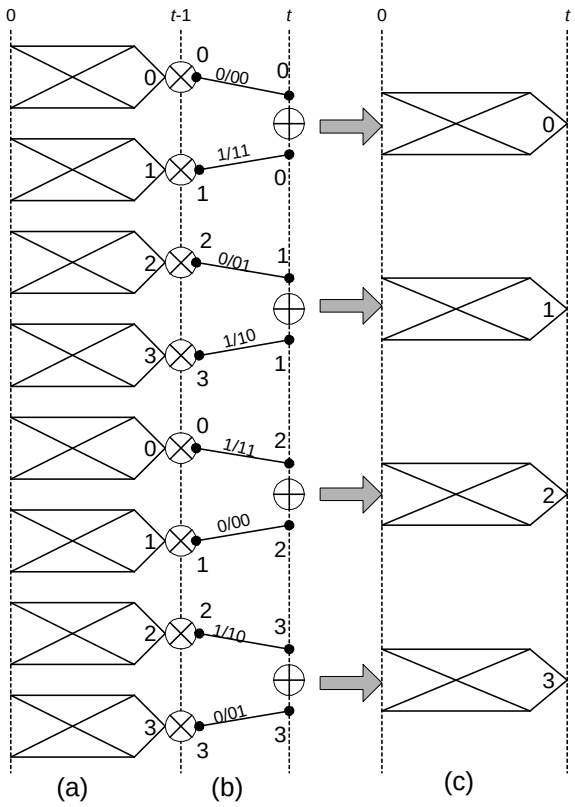

Fig. 3. The forward-backward algorithm. Symbolic description of the forward recursion, equation (6), taking Figure 1-(2) as an example. (a) $T_0^{t-1}(*,*,s_{t-1})$; (b) $b \in \mathcal{B}_t$; (c) $T_0^t(*,*,s_t)$.

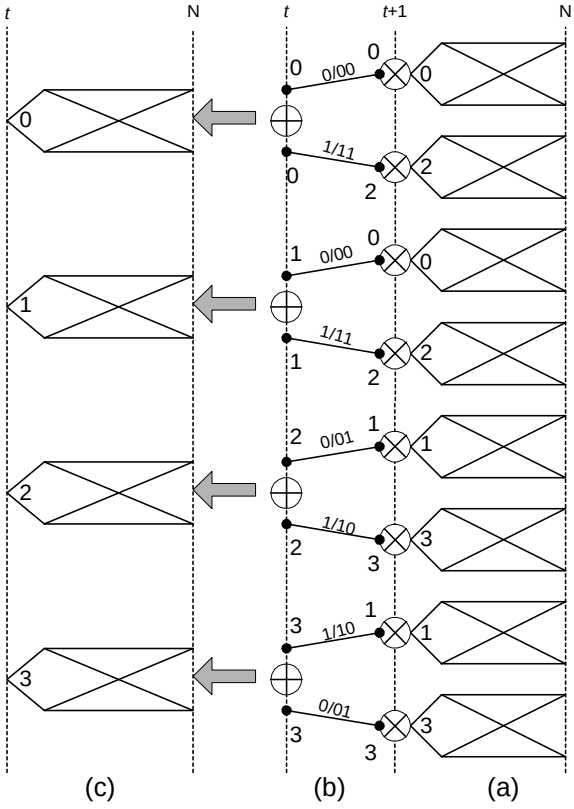

Fig. 4. The forward-backward algorithm. Symbolic description of the backward recursion, equation (7), taking Figure 1-(2) as an example. (a) $T_{t+1}^N(s_{t+1},*,*)$; (b) $b \in \mathcal{B}_{t+1}$; (c) $T_t^N(s_t,*,*)$.

product $P \otimes Q$ may be empty $\emptyset$. The operands related to $\oplus$ and $\otimes$ are sets; however, if a set contains only one element, we will simply use the element itself to replace the set. For example, we can write a path $\underline{p}_i^j$ from time i to time j as $\underline{p}_i^j = b_{i+1} \otimes b_{i+2} \otimes \ldots \otimes b_j$.

It can be verified that $(\mathcal{R}, \oplus, \otimes)$ is a semiring. The identity element under addition is the empty set $1_\oplus = \emptyset$ and the identity element under multiplication is $1_\otimes = \{1_s \mid s \in \mathcal{S}\}$, the set of all paths of length zero, which is equivalent to the set of all states (without the branches). The defined semiring makes it possible to describe some relationships using mathematical equations, for example, we can write

$$T_0^N(*,\mathcal{L}_t^{(m)},*) = \bigoplus_{\substack{b\in\mathcal{B}_t:\\ l(b)\in\mathcal{L}_t^{(m)}}} T_0^{t-1}(*,*,\sigma^-(b)) \otimes b \otimes T_t^N(\sigma^+(b),*,*) \tag{4}$$

or

$$T_0^N(*,\mathcal{L}_t^{(m)},*) = \bigoplus_{s_N\in\mathcal{S}_N} T_0^N(*,\mathcal{L}_t^{(m)},s_N) \tag{5}$$

for $1 \le t \le N$, and $0 \le m \le M-1$. These two equations, (4) and (5), are illustrated next with graphical interpretations.

C. Path-partitioning algorithms and their graphical representations

Consider the **mid-constraint path-partitioning problem:** For all $1 \le t \le N$ and all $0 \le m \le M-1$, how can we determine the subsets $T_0^N(*,\mathcal{L}_t^{(m)},*)$? It seems that this is not a hard problem: by definition, a path $\underline{p}_0^N$ is classified into $T_0^N(*,\mathcal{L}_t^{(m)},*)$ whenever the label of its branch at time t is an element in $\mathcal{L}_t^{(m)}$. But this means that we would have to check this condition for all the paths in $T_0^N(*,*,*)$. In general, such an exhaustive search is efficient only for $N = 1$. A natural question to ask is whether we can do better than an exhaustive search.

One algorithm to solve the mid-constraint path-partitioning problem is the *forward-backward path-partitioning algorithm*. Equation (4) suggests that we may first determine the subsets $T_0^{t-1}(*,*,s_{t-1})$ and $T_t^N(s_t,*,*)$, whereby these subsets can be determined recursively, and then perform the collection operation $\oplus$.

The forward-backward path-partitioning algorithm:

1) **Forward recursion** (Figure 3): For $t = 1, 2, \ldots, N$, for all $s_t \in \mathcal{S}_t$,

$$T_0^t(*,*,s_t) = \bigoplus_{b\in\mathcal{B}_t:\sigma^+(b)=s_t} T_0^{t-1}(*,*,\sigma^-(b)) \otimes b. \tag{6}$$

2) **Backward recursion** (Figure 4): For $t = N-1, N-2, \ldots, 0$, for all $s_t \in \mathcal{S}_t$,

$$T_t^N(s_t,*,*) = \bigoplus_{b\in\mathcal{B}_{t+1}:\sigma^-(b)=s_t} b \otimes T_{t+1}^N(\sigma^+(b),*,*). \tag{7}$$

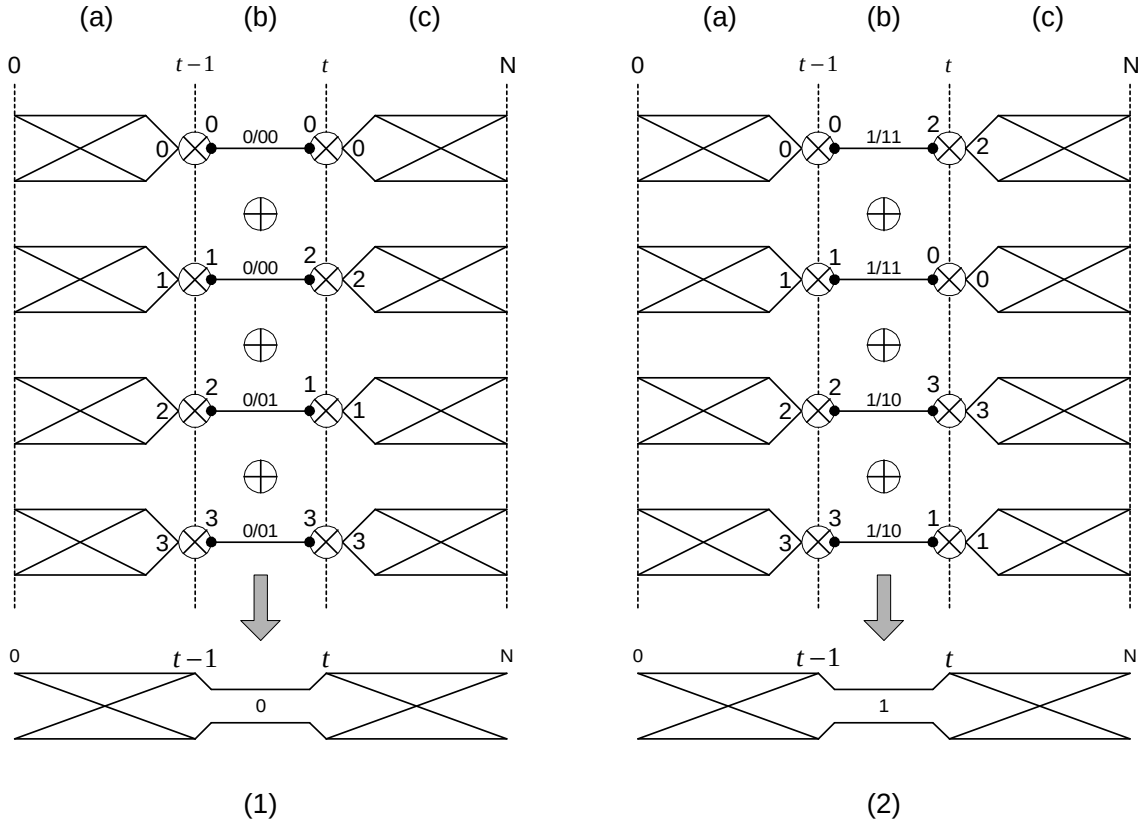

Fig. 5. The forward-backward algorithm. Symbolic description of the mid-constraint extraction, equation (4) or (8), taking Figure 1-(2) as an example. (a) $T_0^{t-1}(*,*,s_{t-1})$; (b) $b \in \mathcal{B}_t$; (c) $T_t^N(s_t,*,*)$. Part (1) outputs $T_0^N(*,\mathcal{L}_t^{(0)},*)$ and Part (2) outputs $T_0^N(*,\mathcal{L}_t^{(1)},*)$.

3) **Mid-constraint extraction** (Figure 5): For $1 \le t \le N$, $0 \le m \le M-1$,

$$T_0^N(*,\mathcal{L}_t^{(m)},*) = \bigoplus_{\substack{b\in\mathcal{B}_t:\\ l(b)\in\mathcal{L}_t^{(m)}}} T_0^{t-1}(*,*,\sigma^-(b)) \otimes b \otimes T_t^N(\sigma^+(b),*,*). \tag{8}$$

We next describe the *forward-only path-partitioning algorithm*. Equation (5) suggests that we may first determine the subsets $T_0^N(*,\mathcal{L}_t^{(m)},s_N)$, whereby these subsets can be determined recursively. Assume that we have obtained all the subsets $T_0^{t-1}(*,\mathcal{L}_i^{(m)},s_{t-1})$ for all $1 \le i \le t-1$ and all $s_{t-1} \in \mathcal{S}_{t-1}$. When we reach time t, we update these subsets to get $T_0^t(*,\mathcal{L}_i^{(m)},s_t)$ for all $1 \le i \le t$ and all $s_t \in \mathcal{S}_t$. The algorithm is summarized as follows.

The forward-only path-partitioning algorithm:

1) **Extend** (Figure 6): Extend the partitions to time t. That is, for all $s_{t-1} \in \mathcal{S}_{t-1}$,

$$T_0^{t-1}(*,*,s_{t-1}) = \bigoplus_{0\le m\le M-1} T_0^{t-1}(*,\mathcal{L}_{t-1}^{(m)},s_{t-1}), \tag{9}$$

and for all $s_t \in \mathcal{S}_t$ and all $0 \le m \le M-1$,

$$T_0^t(*,\mathcal{L}_t^{(m)},s_t) = \bigoplus_{\substack{b\in\mathcal{B}_t: l(b)\in\mathcal{L}_t^{(m)},\\ \sigma^+(b)=s_t}} T_0^{t-1}(*,*,\sigma^-(b)) \otimes b. \tag{10}$$

2) **Update** (Figure 7): Update the prior mid-constraint partitions to include time t. That is, for all $s_t \in \mathcal{S}_t$, $1 \le i \le t-1$ and $0 \le m \le M-1$,

$$T_0^t(*,\mathcal{L}_i^{(m)},s_t) = \bigoplus_{b\in\mathcal{B}_t:\sigma^+(b)=s_t} T_0^{t-1}(*,\mathcal{L}_i^{(m)},\sigma^-(b)) \otimes b. \tag{11}$$

3) **Collect** (Figure 8): At time N, we will get $T_0^N(*,\mathcal{L}_i^{(m)},s_N)$ for all $1 \le i \le N$, $0 \le m \le M-1$, $s_N \in \mathcal{S}_N$. If $|\mathcal{S}_N| = 1$, we have the desired partitions $T_0^N(*,\mathcal{L}_t^{(m)},*)$; otherwise, we obtain the desired partitions by

$$T_0^N(*,\mathcal{L}_t^{(m)},*) = \bigoplus_{s_N\in\mathcal{S}_N} T_0^N(*,\mathcal{L}_t^{(m)},s_N) \tag{12}$$

for all $1 \le t \le N$ and $0 \le m \le M-1$.

It should be pointed out that, in the described forward-backward and forward-only algorithms, there are only two basic operations, $\otimes$ and $\oplus$, which correspond to *concatenating* and *collecting*, respectively. These two algorithms are applications of the sum-product algorithm [18] (or the generalized

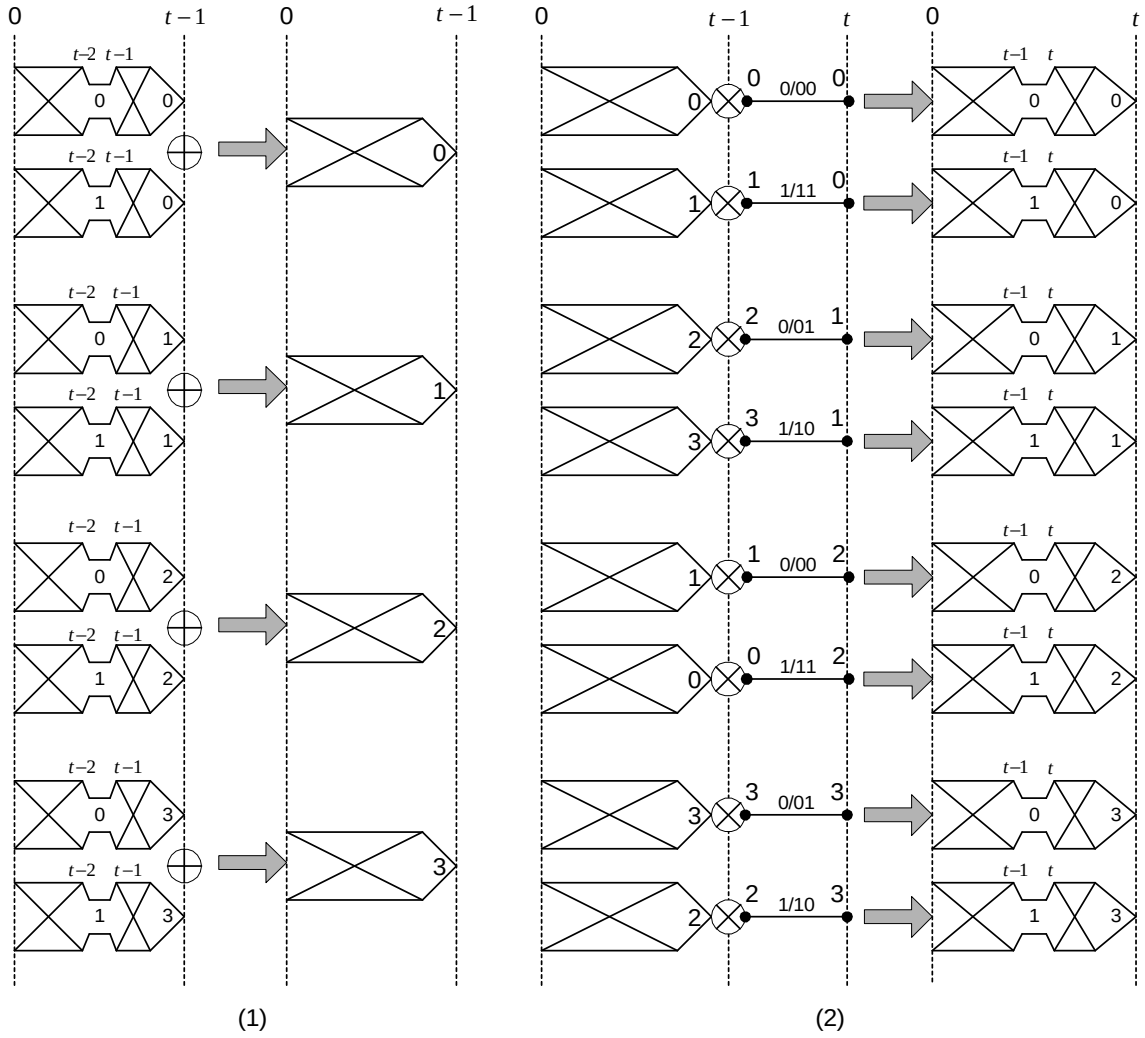

Fig. 6. The forward-only algorithm. Symbolic description of the "extend" operation, equations (9) and (10), taking Figure 1-(2) as an example. Part (1) corresponds to equation (9). Part (2) corresponds to equation (10).

distributive law [16]) to the trellis semiring. Also note that the initialization is obtained from the definition of the zero-length path.

III. The a Posteriori Probability (APP) Algorithms

A. The probabilistic model

Let $\mathcal{X}_0, \mathcal{X}_1, \ldots$ be a sequence of sets. A sequence $(x_i, x_{i+1}, \ldots, x_{j-1}, x_j)$, where $x_t \in \mathcal{X}_t$ for $i \leq t \leq j$, is denoted by $\underline{x}_i^j$. When no confusion can arise, $\underline{x}_1^N$ is simply denoted by $\underline{x}$. A *chance variable*[6] is denoted by an uppercase character, while its realization is denoted by a lowercase character (e.g., a chance sequence $\underline{X}$ has a realization $\underline{x}$). For simplicity, the notation $\mathrm{P}_{\underline{X}}(\underline{x})$ is used to represent the probability mass function if $\underline{X}$ is a discrete chance sequence and also to represent the probability density function if $\underline{X}$ is a continuous (real) random sequence. Similarly, $\mathrm{P}_{\underline{X}|\underline{Y}}(\underline{x}|\underline{y})$ is used to represent the conditional probability mass function or the conditional probability density function.

[6] A chance variable is very much like a random variable except that it may take on non-numeric values.

In decoding/detection applications, we typically denote the trellis labels at time t $(1 \leq t \leq N)$ in the "standard" form[7] u_t/c_t, where $u_t \in \{0, 1, \ldots, M-1\}$ is the input, and c_t is the noiseless output that is determined by the starting and ending states of the branch and the input u_t. Any path $\underline{p}_0^N$ from time $t = 0$ to $t = N$ corresponds to a sequence $\underline{c} = (c_1, c_2, \ldots, c_N)$. Assume that $\underline{c} = (c_1, c_2, \ldots, c_N)$ is "transmitted". The received sequence is $\underline{y} = (y_1, y_2, \ldots, y_N)$ which equals to $\underline{c}$ corrupted by noise.

In the sequel, we assume the following probabilistic model.

1) The considered trellis is assumed to represent a Markov process, which is characterized as follows. At time $t = 0$, the process starts from state $s_0 \in \mathcal{S}_0$ with the probability $\mathrm{P}_{S_0}(s_0)$, where $\mathrm{P}_{S_0}(s_0)$ is known[8]. If at time $t-1$ ($t > 0$), the process is in the state $s_{t-1} \in \mathcal{S}_{t-1}$, then at time

[7] For some trellises, for example, those that represent linear block codes, we can write the labels in a trivial "standard" form u_t/c_t with $u_t = c_t$.

[8] If we have no knowledge of the initial states, we can make the assumption that $\mathrm{P}_{S_0}(s_0) = 1/|\mathcal{S}_0|$ for any $s_0 \in \mathcal{S}_0$.

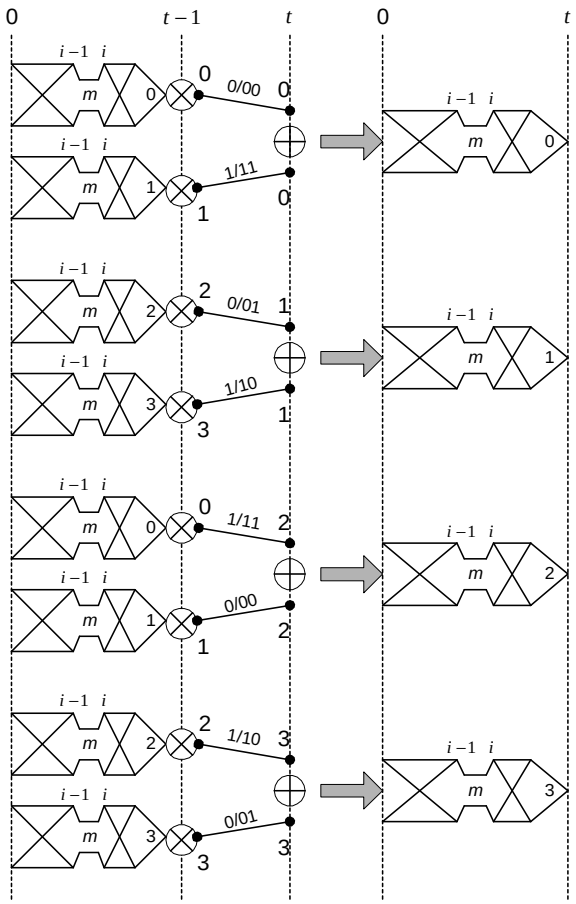

Fig. 7. The forward-only algorithm. Symbolic description of the "update" operation, equation (11), taking Figure 1-(2) as an example. Here, $m \in \{0, 1\}$.

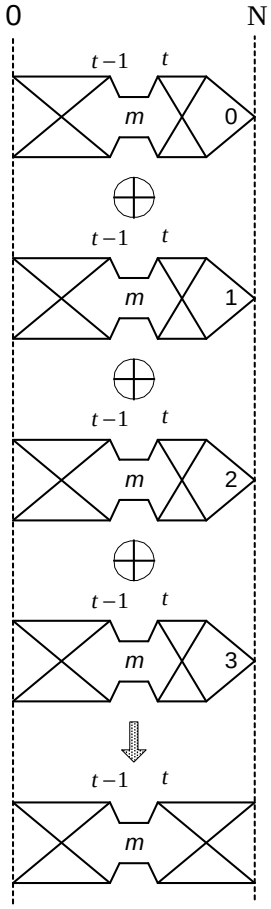

Fig. 8. The forward-only algorithm. Symbolic description of the "collect" operation, equation (5) or (12), taking Figure 1-(2) as an example. Here, $m \in \{0, 1\}$.

t ($t > 0$), the process randomly generates a label $l_t \in \mathcal{L}_t$ and goes into a state $s_t \in \mathcal{S}_t$ such that (s_{t-1}, l_t, s_t) is an element in $\mathcal{B}_t$. In other words, the process generates a branch at time t according to a certain probability distribution function. Obviously, when the process runs from time $t = 0$ to $t = N$, the output is a path through the trellis from time 0 to time N. The Markov property says that the probability $\mathrm{P}_{B_t|\underline{P}_0^{t-1}}(b_t|\underline{p}_0^{t-1})$ of generating a branch $b_t \in \mathcal{B}_t$ at time t, given the generated path up to time $t-1$, depends only on the ending state of the generated path:

$$\mathrm{P}_{B_t|\underline{P}_0^{t-1}}(b_t|\underline{p}_0^{t-1}) = \mathrm{P}_{B_t|S_{t-1}}(b_t|\sigma^+(\underline{p}_0^{t-1})). \quad (13)$$

So, the probability of a given path $\underline{p}_0^N = b_1 \otimes b_2 \otimes \ldots \otimes b_N$ factors as

$$\mathrm{P}_{\underline{P}_0^N}(\underline{p}_0^N) = \mathrm{P}_{S_0}(\sigma^-(b_1)) \prod_{t=1}^{N} \mathrm{P}_{B_t|S_{t-1}}(b_t|\sigma^-(b_t)), \quad (14)$$

where $\mathrm{P}_{B_t|S_{t-1}}(b_t|s_{t-1})$ is assumed to be known for all $1 \le t \le N$.

2) The noise is assumed to be memoryless; that is,

$$\mathrm{P}_{\underline{Y}|\underline{C}}(\underline{y}|\underline{c}) = \prod_{t=1}^{N} \mathrm{P}_{Y_t|C_t}(y_t|c_t), \quad (15)$$

where $\mathrm{P}_{Y_t|C_t}(y_t|c_t)$ is assumed to be known for all $1 \le t \le N$.

B. The APP problem

Suppose that we are given the vector of channel output observations $\underline{y}$. The APP problem is to find the *a posteriori* probabilities $\mathrm{P}_{U_t|\underline{Y}}(u_t = m|\underline{y})$ for all $1 \le t \le N$ and all $0 \le m \le M-1$.

For simplicity, we make an assumption that $|\mathcal{S}_0| = 1$. For $1 \le t \le N$, to a given branch $b_t = (s_{t-1},\ u_t/c_t,\ s_t)$ we assign a *branch metric* $\Gamma[b_t] \stackrel{\mathrm{def}}{=} \mathrm{P}_{B_t|S_{t-1}}(b_t|s_{t-1})\mathrm{P}_{Y_t|C_t}(y_t|c_t)$. For a path $\underline{p}_i^j = b_{i+1} \otimes b_{i+2} \otimes \ldots \otimes b_j$, we define the *path metric* as

$$\Gamma[\underline{p}_i^j] \stackrel{\mathrm{def}}{=} \Gamma[b_{i+1}]\Gamma[b_{i+2}] \ldots \Gamma[b_j]. \quad (16)$$

Note that the definition of $\Gamma[\underline{p}_0^N]$ should include one more factor $\mathrm{P}_{S_0}(\sigma^-(\underline{p}_0^N))$ to reflect the initial probability if $|\mathcal{S}_0| > 1$. For

a nonempty subset $P \subseteq \mathcal{P}$, we define the *set metric* as $\Gamma[P] \stackrel{\text{def}}{=} \sum_{\underline{p} \in P} \Gamma[\underline{p}]$. In particular, $\Gamma[\emptyset] \stackrel{\text{def}}{=} 0$. From the Bayesian rule, and equations (14) and (15), we have the APPs

$$\begin{aligned} \mathrm{P}_{U_t|\underline{Y}}(u_t = m|\underline{y}) &= \frac{\Gamma[T_0^N(*, \mathcal{L}_t^{(m)}, *)]}{\Gamma[T_0^N(*,*,*)]} \\ &= \frac{\Gamma[T_0^N(*, \mathcal{L}_t^{(m)}, *)]}{\sum\limits_{0 \le i \le M-1} \Gamma[T_0^N(*, \mathcal{L}_t^{(i)}, *)]}. \end{aligned} \quad (17)$$

With this notation, the APP problem is equivalent to finding $\Gamma[T_0^N(*, \mathcal{L}_t^{(m)}, *)]$ for all $1 \le t \le N$ and all $0 \le m \le M-1$.

Loosely stated, for a subset $P = \bigoplus \bigotimes_k b_k$ (collection of concatenations of branches in the trellis domain), we have $\Gamma[P] = \sum \prod_k \Gamma[b_k]$ (sum of products of branch-metrics in the probability domain). So, we can decompose the set-metric (global sum of products) efficiently into local recursive expressions since the corresponding subsets can be constructed recursively, which leads to efficient algorithms. Rigorously stated, we have the following two lemmas.

Lemma 1 (Preservation of associativity and commutativity): If P and Q are two disjoint subsets, that is, $P \cap Q = \emptyset$, then $\Gamma[P \oplus Q] = \Gamma[P] + \Gamma[Q]$.

Lemma 2 (Preservation of distributivity): If P and Q are concatenated, that is, if $P \subseteq T_i^j(*, *, s_j)$ and $Q \subseteq T_j^k(s_j, *, *)$, then $\Gamma[P \otimes Q] = \Gamma[P] \times \Gamma[Q]$.

Here, $+$ stands for the ordinary real addition, and $\times$ stands for the ordinary real multiplication. These two lemmas show that the metric of one complex subset can be computed from metrics of simpler subsets.

Given Lemmas 1 and 2, it may be tempting to conclude that the mapping $\Gamma : \mathcal{R} \to \mathbb{R}$ is a homomorphism of semirings. Recall that a homomorphism needs to satisfy (for any $P \in \mathcal{R}$ and any $Q \in \mathcal{R}$)

$$\Gamma[P \oplus Q] = \Gamma[P] + \Gamma[Q] \quad (18)$$

$$\Gamma[P \otimes Q] = \Gamma[P] \times \Gamma[Q] \quad (19)$$

$$\Gamma[1_{\otimes}] = 1. \quad (20)$$

Lemma 1 shows that requirement (18) is satisfied under the restrictive condition $P \cap Q = \emptyset$; similarly Lemma 2 shows that requirement (19) is satisfied under the restrictive condition that the ending state of P equals the starting state of Q. Because of these restrictive conditions, the mapping $\Gamma : \mathcal{R} \to \mathbb{R}$ is *not* a homomorphism, though it shows some resemblances of a homomorphic mapping.

C. The forward-backward APP algorithm

Combining Lemma 1 and Lemma 2 with the forward-backward path-partitioning algorithm, we get the forward-backward APP algorithm, which is also called the BCJR algorithm [4].

The forward-backward APP algorithm (BCJR algorithm):

1) **Forward recursion**: For $s_0 \in \mathcal{S}_0$, set $\Gamma[T_0^0(*,*,s_0)] = \mathrm{P}_{S_0}(s_0)$. For $t = 1, 2 \ldots, N-1$, for all $s_t \in \mathcal{S}_t$, compute

$$\Gamma[T_0^t(*,*,s_t)] = \sum_{b \in \mathcal{B}_t : \sigma^+(b) = s_t} \Gamma[T_0^{t-1}(*,*,\sigma^-(b))] \times \Gamma[b]. \quad (21)$$

2) **Backward recursion**: For $s_N \in \mathcal{S}_N$, set $\Gamma[T_N^N(s_N,*,*)] = 1$. For $t = N-1, N-2, \ldots, 1$, for all $s_t \in \mathcal{S}_t$, compute

$$\Gamma[T_t^N(s_t,*,*)] = \sum_{b \in \mathcal{B}_{t+1} : \sigma^-(b) = s_t} \Gamma[b] \times \Gamma[T_{t+1}^N(\sigma^+(b),*,*)]. \quad (22)$$

3) **Mid-constraint extraction**: For $1 \le t \le N$, for $0 \le m \le M-1$, compute

$$\Gamma[T_0^N(*, \mathcal{L}_t^{(m)}, *)] = \sum_{\substack{b \in \mathcal{B}_t : \\ l(b) \in \mathcal{L}_t^{(m)}}} \Gamma[T_0^{t-1}(*,*,\sigma^-(b))] \times \Gamma[b] \times \Gamma[T_t^N(\sigma^+(b),*,*)]. \quad (23)$$

Using these metrics $\Gamma[T_0^N(*, \mathcal{L}_t^{(m)}, *)]$ and equation (17), we can find the APPs.

From above, we see that the BCJR algorithm can be obtained from the forward-backward path-partitioning algorithm by replacing the subsets with their metrics, the operation $\otimes$ (path-concatenation) with $\times$ (real multiplication), and the operation $\oplus$ (path-collection) with $+$ (real addition).

D. The forward-only APP algorithm

Based on the forward-only path-partitioning algorithm described in Section II, if we replace the subsets with their metrics, $\oplus$ with real addition +, and $\otimes$ with real multiplication $\times$, we can obtain the forward-only APP algorithm, which is equivalent to the algorithms in [21], [22], [24]. The important feature is, similar to the Viterbi algorithm [2], that the forward-only APP algorithm stores M real-sequences (called *soft survivors*) for each state at each time. At time t, for a given state $s_t \in \mathcal{S}_t$, the m-th ($0 \le m \le M-1$) soft survivor is a collection of t real numbers $\Gamma[T_0^t(*, \mathcal{L}_i^{(m)}, s_t)]$ (for $1 \le i \le t$). As in the Viterbi algorithm [2], after receiving y_{t+1}, these soft survivors are extended and updated. More precisely, we have

The forward-only APP algorithm:

0) **Initialization**: For $s_0 \in \mathcal{S}_0$, set $\Gamma[T_0^0(*, \mathcal{L}_0^{(0)}, s_0)] = \mathrm{P}_{S_0}(s_0)$ and $\Gamma[T_0^0(*, \mathcal{L}_0^{(m)}, s_0)] = 0$ for all $1 \le m \le M-1$.

For $t = 1, 2, \ldots, N$:

1) **Extend the soft survivors to time** t: For all $s_{t-1} \in \mathcal{S}_{t-1}$, compute

$$\Gamma[T_0^{t-1}(*,*,s_{t-1})] = \sum_{0 \le m \le M-1} \Gamma[T_0^{t-1}(*, \mathcal{L}_{t-1}^{(m)}, s_{t-1})]. \quad (24)$$

For all $s_t \in \mathcal{S}_t$ and all $0 \le m \le M-1$, compute

$$\Gamma[T_0^t(*, \mathcal{L}_t^{(m)}, s_t)] = \sum_{\substack{b \in \mathcal{B}_t : l(b) \in \mathcal{L}_t^{(m)}, \\ \sigma^+(b) = s_t}} \Gamma[T_0^{t-1}(*,*,\sigma^-(b))] \times \Gamma[b]. \quad (25)$$

2) **Update the soft survivors before time** t: For $s_t \in \mathcal{S}_t$, $1 \le i \le t-1$ and $0 \le m \le M-1$, compute

$$\Gamma[T_0^t(*, \mathcal{L}_i^{(m)}, s_t)] = \sum_{\substack{b \in \mathcal{B}_t: \\ \sigma^+(b)=s_t}} \Gamma[T_0^{t-1}(*, \mathcal{L}_i^{(m)}, \sigma^-(b))] \times \Gamma[b]. \tag{26}$$

Note that the updates in (26) can be performed in parallel. That is, at time t, if we have $t-1$ processors (one processor for each time slot i, where $1 \le i \le t-1$), then we can perform the updates simultaneously.

3) **Collect**: After time N, we obtain $\Gamma[T_0^N(*, \mathcal{L}_i^{(m)}, s_N)]$ for all $1 \le i \le N$, $0 \le m \le M-1$, $s_N \in \mathcal{S}_N$. Hence, we can compute

$$\Gamma[T_0^N(*, \mathcal{L}_t^{(m)}, *)] = \sum_{s_N \in \mathcal{S}_N} \Gamma[T_0^N(*, \mathcal{L}_t^{(m)}, s_N)] \tag{27}$$

for all $1 \le t \le N$ and $0 \le m \le M-1$. The APPs can be found using equation (17).

E. Complexity comparison

Up to this point, our derivations applied to any trellis (regular or irregular). We conduct the computational complexity analysis also based on a general trellis. Naturally, the complexity may be reduced by considering the special structure of a given trellis, which is beyond our task here. When deriving the complexity, we do not consider the computational complexity of the branch metric computation because this is the same for any APP algorithm.

Memory requirements:

The forward-backward APP algorithm needs $\sum_{0 \le t \le N-1} |\mathcal{S}_t| = |\mathcal{S}| - |\mathcal{S}_N|$ memory units to store all the results of the forward recursions (21). It also needs $\sum_{1 \le t \le N} |\mathcal{B}_t| = |\mathcal{B}|$ memory units to store all the branch metrics[9]. However, we do not have to store all the results of the backward recursions (22) since we can alternatively perform the backward recursions (22) and the mid-constraint extractions (23).

Before time t, the forward-only APP algorithm needs $|\mathcal{S}_{t-1}| \cdot M \cdot (t-1)$ memory units to store all the soft survivors. So the maximal memory requirement is not greater than $M \cdot N \cdot \max_{0 \le t \le N} |\mathcal{S}_t|$.

Computational complexity:

From equations (21)-(27), we see that the dominant term of the computational complexity is determined by the number of branches. For the forward-backward algorithm, the forward recursions (21) require $\sum_{1 \le t \le N-1} |\mathcal{B}_t| = |\mathcal{B}| - |\mathcal{B}_N|$ multiplications and $\sum_{1 \le t \le N-1} (|\mathcal{B}_t| - |\mathcal{S}_t|) = (|\mathcal{B}| - |\mathcal{B}_N|) - (|\mathcal{S}| - |\mathcal{S}_N|)$ additions; the backward recursions (22) need $\sum_{N > t \ge 2} |\mathcal{B}_t| = |\mathcal{B}| - |\mathcal{B}_1|$ multiplications and $\sum_{N > t \ge 2} (|\mathcal{B}_t| - |\mathcal{S}_t|) = (|\mathcal{B}| - |\mathcal{B}_1|) - (|\mathcal{S}| - |\mathcal{S}_1|)$ additions; the mid-constraint extractions (23) need $\sum_{1 \le t \le N} (2 \cdot |\mathcal{B}_t|) = 2 \cdot |\mathcal{B}|$ multiplications and $\sum_{1 \le t \le N} (|\mathcal{B}_t| - M) = |\mathcal{B}| - MN$ additions.

At time t, the forward-only APP algorithm needs $|\mathcal{S}_{t-1}| \cdot (M-1)$ additions for equation (24), and $|\mathcal{B}_t|$ multiplications and $|\mathcal{B}_t| - M$ additions for equation (25); to update the survivors before time t using equation (26), we need $|\mathcal{B}_t| \cdot M \cdot (t-1)$ multiplications and $(|\mathcal{B}_t| - |\mathcal{S}_t|) \cdot M \cdot (t-1)$ additions; equation (27) needs $(|\mathcal{S}_N| - 1) \cdot M \cdot N$ additions. Over N trellis steps, this amounts to approximately $|\mathcal{B}| \cdot M(N-1)/2$ multiplications and $(|\mathcal{B}| - |\mathcal{S}|) \cdot M(N-1)/2 - MN$ additions.

The results are listed in Table I, where we did not include the complexity of the computations in equation (17). It is clear from Table I that, either in terms of memory requirements or in terms of computational complexity, the forward-only algorithm does not offer an advantage over the forward-backward algorithm. However, as hinted in the discussion after equation (26), the forward-only algorithm has two advantages:

1) Ability to operate in real time; see, e.g., [21]. That is, if we update the soft survivors simultaneously at time t, then we can compute the APPs $\mathrm{P}_{U_i|\underline{Y}_1^t}(u_i|\underline{y}_1^t)$ almost instantaneously after receiving the observation y_t.
2) Potential for parallelization, as we demonstrate on an example in Section IV-B.

IV. Approximations of the APP Algorithms

A. The SW-BCJR algorithm and the L/HCR/LVS algorithm

To reduce the decoding/detection delay, and also to reduce the memory requirement, we can utilize the following approximation

$$\begin{aligned} \mathrm{P}_{U_t|\underline{Y}_1^N}(u_t = m|\underline{y}_1^N) &\approx \mathrm{P}_{U_t|\underline{Y}_1^{t+D}}(u_t = m|\underline{y}_1^{t+D}) \\ &= \frac{\Gamma[T_0^{t+D}(*, \mathcal{L}_t^{(m)}, *)]}{\Gamma[T_0^{t+D}(*, *, *)]}, \\ &= \frac{\Gamma[T_0^{t+D}(*, \mathcal{L}_t^{(m)}, *)]}{\sum_{0 \le i \le M-1} \Gamma[T_0^{t+D}(*, \mathcal{L}_t^{(i)}, *)]}, \end{aligned} \tag{28}$$

where the integer $D > 0$ denotes a predetermined fixed delay. If the trellis length is infinite, for example, see [30], such a "truncation" is necessary. In other words, to get the approximate APPs at time t, we only utilize the trellis section $T_0^{t+D}(*, *, *)$ and the corresponding received sub-vector $\underline{y}_1^{t+D}$.

The suboptimal fixed-delay versions of the forward-backward APP algorithms have been developed by Benedetto *et al.* [29] and Viterbi [31][10]. These are called sliding-window

[9] If there exist several branches that share the same metric, we can reduce the memory requirement. Another way to reduce the memory is to store the received sequence and to recompute the branch metric for the backward recursions, which requires a higher computational load.

[10] In strict sense, the algorithm in [31] is different from the algorithm in [29], since in [31] the delay is periodically variant for different time instants.

TABLE I

THE COMPLEXITY COMPARISON – FULL ALGORITHMS

	memory	multiplications	additions
forward-backward	$\approx \lvert\mathcal{S}\rvert + \lvert\mathcal{B}\rvert$	$\approx 4\lvert\mathcal{B}\rvert$	$\approx 3\lvert\mathcal{B}\rvert - 2\lvert\mathcal{S}\rvert - MN$
forward-only	$\approx \lvert\mathcal{S}\rvert M$	$\approx \lvert\mathcal{B}\rvert M(N-1)/2$	$\approx (\lvert\mathcal{B}\rvert - \lvert\mathcal{S}\rvert)M(N-1)/2 - MN$

TABLE II

THE COMPLEXITY COMPARISON – TRUNCATED ALGORITHMS

	memory	multiplications	additions	delay
SW-BCJR in [29]	$S + B(D+1)$	$B(D+3)$	$B(D+2) - S(D+1) - M$	$\geq D$
L/HCR/LVS	$SM(D+1)$	$BM(D+1)$	$BMD - SM(D-2) - 2M$	1

BCJR (SW-BCJR) algorithms. It should be pointed out that the sliding-window idea seems to have been known prior to the advent of turbo codes to researchers familiar with the BCJR algorithm; see e.g., Loeliger [32]. The suboptimal fixed-delay version of the forward-only APP algorithm has been developed by Lee [21], by Hayes, Cover and Riera [22], and also by Li, Vucetic and Sato [24]. We call this algorithm the L/HCR/LVS algorithm.

In the following, we will compare the complexity of the L/HCR/LVS algorithm to that of the SW-BCJR algorithm proposed by Benedetto *et al.* [29]. Thereby, we assume that the trellis has the time-invariant structure, so that we can use shorter notation $S \stackrel{\text{def}}{=} |\mathcal{S}_t|$ and $B \stackrel{\text{def}}{=} |\mathcal{B}_t|$.

Computational complexity and memory requirements for SW-BCJR: Note that the results of the complexity analysis cannot be derived directly from Table I. The SW-BCJR algorithm in [29] needs B multiplications and $B - S$ additions to compute the metrics $\Gamma[T_0^{t-1}(*, *, s_{t-1})]$ from $\Gamma[T_0^{t-2}(*, *, s_{t-2})]$ (which constitutes one step of the forward recursion). These metrics need to be stored, which requires S memory units. The SW-BCJR algorithm also requires $(1+D)B$ memory units to store the branch metrics from time t to $t + D$. Then the SW-BCJR algorithm performs D steps of the backward recursion to compute the metrics $\Gamma[T_t^{t+D}(s_t, *, *)]$, which requires BD multiplications and $(B - S)D$ additions. Finally, the mid-constraint extractions require $2B$ multiplications and $B - M$ additions. Note that we do not have to store the intermediate metrics $\Gamma[T_i^{t+D}(s_i, *, *)]$ for $t < i < t + D$.

Computational complexity and memory requirements for L/HCR/LVS: The L/HCR/LVS algorithm requires a memory array of size $(1 + D)SM$ to store the latest $(1 + D)$-long soft survivors $\Gamma[T_0^{t-1+D}(*, \mathcal{L}_i^{(m)}, s_{t-1+D})]$ for $t - 1 \leq i \leq t - 1 + D$, $0 \leq m \leq M - 1$, and $s_{t-1+D} \in \mathcal{S}_{t-1+D}$. After receiving y_{t+D}, the L/HCR/LVS algorithm delivers the metrics $\Gamma[T_0^{t-1+D}(*, \mathcal{L}_{t-1}^{(m)}, *)]$, which requires $(S - 1)M$ additions. Using the branch metrics at time $t + D$, the L/HCR/LVS algorithm updates all these metrics to obtain $\Gamma[T_0^{t+D}(*, \mathcal{L}_i^{(m)}, s_{t+D})]$ for $t \leq i \leq t + D$, $0 \leq m \leq M - 1$, and $s_{t+D} \in \mathcal{S}_{t+D}$, which requires $BM(1+D)$ multiplications, and $S(M - 1) + S - M + (B - S)MD$ additions.

The comparison results are listed in Table II, where the number of multiplications/additions that constitute the computational loads are given *per input symbol*. The delay represents the waiting time required to deliver the soft output $\mathrm{P}_{U_t|\underline{Y}_1^{t+D}}(u_t|\underline{y}_1^{t+D})$ for the input variable u_t after receiving y_{t+D}. Note that the actual delay is related to the speed of the processors.

In terms of complexity alone, the L/HCR/LVS algorithm has a clear disadvantage by a factor of M. However, since it can be implemented in parallel (which requires D processors), the APPs $\mathrm{P}_{U_t|\underline{Y}_1^{t+D}}(u_t|\underline{y}_1^{t+D})$ can be delivered almost instantaneously after receiving y_{t+D}. On the other hand, the SW-BCJR algorithms have to wait for at least D-steps of the backward recursions.

B. A decoding example

In this subsection, we illustrate the structure of the forward-only APP algorithm with a fixed-delay constraint for the convolutional code depicted in Figure 1. Note that the convolutional encoder in Figure 1 is *recursive*, so the forward-only APP algorithm is *not* equivalent to the algorithms in Lee [21], Hayes, Cover and Riera [22] and Li Vucetic and Sato [24], which were all derived for *non-recursive* finite-state machines. However, throughout this subsection, we neglect this subtle difference, and refer to the forward-only APP algorithm for decoding the code in Figure 1 as the L/HCR/LVS algorithm.

At time $t = 0$, the state of the encoder is set to zero. The input to the encoder is a binary sequence $\underline{u} = (u_1, u_2, \ldots, u_t, \ldots)$, where $u_t \in \{0, 1\}$ is the output from a memoryless source at time t. The output from the encoder is denoted by $\underline{c} = (c_1, c_2, \ldots, c_t, \ldots)$, where $c_t = (c_{t1}, c_{t2}) \in \{00, 01, 10, 11\}$. The corresponding trellis graph is given in Figure 1-(2), where N is sufficiently large. The label set $\mathcal{L}_t$ at time t can be divided into two disjoint subsets according to the input u_t, i.e., $\mathcal{L}_t^{(0)} = \{u_t/c_t \mid u_t = 0\}$, and $\mathcal{L}_t^{(1)} = \{u_t/c_t \mid u_t = 1\}$.

Assume that $\underline{c}$ is transmitted through a memoryless channel and $\underline{y} = (y_1, y_2, \ldots, y_t, \ldots)$ is the received sequence. Also assume that the probability mass (or density) functions $\mathrm{P}_{U_t}(\cdot)$ and $\mathrm{P}_{Y_t|C_t}(\cdot|\cdot)$ are known. To any given branch $b_t \in \mathcal{B}_t$ with a label u_t/c_t, assign a metric $\Gamma[b_t] = \mathrm{P}_{U_t}(u_t) \cdot \mathrm{P}_{Y_t|C_t}(y_t|c_t)$. Since c_t determines u_t in this example, there are only four different branch metrics at time t once y_t is given. We denote these metrics by m_{c_t}. So, at every trellis stage, we will need to

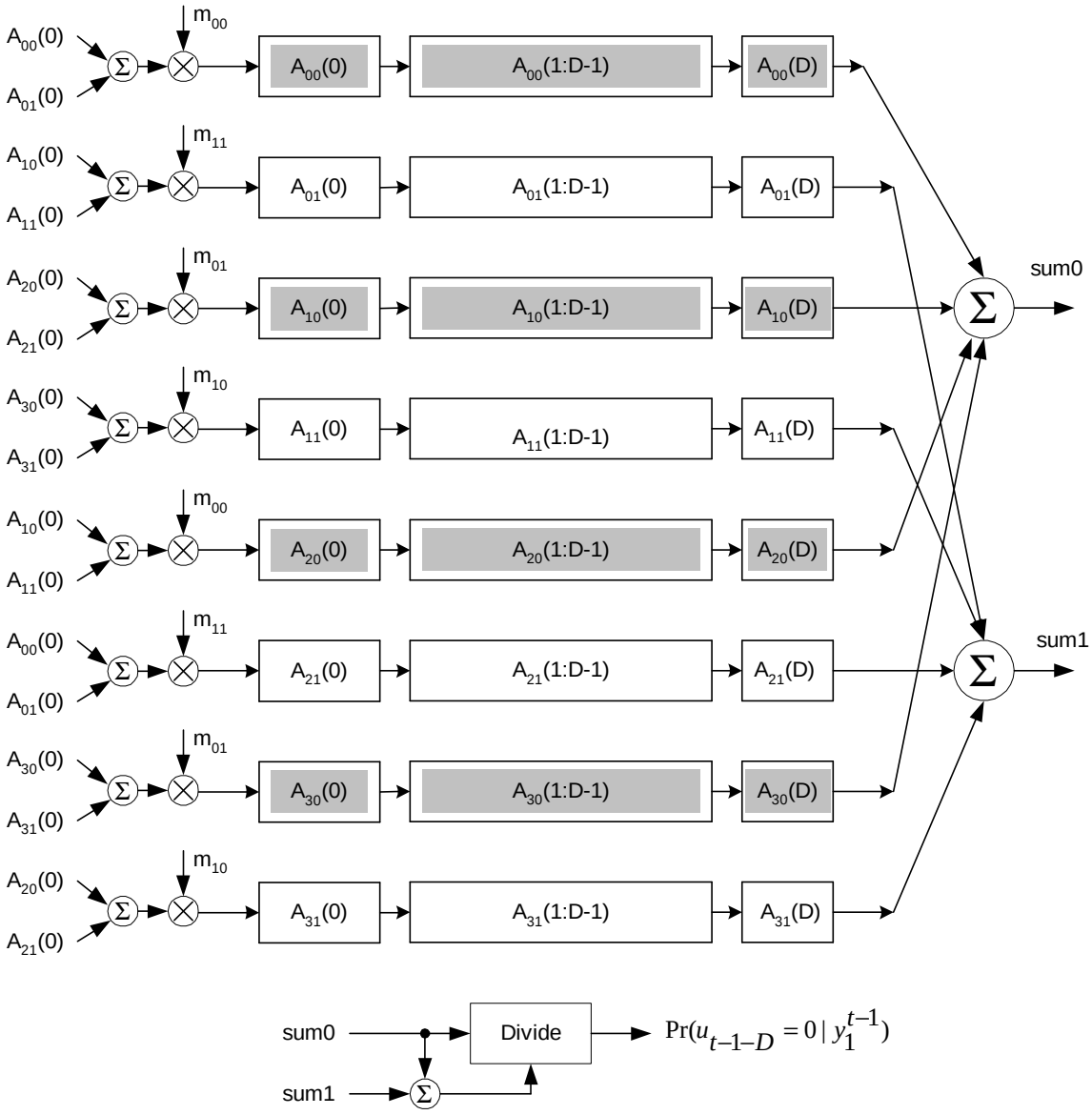

Fig. 9. The second step of the forward-only APP algorithm.

compute 4 branch metrics m_{00}, m_{01}, m_{10} and m_{11}.

For a fixed delay D, we utilize the L/HCR/LVS algorithm to calculate $\Gamma[T_0^{t+D}(*, \mathcal{L}_t^{(m)}, *)]$, where $m \in \{0, 1\}$, and then utilize equation (28) to compute the APPs $\mathrm{P}_{U_t|\underline{Y}_1^{t+D}}(u_t|\underline{y}_1^{t+D})$. Utilizing these APPs, we make a decision $u_t = 0$ if $\mathrm{P}_{U_t|\underline{Y}}(u_t = 0|\underline{y}) > 0.5$; otherwise we decide $u_t = 1$. The APPs can also be used to implement other recursive (or iterative) decoding/detection algorithms, for example, see [33].

Very similar to the Viterbi algorithm, the L/HCR/LVS algorithm is an updating algorithm that stores the *soft* survivors and recursively updates them. For the example in Figure 1, the L/HCR/LVS algorithm requires $8(D+1)$ memory units. At any time t, for each state $s_t \in S_t$, we store two soft survivors (real-valued sequences) of length $D+1$, shown in Figure 9. For $s \in \mathcal{S}$, the soft survivors corresponding to state s are denoted by A_{s0} and A_{s1}, where at time t, the i-th ($0 \le i \le D$) terms of A_{s0} and A_{s1} are (see Figure 9)

$$A_{s0}(i) = \Gamma[T_0^t(*, \mathcal{L}_{t-i}^{(0)}, s)], \tag{29}$$

and

$$A_{s1}(i) = \Gamma[T_0^t(*, \mathcal{L}_{t-i}^{(1)}, s)]. \tag{30}$$

The L/HCR/LVS algorithm:

Initialization: At time $t = 0$, for $0 \le i \le D$, set $A_{sm}(i) = 1$ if $s = 0$ and $m = 0$; otherwise set $A_{sm}(i) = 0$.

Recursion: At time $t > 0$, after receiving y_t, the forward-only APP algorithm performs the following three steps:

1) First, compute the 4 branch metrics m_{00}, m_{01}, m_{10} and m_{11}.
2) Then, as shown in Figure 9, the soft survivors are shifted to the right by one term. At the same time, the first terms $A_{sm}(0)$ of the survivors are generated according to equations (24) and (25). Concurrently, from the last terms

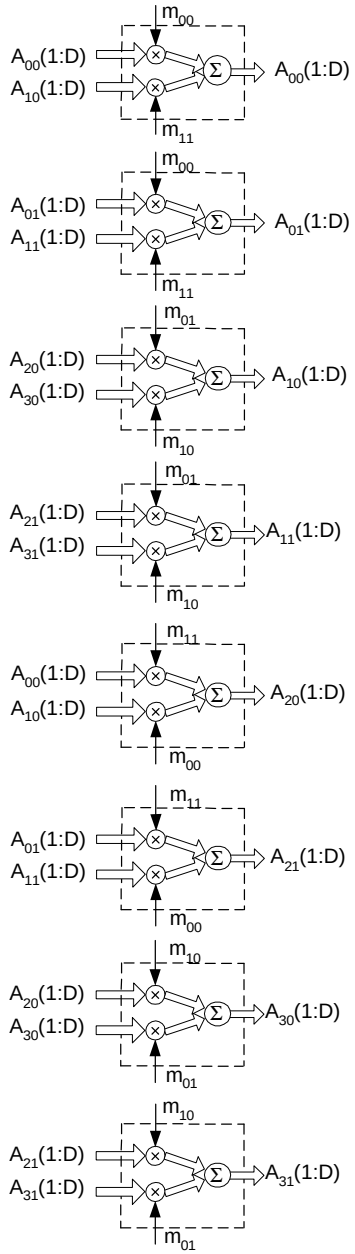

Fig. 10. The third step of the forward-only APP algorithm.

$A_{sm}(D)$ of the survivors, we get

$$\Pr(u_{t-1-D}=0|\underline{y}_1^{t-1}) = \frac{\sum\limits_{0\le s\le 3} A_{s0}(D)}{\sum\limits_{0\le s\le 3} A_{s0}(D) + \sum\limits_{0\le s\le 3} A_{s1}(D)}. \tag{31}$$

3) Finally, the interiors $A_{sm}(i)$ (for $1 \le i \le D$) of the survivors are updated in parallel, where the basic operation block is a linear combiner; see Figure 10.

The L/HCR/LVS algorithm is a stream-oriented algorithm that delivers its outputs in real time, just as the Viterbi algorithm. The inputs to the algorithm are the observations

$$\cdots,\ y_t,\ y_{t+1},\ y_{t+2},\ \cdots$$

which are continuously arriving. Given these inputs, the Viterbi algorithm [2] would deliver the hard decisions

$$\cdots,\ \hat{u}_{t-D},\ \hat{u}_{t+1-D},\ \hat{u}_{t+2-D},\ \cdots$$

in real time with a delay D. Similarly, the L/HCR/LVS delivers the soft decisions

$$\cdots,\ \tilde{u}_{t-D},\ \tilde{u}_{t+1-D},\ \tilde{u}_{t+2-D},\ \cdots$$

in real time with a delay D, where $\tilde{u}_{t-D} = \Pr(u_{t-D}=0|\underline{y}_1^t)$. The probabilities of the complementary input symbols are obtained as $\Pr(u_{t-D}=1|\underline{y}_1^t) = 1 - \Pr(u_{t-D}=0|\underline{y}_1^t)$.

C. The Max-Log BCJR algorithm, SOVA and SSA

We can consider the fixed-delay truncation described in Section IV-B as a technique to reduce the complexity in the horizontal (time) direction. To reduce the complexity in the vertical (state) direction, we can utilize the following approximation

$$\mathrm{P}_{U_t|\underline{Y}_1^N}(u_t = m|\underline{y}_1^N) \approx \frac{\Gamma[P]}{\Gamma[P]+\Gamma[Q]}, \tag{32}$$

where $P \subseteq T_0^N(*, \mathcal{L}_t^{(0)}, *)$ and $Q \subseteq T_0^N(*, \mathcal{L}_t^{(1)}, *)$ are two "dominant" subsets. In other words, we utilize smaller subsets to replace larger subsets to reduce the complexity. For example, if we select $P \subseteq T_0^N(*, \mathcal{L}_t^{(0)}, *)$ to be (strictly speaking, the subset that contains) the path with the maximum metric, and $Q \subseteq T_0^N(*, \mathcal{L}_t^{(1)}, *)$ to be the path with the maximum metric, we can perform the algorithms (in log-domain) by using the Max-Log-BCJR algorithm [33], [34], [35] or by using the modified soft-output Viterbi algorithm (SOVA) [36], which is a forward-only algorithm. Their equivalence was proved in [36]. It should be pointed out that an algorithm similar to the modified SOVA has also been proposed in [24], where the algorithm was called a suboptimum soft-output algorithm (SSA). All these suboptimal forward-only algorithms have a unified representation as versions of the min-sum algorithm in [10][11]. For other reduced complexity BCJR algorithms [37], it is also possible to derive the corresponding forward-only APP algorithms based on the path-partitioning algorithms presented in Section II.

D. Some implementation issues

To describe the forward-only APP algorithm clearly, we did not represent the forward-only algorithm in its most efficient memory-requirement form. Since some stored sequences are dependent, one can reduce the memory by increasing the computational load. It should also be pointed out that, to make the APP algorithms stable, normalizations for the intermediate variables are required for large N, which does not affect the APPs because equation (17) is in a fractional form. Note that we did not include the normalizations in the complexity analysis of either Table I or Table II.

In practical systems, we can perform the APP algorithms in the log-domain, for example, see [34][35]. We may change the metric $\Gamma[\cdot]$ to $\ln\Gamma[\cdot]$ and map the path-concatenation $\otimes$ to

$\otimes_{\log}$ and the path-collection $\oplus$ to $\oplus_{\log}$. Here, for any two real numbers a and b, we define $a \otimes_{\log} b \stackrel{\text{def}}{=} a + b$ and $a \oplus_{\log} b \stackrel{\text{def}}{=} \ln(e^a + e^b) = \max(a,b) + \ln(1 + e^{-|a-b|})$. In this case, we can get the complexity analysis by using the facts that 1 multiplication in the probability domain corresponds to 1 addition in the log-domain, and 1 addition in the probability domain corresponds to 2 additions, 1 compare-select, and 1 table-look-up in the log-domain. Another implementation issue for digital computations is that we have to quantize the intermediate variables, for example see [35][38].

V. Conclusion

In this paper, we have shown that the trellis-based decoding/detection problems can be transformed into trellis-based path-partitioning problems[11]. We further showed that the trellis-based path-partitioning problems can be solved recursively in a forward-only manner by performing two basic operations: path-collection and path-concatenation (which respectively play the roles of the "sum" and the "product"). By creating different mappings from the path-subsets to set metrics, we rederived several existing decoding/detection algorithms. In particular, we rederived the forward-only APP algorithm proposed by Lee [21], by Hayes *et al.* [22] and by Li *et al.* [24]. The method presented here is slightly different from that in [21], [22], [24] since it can be applied to any trellis, while the algorithm in [21] is specialized to non-recursive convolutional codes and the algorithms in [22], [24] are specialized to intersymbol interference (ISI) channels. Another difference is that we intentionally slightly increased the memory requirement in order to arrive at a version that we believe is easier to understand (and adapt to different tasks) and has a lower (and parallelizable) computational load. In practice, one can make a trade-off between the memory-requirement and the computational complexity. It should be pointed out that all the presented derivations here can be obtained by slightly modifying the definition of the *flow* in [14] [15]. In fact, one path is itself an ingredient of some (sub)flow. However, we think that the method based on the path-partitioning algorithms is more intuitive.

Acknowledgments

The authors wish to thank the reviewers for extremely helpful comments and for pointing out reference [22]; they also wish to thank Dr. Bao-Ming Bai for providing reference [21] and Dr. Ping Li for invaluable suggestions.

Appendix: The Viterbi Algorithms

In this Appendix, we show how the Viterbi algorithm [2], [3] may be derived from a forward-recursive path-partitioning algorithm. We also derive the less familiar parallel list Viterbi algorithm [2], [26].

[11] We believe that we can do this transformation for most other (non-decoding/non-detection) trellis-based problems, one example being the capacity computation [39].

Utilizing the APP algorithm, we can choose $\hat{u}_t (1 \le t \le N)$ such that

$$\mathrm{P}_{U_t|\underline{Y}}(\hat{u}_t|\underline{y}) = \max_{u_t \in \{0,1,\ldots,M-1\}} \mathrm{P}_{U_t|\underline{Y}}(u_t|\underline{y}). \tag{33}$$

This method is called *maximum a posteriori probability (MAP) symbol decoding/detection*, which is optimal in the sense that the symbol error probability is minimized.

Another commonly used optimality criterion is to select one path $\hat{\underline{p}} \in T_0^N(*,*,*)$ such that

$$\mathrm{P}_{\underline{P}|\underline{Y}}(\hat{\underline{p}}|\underline{y}) = \max_{\underline{p} \in T_0^N(*,*,*)} \mathrm{P}_{\underline{P}|\underline{Y}}(\underline{p}|\underline{y}), \tag{34}$$

which is called *MAP sequence decoding/detection* [2], [3]. When all paths are equiprobable, such a decoder/detector is equivalent to the well-known *maximum-likelihood decoder (MLD)/maximum-likelihood sequence detector (MLSD)* [2], [3], which is implemented by the Viterbi algorithm.

For simplicity, we make the assumption that $|\mathcal{S}_0| = 1$. For $1 \le t \le N$, to a given branch $b_t = (s_{t-1},\ u_t/c_t,\ s_t) \in \mathcal{B}_t$ assign a *branch metric* $\gamma[b_t] \stackrel{\text{def}}{=} -\ln \mathrm{P}_{B_t|S_{t-1}}(b_t|s_{t-1}) - \ln \mathrm{P}_{Y_t|C_t}(y_t|c_t)$. For a path $\underline{p}_i^j = b_{i+1} \otimes b_{i+2} \otimes \ldots \otimes b_j$, define the *path metric*

$$\gamma[\underline{p}_i^j] \stackrel{\text{def}}{=} \gamma[b_{i+1}] + \gamma[b_{i+2}] + \ldots + \gamma[b_j]. \tag{35}$$

For a nonempty subset $P \subseteq \mathcal{P}$ (recall that $\mathcal{P}$ consists of all the paths in a trellis), define the *Viterbi set metric* as the pair[12]

$$\Gamma[P] \stackrel{\text{def}}{=} (\hat{\underline{p}}, \gamma[\hat{\underline{p}}]), \tag{36}$$

where $\hat{\underline{p}} \in P$ is a path that satisfies $\gamma[\hat{\underline{p}}] = \min_{\underline{p} \in P} \gamma[\underline{p}]$. If there exists more than one such path, we arbitrarily choose only one of them. We call $\hat{\underline{p}}$ the *survivor* of P. In particular, define $\Gamma[\emptyset] \stackrel{\text{def}}{=} (\emptyset, \infty)$.

It can be verified that the MAP sequence decoding/detection problem is equivalent to finding the Viterbi set metric of $T_0^N(*,*,*)$. The Viterbi algorithm is an efficient method to solve such a problem. The key step is to develop a recursive method to find the Viterbi set metric of one complex subset from the metrics of several simpler subsets. For two subsets $P \subseteq \mathcal{P}, Q \subseteq \mathcal{P}$, assume that $\Gamma[P] = (\hat{\underline{p}}, \gamma[\hat{\underline{p}}])$ and $\Gamma[Q] = (\hat{\underline{q}}, \gamma[\hat{\underline{q}}])$. Define the following two operations

$$\Gamma[P] \tilde{\oplus} \Gamma[Q] \stackrel{\text{def}}{=} \Gamma[P \oplus Q] \tag{37}$$

and

$$\Gamma[P] \tilde{\otimes} \Gamma[Q] \stackrel{\text{def}}{=} \Gamma[P \otimes Q]. \tag{38}$$

We have the following two lemmas

Lemma 3 (Compare-Select): If $\gamma[\hat{\underline{p}}] \le \gamma[\hat{\underline{q}}]$, then $\Gamma[P] \tilde{\oplus} \Gamma[Q] = \Gamma[P]$; else $\Gamma[P] \tilde{\oplus} \Gamma[Q] = \Gamma[Q]$.

[12] This is a somewhat unconventional definition of the Viterbi metric, where the metric consists of both the survivor path $\hat{\underline{p}}$ and the path metric $\gamma[\hat{\underline{p}}]$.

Lemma 4 (Add-Concatenate): If $\sigma^+(\underline{\hat{p}}) = \sigma^-(\underline{\hat{q}})$, then $\Gamma[P]\tilde{\otimes}\Gamma[Q] = (\underline{\hat{p}} \otimes \underline{\hat{q}}, \gamma[\underline{\hat{p}}] + \gamma[\underline{\hat{q}}])$.

Armed with these two lemmas, we can rederive the Viterbi algorithm as a forward-recursive path-partitioning algorithm. We simply use the forward recursion, equation (6), of the forward-backward algorithm by replacing the subsets by their Viterbi set metrics, the operation $\oplus$ (collection) by $\tilde{\oplus}$ (compare-select), and the operation $\otimes$ (concatenation) by $\tilde{\otimes}$ (add-concatenate). More precisely, we get the following algorithm

The Viterbi algorithm:

1) **Initialization**: For $s_0 \in \mathcal{S}_0$, set

$$\Gamma[T_0^0(*,*,s_0)] = (1_{s_0}, -\ln \mathrm{P}_{S_0}(s_0)). \tag{39}$$

2) **Forward Recursion**: For $t = 1, 2, \ldots, N$, for all $s_t \in \mathcal{S}_t$,

$$\Gamma[T_0^t(*,*,s_t)] = \tilde{\bigoplus_{b \in \mathcal{B}_t : \sigma^+(b) = s_t}} \Gamma[T_0^{t-1}(*,*,\sigma^-(b))]\tilde{\otimes}\Gamma[b]. \tag{40}$$

3) **Decision**: At time N, we have

$$\Gamma[T_0^N(*,*,*)] = \tilde{\bigoplus_{s_N \in \mathcal{S}_N}} \Gamma[T_0^N(*,*,s_N)]. \tag{41}$$

The Viterbi algorithm performs two basic operations over the Viterbi metrics: $\tilde{\oplus}$ (compare-select) and $\tilde{\otimes}$ (add-concatenate). The main memory requirement is to store the survivors, which is proportional to the trellis length N. To reduce the memory requirement, and also to reduce the decoding/detection delay, the truncated (fixed-delay) Viterbi algorithm is widely used in practical systems [2] [3].

The list Viterbi algorithm:

Utilizing a similar approach, we can derive the parallel list Viterbi algorithm [2] [26]. The objective is to list L best (that is, L lowest metric) paths, which are called *candidates*. First, for any $P \subseteq \mathcal{P}$, define the *list Viterbi set metric* as

$$\Gamma[P] = \begin{pmatrix} \underline{p}^{(1)} & \gamma[\underline{p}^{(1)}] \\ \underline{p}^{(2)} & \gamma[\underline{p}^{(2)}] \\ \vdots & \vdots \\ \underline{p}^{(L')} & \gamma[\underline{p}^{(L')}] \end{pmatrix}. \tag{42}$$

where $L' = \min(|P|, L)$ and $p^{(i)}$ (for $1 \le i \le L'$) are L' different paths in P with the lowest metrics, $\gamma[\underline{p}^{(1)}] \le \gamma[\underline{p}^{(2)}] \le \ldots \le \gamma[\underline{p}^{(L')}]$. Now the problem is transformed into finding the list Viterbi set metric of $T_0^N(*,*,*)$. Second, define the "sum" and "product" operations and derive the corresponding "compare-select" and "add-concatenate" lemmas. Finally, replace the subsets in the forward path-partitioning algorithm, equation (6), with their list Viterbi set metrics, and perform the corresponding operations.

REFERENCES

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

[2] G. D. Forney Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, March 1973.

[3] G. D. Forney Jr., "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Transactions on Information Theory*, vol. 18, pp. 363–378, March 1972.

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

[5] R. W. Chang and J. C. Hancock, "On receiver structures for channels having memory," *IEEE Trans. Inform. Theory*, vol. 12, pp. 463–468, Oct. 1966.

[6] L. E. Baum and T. Petrie, "Statistical interference for probabilistic functions of finite state Markov chains," *Ann. Math. Statist.*, vol. 37, pp. 1559–1563, 1966.

[7] L. Rabiner, "A tutorial on hidden Markov models and selected areas in speech recognition," *Proc. IEEE*, vol. 77, pp. 257–286, Feb. 1989.

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

[9] B. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. 27, pp. 533–547, September 1981.

[10] N. Wiberg, H.-A. Loeliger, and R. Kötter, "Codes and iterative decoding on general graphs," *Eoropean Trans. on Telecommun.*, vol. 6, pp. 513–526, September 1995.

[11] N. Wiberg, *Codes and Decoding on General Graphs*. PhD thesis, Linköping University, Linköping, Sweden, April 1996.

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

[13] J. Hagenauer and L. Papke, "Decoding "turbo"-codes with the soft output Viterbi algorithm (SOVA)," in *Proc. IEEE Int. Symp. Inform. Theory*, (Trondheim, Norway), p. 164, June 27-July 1, 1994.

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

[15] C. Heegard and S. B. Wicker, *Turbo Coding*. Boston: Kluwer Academic, 1999.

[16] S. M. Aji and R. J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol. 46, pp. 325–343, Mar. 2000.

[17] J. Bajcsy and H. Kobayashi, "On obtaining the BCJR algorithm via path counting," in *Proc. IEEE Int. Symp. Inform. Theory and Its Application*, (Hawaii), November 2000.

[18] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, pp. 498–519, Feb. 2001.

[19] G. D. Forney Jr., "Codes on graphs: Normal realizations," *IEEE Trans. Inform. Theory*, vol. 47, pp. 520–548, February 2001.

[20] R. J. McEliece, D. J. C. Mackay, and J. F. Cheng, "Turbo decoding as an instance of Pearl's "belief propagation" algortihm," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 140–152, February 1998.

[21] L.-N. Lee, "Real-time minimal-bit-error probability decoding of convolutional codes," *IEEE Trans. Commun.*, vol. Com-22, pp. 146–151, Feb. 1974.

[22] J. F. Hayes, T. M. Cover, and J. B. Riera, "Optimal sequence detection and optimal symbol-by-symbol detection: Similar algorithms," *IEEE Transactions on Communications*, vol. 30, pp. 152–157, January 1982.

[23] K. Abend and B. D. Fritchman, "Statistical detection for communication channels with intersymbol interference," *Proc. IEEE*, vol. 58, pp. 779–785, May 1970.

[24] Y. Li, B. Vucetic, and Y. Sato, "Optimum soft-output detection for channels with intersymbol interference," *IEEE Trans. Inform. Theory*, vol. 41, pp. 704–713, May 1995.

[25] B. Bai, X. Ma, and X. Wang, "Novel algorithm for continuous decoding of turbo codes," *IEE Proc. Commun.*, vol. 46, pp. 314–315, October 1999.

[26] N. Seshadri and C.-E. W. Sundberg, "List Viterbi decoding algorithms with applications," *IEEE Trans. Commun.*, vol. 42, pp. 313–323, Feb./Mar./April 1994.

[27] M. Schmidt and G. P. Fettweis, "On memory redundancy in the BCJR algorithm for nonrecursive shift register processes," *IEEE Trans. Inform. Theory*, vol. 46, pp. 1580–1584, July 2000.

[28] A. Vardy, "Trellis structure of codes," in *Handbook of Coding Theory* (V. S. Pless and W. C. Huffman, eds.), vol. 2, Amsterdam, The Netherlands: Elsevier, 1998.

[29] S. Benedetto, D. Divsalar, G. Montrosi, and F. Pollara, "Algorithm for continuous decoding of turbo codes," *Electronics Letters*, vol. 32, pp. 314–315, February 1996.

[30] E. K. Hall and S. G. Wilson, "Stream-oriented turbo codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 1813–1831, July 2001.

[31] A. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 260–264, February 1998.
[32] H.-A. Loeliger, "A posteriori probabilities and performance evaluation of trellis codes"," in *Proc. IEEE Int. Symp. Inform. Theory*, (Trondheim), p. 335, 1994.
[33] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 429–445, March 1996.
[34] W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data distributed by time-varying intersymbol interference," in *Proc. IEEE GLOBECOM'90*, (San Diego, CA), pp. 1679–1684, Dec. 1990.
[35] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. Telecommun.*, vol. 8, pp. 119–125, Mar./Apr. 1997. also published at IEEE ICC'95.
[36] M. P. C. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the equivalence between SOVA and max-log-MAP decodings," *IEEE Communications Letters*, vol. 2, pp. 137–139, May 1998.
[37] V. Franz and J. B. Anderson, "Concatenated decoding with a reduced-search BCJR algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 16, pp. 186–195, February 1998.
[38] P. Li and W. K. Leung, "Decoding low density parity check codes with finite quantization bits," *IEEE Communications Letters*, pp. 62–64, 2000.
[39] A. Kavčić, "On the capacity of Markov sources over noisy channels," in *Proceedings IEEE Global Communications Conference 2001*, vol. 5, (San Antonio, Texas), pp. 2997–3001, November 2001.

PLACE PHOTO HERE

Xiao Ma received the B.Sc. degree in Applied Mathematics from Harbin Engineering University, China in 1991. He received the M.S. degree in Communication and Electronic Systems and Ph.D. degree in Communication and Information Systems from Xidian University, China in 1997 and 2000, respectively.

From 1991 to 1994, he was an Assistant Professor at Henan Jiaozuo University, China. He held short-term research positions at City University of Hong Kong and University of Delaware in 1999 and 2000, respectively. From 2000 to 2002, he was a Postdoctoral Research Fellow at Harvard University. He is currently a Research Fellow at City University of Hong Kong. His research interests include applied mathematics, information theory and channel coding.

PLACE PHOTO HERE

Aleksandar Kavčić (S'93–M'98) was born in Belgrade, Yugoslavia in 1968. He received the Dipl.-Ing. degree in Electrical Engineering from Ruhr-University, Bochum, Germany in 1993, and the Ph.D. degree in Electrical and Computer Engineering from Carnegie Mellon University, Pittsburgh, Pennsylvania in 1998.

Since 1998, he has been with the Division of Engineering and Applied Sciences at Harvard University where he is currently the John L. Loeb Associate Professor of the Natural Sciences. From 1998 untli 2002, he was an Assistant Professor of Electrical Engineering at Harvard University. He held short-term research positions at Seagate Technology in 1995, Read-Rite Corporation in 1996, and Quantum Corporation from 1997 to 1998, and has served as a technical consultant for Quantum Corporation in 1999 and 2000. His research spans topics in Communications, Signal Processing, Information Theory and Magnetic Recording, with the most recent interests in magnetic recording channel modeling, multichannel signal processing, detector design, timing recovery, capacity computations for channels with memory and iterative decoding algorithms.

Dr. Kavčić received the IBM Partnership Award in 1999 and the NSF CAREER Award in 2000. He is presently serving on the editorial board of the IEEE TRANSACTIONS ON INFORMATION THEORY as Associate Editor for Detection and Estimation.