Hindawi Publishing Corporation
EURASIP Journal on Wireless Communications and Networking
Volume 2008, Article ID 729180, 8 pages
doi:10.1155/2008/729180

Research Article

Decoding LDPC Convolutional Codes on Markov Channels

Manohar Kashyap and Chris Winstead

Department of Electrical and Computer Engineering, College of Engineering, Utah State University, Logan, UT 84322-4120, USA

Correspondence should be addressed to Manohar Kashyap, manohar.kashyap@gmail.com

Received 31 August 2007; Accepted 26 February 2008

Recommended by Yonghui Li

This paper describes a pipelined iterative technique for joint decoding and channel state estimation of LDPC convolutional codes over Markov channels. Example designs are presented for the Gilbert-Elliott discrete channel model. We also compare the performance and complexity of our algorithm against joint decoding and state estimation of conventional LDPC block codes. Complexity analysis reveals that our pipelined algorithm reduces the number of operations per time step compared to LDPC block codes, at the expense of increased memory and latency. This tradeoff is favorable for low-power applications.

1. INTRODUCTION

LDPC convolutional codes (LDPC-CCs) are the convolutional counterparts of LDPC block codes (LDPC-BCs) and were first presented in 1999 by Feltström and Zigangirov [1]. Algorithms have been studied for decoding LDPC Convolutional codes over memoryless channels, such as the additive white Gaussian noise (AWGN) channel. LDPC-CC decoding over channels with memory has not been addressed to date. Wireless channels typically have memory, and are often approximated using discrete Markov channel models, of which the simplest is the well-known Gilbert-Elliott model.

LDPC-CC codes are attractive for three main reasons. First, they support arbitrary frame lengths, which is useful in packet-switched networks. Second, they achieve performance comparable to conventional LDPC codes. Finally, LDPC-CC decoders can be implemented using a pipelined architecture that significantly reduces the number of active operations required per iteration.

Markov models are widely used to represent time-varying communication channels [2]. If the channel is subject to significant variation within a transmitted frame, then performance can be significantly improved if decoding and channel state estimation are performed jointly using iterative algorithms.

This paper introduces a new algorithm for LDPC-CC decoding over Markov channels. The new algorithm applies the general principles of joint inference [3, 4] to the specialized problem of efficient LDPC-CC decoding on channels with memory. We demonstrate that joint decoding and channel state estimation can be performed by adding a few steps to the pipeline decoding algorithm. This means that the cost of adding joint channel state estimation to an existing LDPC-CC decoder is extremely small. We show that when complexity is measured as arithmetic operations per iteration, joint state estimation and decoding is much less complex for LDPC-CCs than for traditional LDPC block codes. The tradeoff is that LDPC-CC decoders require considerably more memory, but in most cases this is a favorable trade in terms of power, complexity, and circuit area.

As a proof-of-concept demonstration, we apply our algorithm to joint decoding and state estimation on the Gilbert-Elliott channel. The Gilbert-Elliott channel [5] represents a burst-error channel which toggles between “good” and “bad” binary symmetric channel states. Interleaving methods were previously used to remove the memory from a Gilbert-Elliott channel, but this technique resulted in substantially reducing the channel’s effective capacity [6]. More recently, it was shown that joint state estimation and decoding of turbo codes on Gilbert-Elliott channels allows transmission at rates closer to the Gilbert-Elliott channel’s native capacity [7]. Our algorithm provides a power-efficient adaptation of these approaches for the LDPC-CC case.

The paper is organized as follows. Section 2 reviews LDPC-CC codes and their pipelined decoding algorithms for memoryless channels. Section 3 reviews methods for joint decoding and channel state estimation for Markov channels.

Section 3.4 presents the new pipelined estimation-decoding algorithm for LDPC-CCs. Section 4 presents performance and complexity analysis, for example, LDPC and LDPC-CC codes over the Gilbert-Elliott channel. Section 5 offers conclusions.

The paper uses the following notation. Random variables and their quantities are indicated by lower-case Latin letters. If X and Y are random variables, then $\Pr\{x \mid y\}$ should be taken to mean the probability that $X = x$, given that $Y = y$. Sets are indicated by upper-case Latin letters. Sequences are indicated by lower-case bold letters, and matrices by upper-case bold letters. Lower-case Greek letters are used to indicate probability messages in the decoding algorithm.

2. LDPC CONVOLUTIONAL CODES

2.1. Code structure

LDPC-CCs are a family of time-varying convolutional codes with characteristics similar to conventional LDPC block codes (LDPC-BCs). LDPC-CCs are defined by a periodic low-density parity-check matrix:

$$\mathbf{H}^T = \begin{pmatrix} \mathbf{H}_0^T(0) & \cdots & \mathbf{H}_M^T(M) & & \\ & \ddots & & \ddots & \\ & & \mathbf{H}_0^T(t) & \cdots & \mathbf{H}_M^T(t+M) \\ & & \ddots & & \ddots \end{pmatrix}, \quad (1)$$

where M is the *memory* and T is the *period* of the LDPC-CC, and $t \in \mathbb{Z}$ is the time index. The submatrices $\mathbf{H}_i^T(t+i)$, $i = 0, \ldots, M$ are $c \times (c-b)$ binary matrices, where b is the number of information bits that enter the encoder, and c is the number of coded bits that exit the encoder at a given time index. The rate of the code is $R = b/c$. The memory is equal to the largest i such that $\mathbf{H}_i^T(t+i)$ is a nonzero matrix, and $T = M(c-b)$. An example parity-check matrix for an $M = 7$ rate-1/2 LDPC-CC is shown in Figure 1.

Similar to the Tanner graph representation of LDPC-BCs, LDPC-CCs can also be represented graphically [8]. Figure 2 shows a Tanner graph representation corresponding to the parity-check matrix in Figure 1. The Tanner graph exhibits a pattern that repeats itself (for time invariant LDPC-CCs) every M time indices or, equivalently, every Mc symbol nodes. Edges are present between symbols and check nodes when a "1" is present in the corresponding location of the parity-check matrix.

2.2. Decoding LDPC-CCs on memoryless channels

The seminal paper on LDPC-CCs [1] presented an iterative decoding strategy for memoryless channels. In this subsection, we briefly review the LDPC sliding window decoding algorithm using factor graphs as in [8].

The LDPC-CC decoder is a chain of sliding window *processors* performing sum product algorithm (SPA) [9] calculations on symbols within the window. While an LDPC-BC decoder performs parallel operations on symbol and

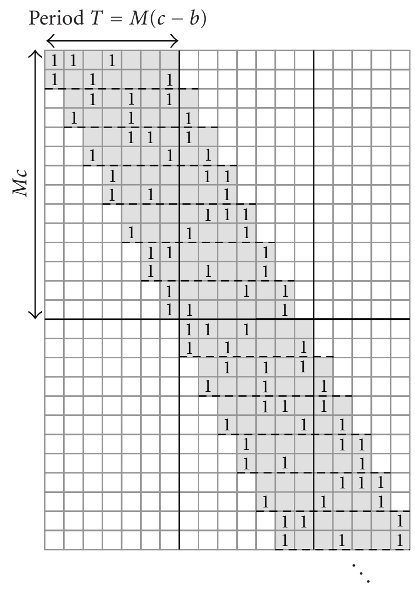

Figure 1: An example parity-check matrix, H^T, for a rate-1/2 LDPC-CC. The row indices correspond to symbols (i.e., channel bits), and the column indices correspond to parity check equations.

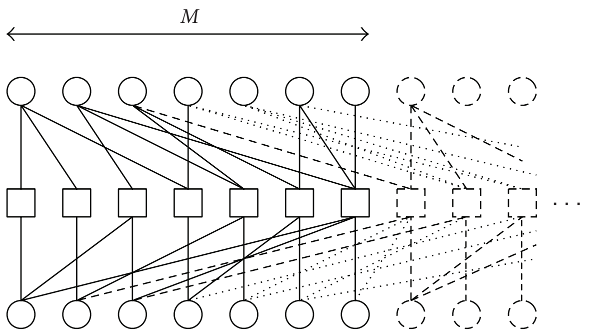

Figure 2: Tanner graph corresponding to Figure 1. The circles indicate symbol nodes, and the squares indicate parity-check nodes.

parity-check nodes, iterations in the LDPC-CC decoder happen sequentially. As a symbol node shifts through the sliding window, SPA operations are performed on some of its edges. SPA operations are split into two phases per time index. During the *vertical* phase, SPA operations are performed on b parity-check nodes. During the *horizontal* phase, SPA operations are performed on c symbol nodes. The LDPC-CC H matrix is designed to guarantee that all necessary SPA operations are performed on a node after $M+1$ time shifts, just before the symbol node exits the window [1]. For a rate-$b/c(J, K)$ LDPC-CC decoder, the vertical and horizontal phases require Kb and Jc SPA operations, respectively.

To carry out multiple iterations, abutting sliding window processors are used. As a node exits one processor, it enters the next processor for an additional iteration. Therefore, for I iterations, the decoder has a latency of $I(M+1)$ time units. Hard decisions are made based on the *a posteriori probability* value of a symbol as it exits the last processor in the chain.

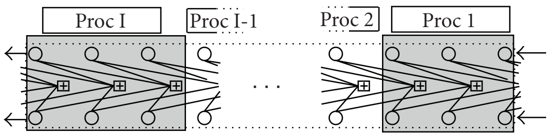

Figure 3: The LDPC-CC decoder sliding window represented as a series of processors.

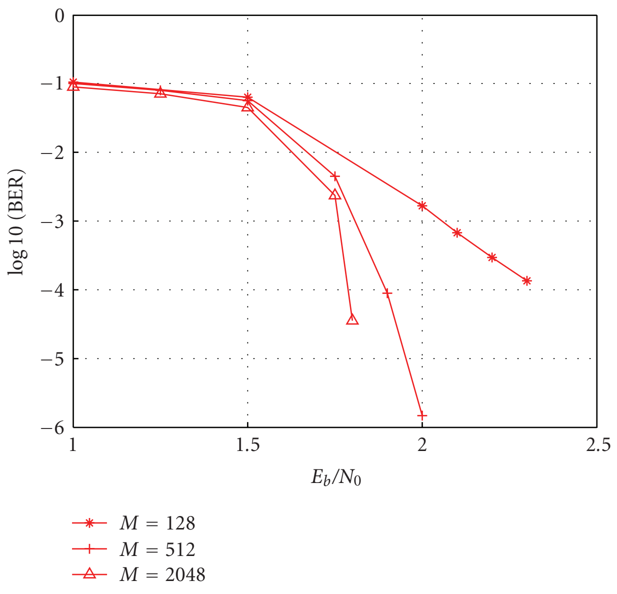

Figure 4: LDPC-CC decoder performance over the AWGN channel, based on codes and algorithms from [10].

Figure 3 shows the decoder window in operation over a rate 1/2 code, sliding from right to left.

The sliding window abstraction is very useful from a hardware design perspective. A designer only needs to implement one of these processors as a hardware block. The complete decoder is then constructed by tiling I copies of the processor block. As a point of reference, a comparison of the BER plots for rate 1/2 LDPC-CC and LDPC-BC codes on the AWGN channel [10] reveals that an $M = 128$ LDPC-CC has roughly the same performance as an $N = 1024$ LDPC-BC. In Section 4, we demonstrate that this rough equivalence also holds for joint decoding and state estimation on Gilbert-Elliott channels. Figure 4 summarizes AWGN decoding results obtained in [10].

3. JOINT ESTIMATION DECODING

In this section, we present an iterative decoding algorithm for LDPC-CCs over channels with memory. Memory in channel states is modeled as a *Markov chain*, hence the name *Markov channel*. We review the Gilbert-Elliott channel and LDPC-BC decoding over Markov channels before presenting the LDPC-CC decoding algorithm. All of these algorithms employ the SPA on factor graphs of the decoder. The general procedure followed in deriving these algorithms is as follows, summarized from [11]:

(1) derive the factor graph (or, equivalently, the Tanner graph) of the code constraints. The factor graph represents the probability mass function (PMF) of the code itself, as elaborated in [9]. In the typical case where all codewords are equiprobable, the code's PMF is determined by the *characteristic function*:

$$\Upsilon(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \text{ is a code word} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

(2) Derive the conditional joint PMF of the received vector $\mathbf{y}$ and the channel states $\mathbf{s}$, that is, $\Pr(\mathbf{y},\mathbf{s} \mid \mathbf{x})$. For discrete channels, the PMF takes the form of a Markov state-transition model, which has a well-known factor graph structure.

(3) The joint PMF describing the code and channel, $\Pr(\mathbf{y},\mathbf{s},\mathbf{x})$, is the product of the above derived PMFs, $\Upsilon(\mathbf{x})$ and $\Pr(\mathbf{y},\mathbf{s} \mid \mathbf{x})$. The corresponding decoder factor graph is constructed by joining together the channel graph and the code graph at their intersecting symbol nodes.

3.1. *Markov channels*

A Markov channel is characterized by a set of *states*, S, which models the channel's memory. At any time index i, the channel is in some state $s_i \in S$. At each time index, the channel's state can undergo a random transition governed by the state transition probability matrix $\mathbf{P}$. Let ρ_i be the PMF vector for the channel's state at time i. Then $\rho_{i+1} = \mathbf{P} \times \rho_i$.

For a general Markov channel, the PMF of the channel state process can be written as

$$\Pr(\mathbf{s}) = \Pr(s_1)\prod_{i=1}^{n}\Pr(s_{i+1} \mid s_i), \tag{3}$$

where $\Pr(s_{i+1} \mid s_i)$ are obtained from $\mathbf{P}$, the state transition probability matrix of a general Markov channel. The channel state affects the channel output probability through a conditional PDF $f_{y_i}(y_i \mid x_i, s_i)$ for each $s_i \in S$. Let γ_i be the *channel information* which is conditional upon the channel state, defined as

$$\gamma_i(y_i, x_i, s_i, s_{i+1}) = \Pr(s_{i+1} \mid s_i) f_{y_i}(y_i \mid x_i, s_i). \tag{4}$$

In the sequel, the arguments to γ_i are be omitted, but γ_i should always be understood to have the functional dependence as expressed by (4).

Then the conditional joint PMF of the received symbols and channel states is

$$\Pr(\mathbf{y},\mathbf{s} \mid \mathbf{x}) = \Pr(s_1)\prod_{i=1}^{n}\gamma_i, \tag{5}$$

where $\mathbf{y}$ is the received symbol sequence and $\mathbf{x}$ is the transmitted sequence.

3.2. Gilbert-Elliott channel

The Gilbert-Elliott (GE) channel [5] is a binary input-output Markov channel. It is a binary symmetric channel (BSC) whose inversion probability is modulated by a Markov chain. It can be described as

$$\mathbf{y} = \mathbf{x} \oplus \mathbf{z}, \quad \mathbf{x}, \mathbf{y}, \mathbf{z} \in \{0,1\}^n, \tag{6}$$

where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ are the channel input, the channel output, and the error sequence, respectively. The GE channel has two states, good and bad, as indicated in the channel model [5]. Each state contains a BSC with its own inversion probability. In the good state, the inversion probability is lower than the bad state. The error sequence $\mathbf{z}$ is a random sequence. At any given time, the error probability is

$$\Pr(z_i = 1 \mid s_i) = \begin{cases} \eta_B, & s_i = B; \\ \eta_G, & s_i = G; \end{cases} \tag{7}$$

where η_B and η_G are the inversion probabilities in the bad and good channel states, respectively.

If the good and bad states are assigned to numerical values 1 and 0, respectively, then the state transition matrix $\mathbf{P}$ is

$$\mathbf{P} = \begin{bmatrix} 1-g & g \\ b & 1-b \end{bmatrix}, \tag{8}$$

where b and g are the $1 \to 0$ and $0 \to 1$ transition probabilities, respectively. The matrix elements are $p_{jk} = \Pr(s_{i+1} = j \mid s_i = k)$. The last term needed to compute γ_i is the channel PDF:

$$f_{y_i}(y_i \mid x_i, s_i) = \begin{cases} (1-\eta_G), & x_i = y_i,\ s_i = G \\ \eta_G, & x_i \neq y_i,\ s_i = G \\ (1-\eta_B), & x_i = y_i, s_i = B \\ \eta_B, & x_i \neq y_i, s_i = B. \end{cases} \tag{9}$$

3.3. LDPC BC decoding on Markov channels

Iteratively decoded codes allow for channel estimation during the decoding process and these estimates can further assist in decoding and vice versa. This is known as *joint estimation decoding*. In this subsection, we review an estimation-decoding algorithm first presented in [6] and analyzed for LDPC-BCs in [3, 4, 11, 12]. In Section 3.4, these concepts are adapted for use with LDPC-CCs.

To characterize the PMF of the channel, it is convenient to decompose the characteristic function, $\Upsilon(\mathbf{x})$, into component functions $h_j(\mathbf{x})$ representing the individual rows of the parity-check matrix:

$$\Upsilon(\mathbf{x}) = \prod_{j=1}^{M} h_j(\mathbf{x}). \tag{10}$$

The joint PMF of the transmitted codeword received symbol sequence and channel states is then given by

$$\Pr(\mathbf{y}, \mathbf{s}, \mathbf{x}) = \xi \Pr(s_1) \prod_{i=1}^{N} \gamma_i \prod_{j=1}^{M} h_j(\mathbf{x}), \tag{11}$$

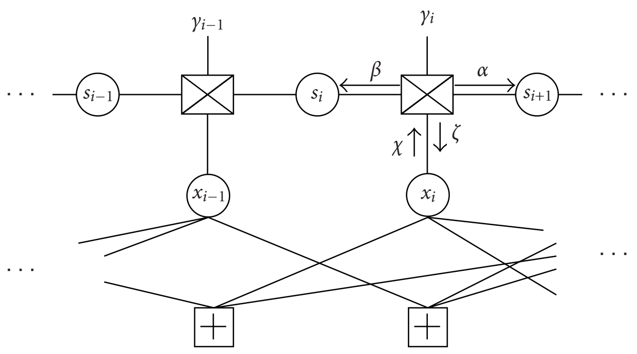

Figure 5: Joint factor graph for decoding and channel state estimation of an LDPC-BC on a Markov channel [11].

where ξ is the normalization constant for $\Upsilon(\mathbf{x})$, and N and M are the dimensions of the parity-check matrix.

The factor graph corresponding to (11) is shown in Figure 5. The channel state and transition nodes comprising the Markov chain form the *Markov subgraph*. The symbol and check nodes of the classic LDPC graph form the *LDPC subgraph*. Joint state estimation and decoding are then performed by applying the SPA using a flooding schedule. In the LDPC subgraph, the ζ messages are substituted in place of the channel information. The χ messages are the a posteriori probabilities computed using the usual LDPC decoding computations.

In the Markov subgraph, BCJR operations are performed with the χ messages applied as a priori information. At the start of each iteration, every node in the factor graph receives messages from adjacent nodes in the graph. Each message is a local conditional PMF. To complete the iteration, the SPA algorithm is applied to update the outgoing messages at each node in the graph. The α, β, and ζ messages have the usual definition, based on the BCJR algorithm, and are updated according to the standard rules:

$$\begin{aligned} \alpha_i &= \sum_{s_{i-1} \in S} \alpha_{i-1} \sum_{x_i \in \{0,1\}} \chi_{i-1} \gamma_{i-1}, \\ \beta_i &= \sum_{s_{i+1} \in S} \beta_{i+1} \sum_{x_i \in \{0,1\}} \chi_{i+1} \gamma_{i+1}, \\ \zeta_i &= \sum_{s_i, s_{i+1}} \alpha_i \beta_{i+1} \gamma_i, \end{aligned} \tag{12}$$

where α_i and β_i are functions of s_i, χ_i and ζ_i are functions of x_i, and the function arguments are again omitted.

3.4. LDPC convolution decoding on Markov channels

In this section, we present an algorithm for sequential decoding of LDPC convolutional codes on general Markov channels, the Gilbert-Elliott channel LDPC convolutional decoder is presented as an example. We will show that a *flooding schedule* of message-passing between the channel model and decoder fits conveniently into the pipelined LDPC-CC decoding algorithm.

To derive the decoder graph, we combine the Markov channel factor graph with the graph of an LDPC convolutional code by joining the two graphs at the shared symbol nodes, x_i. The derivation of the LDPC convolutional code characteristic function, and hence its factor graph, is followed by the derivation of the decoder PMF, and hence the decoder graph, in the next few paragraphs.

The sequence $\boldsymbol{v} = (\ldots, \boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \ldots)$, $\boldsymbol{v}_\mathbf{t} \in \mathbf{F}_2^c$ forms a convolutional code frame (equivalent to a block codeword) if and only if the constraint imposed by the syndrome former of the convolutional code is fulfilled, that is,

$$\boldsymbol{v}_t \mathbf{H}_0^T(t) + \boldsymbol{v}_{t-1}\mathbf{H}_1^T(t) + \cdots + \boldsymbol{v}_{t-M}\mathbf{H}_M^T(t) = 0. \qquad (13)$$

The characteristic function of the convolutional code is

$$\Upsilon(\boldsymbol{v}) = \begin{cases} 1, & \text{if (13) is satisfied for all } t \in \mathbb{Z}, \\ 0, & \text{otherwise.} \end{cases} \qquad (14)$$

The convolutional characteristic function can be decomposed as a product of component functions over time. Let $\mathbf{x}_j$ be defined as the sequence $(\boldsymbol{v}_j, \boldsymbol{v}_{j-1}, \ldots \boldsymbol{v}_{j-M})$, corresponding to the symbols in the encoder's memory. Then at the jth time instant, the characteristic component function is

$$h_j(\mathbf{x}_j) = \begin{cases} 1 & \text{if (13) is satisfied at time } t = j, \\ 0 & \text{otherwise.} \end{cases} \qquad (15)$$

The convolutional characteristic function is then the product of the component functions over time:

$$\Upsilon(\boldsymbol{v}) = \prod_{j=0}^{t_{\text{end}}} h_j(\mathbf{x_j}), \qquad (16)$$

where t_{end} is the time at which the convolutional sequence terminates.

To construct the joint PMF of the convolutional code and the Markov channel model, we note that there are c channel symbols per time index in our notation, and therefore also c Markov channel states per time index. Let n be the total number of channel symbols transmitted, that is, $n = t_{\text{end}} \times c$. Then,

$$\Pr(\mathbf{y}, \mathbf{s}, \mathbf{x}) = \xi \Pr(s_1) \prod_{i=1}^{n} \gamma_i \prod_{j=1}^{t_{\text{end}}} h_j(\mathbf{x}_j). \qquad (17)$$

The factor graph for the LDPC convolutional decoder is shown in Figure 6. Notice that the Markov channel graph of Figure 5 has been wrapped around the convolutional graph of Figure 3 resulting in a graph suitable for sequential operations. As the decoding window slides across this graph, it completes SPA operations on c channel state variable nodes along with c symbol nodes and $(c - b)$ check nodes at each time index.

Joint decoding and estimation is performed using a flooding schedule as follows. Whenever a symbol node is updated in the decoder, the adjacent Markov state variables are also updated. At each time index in the LDPC-CC pipeline, the χ messages are updated for c symbol nodes using the usual decoding operations. To perform the Markov state update, the α, β, and ζ messages are subsequently computed for each updated symbol node.

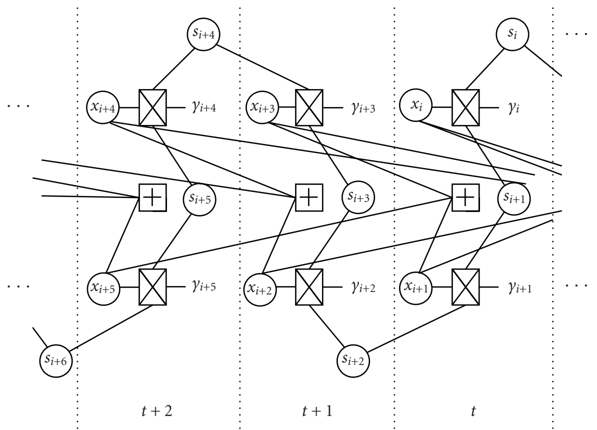

FIGURE 6: Joint factor graph for an LDPC-CC decoder over a Markov channel. The Markov model wraps around the LDPC-CC factor graph.

Whenever processing is completed for a node x_i, the resulting χ_i message is used to update α_i, β_i, and ζ_i. The entire column of updated messages is then passed to the next processor, which performs the subsequent iteration. Messages on the Markov subgraph are updated directly alongside the x_i symbol nodes within the pipeline.

Figure 7 shows the memory-based architecture for a typical LDPC-CC processor, augmented to support joint channel state estimation [10, 13, 14]. Each column in the memory grid represents a symbol node. Each row represents a message passed along an edge in the code's factor graph. Each symbol node is connected to the channel and to (up to) J parity-check nodes. Due to the structure of the LDPC-CC factor graph, a symbol's influence vanishes after $M \times c$ time steps, so this is the number of columns that needs to be stored in the memory.

The processor requires one row to store channel information for each symbol and J rows to store messages between variable and check nodes in the LDPC-CC subgraph. For a Markov channel with S states, $2S$+2 extra rows must be added to support channel state estimation. In the particular case of the GE channel, where $S = 2$, six extra rows are needed. SPA operations are completed for c symbol nodes per time index per processor, so each processor needs to update $4c$ additional messages per time index per processor to perform joint state estimation. As these messages pass through the pipeline of processors, updates propagate across the Markov subgraph.

3.5. Alternative message passing schedules

The flooding schedule produces a very efficient architecture for joint estimation and decoding in LDPC-CCs. Other message passing schedules are not expected to coexist well with the pipelined structure of LDPC-CC decoders. Turbo

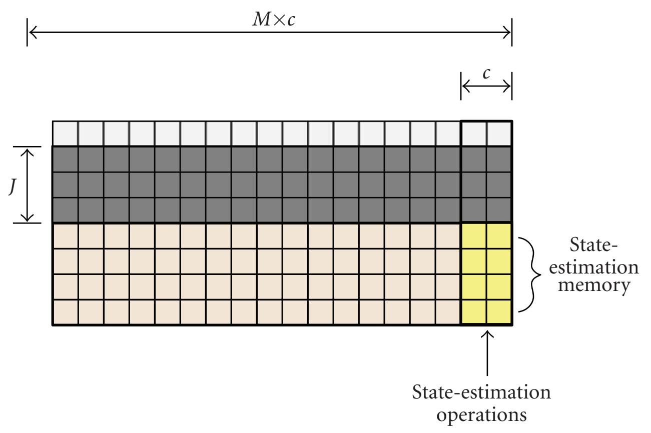

FIGURE 7: The memory-based LDPC-CC decoding architecture for a rate-1/2 code, highlighting the extramemory resources and additional operations needed to implement joint channel state estimation.

estimation/decoding, for example, is known to provide improved performance for LDPC block codes in some cases, but cannot obviously be applied to the LDPC-CC case.

In the turbo schedule, Markov state estimation is carried out using the BCJR algorithm on a sliding window. All α, β, and ζ messages are computed while the χ messages are held constant. Meanwhile, the decoder computes new χ messages by performing one iteration with the ζ messages held constant. This schedule requires that *blocks* of symbol messages be exchanged between the estimator and the decoder. Because the pipelined decoder computes only c symbol messages per time index, it is not possible to accumulate a large frame of messages without interrupting the pipeline.

4. PERFORMANCE RESULTS

Figure 8 shows the BER plot for rate 1/2, (3, 6) LDPC convolutional codes with memories 128, 256, and 2048 with channel parameters $(b, g, \eta_G) = (0.01, 0.01, 0.01)$, where η_B is swept from 0.04 to 0.18. As expected, decoding gain increases with memory. As with joint estimation and decoding on LDPC-BCs, LDPC-CCs are observed to have low error floors. Figure 9 plots the BERs for an $M = 128$ code, with iterations ranging from 10 to 60. Statistically, significant improvements are not observed for pipelines longer than 50 processors.

4.1. *Comparison of LDPC-BC and LDPC-CC decoders*

Figure 10 shows the comparative performance of an LDPC-BC and an LDPC-CC performing joint decoding and state estimation over the GE channel. The performance of an $N = 1024$ LDPC-BC is roughly comparable to that of an $M = 128$ LDPC-CC. The LDPC-CC decoder uses 50 processors, and the LDPC-BC decoder uses 50 iterations.

The complexity of these decoders is estimated by counting the number of edges that requires SPA updates at each time in the decoder. The comparative processor complexity, memory requirements, and latency are reported in Table 1.

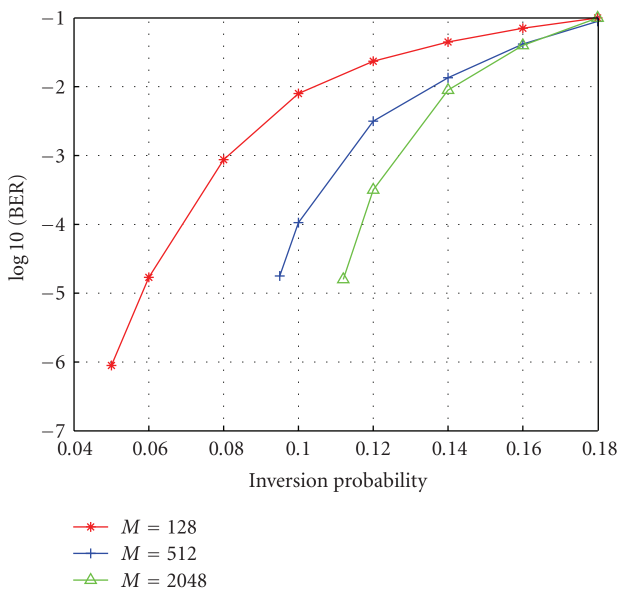

FIGURE 8: LDPC-CC decoding over the Gilbert-Elliott channel.

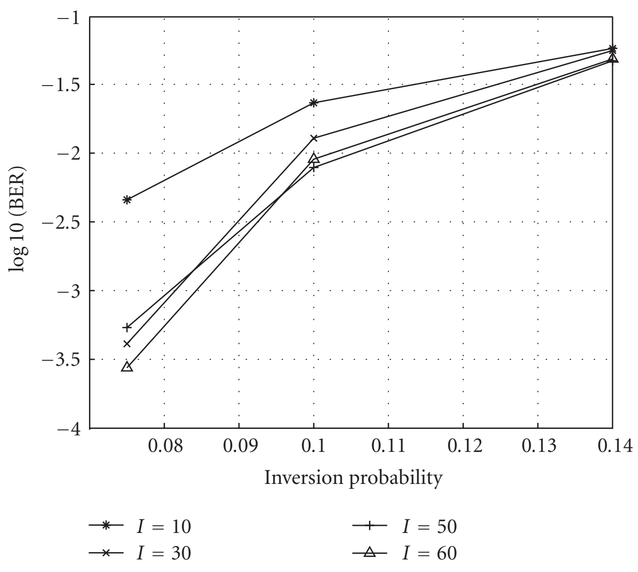

FIGURE 9: Number of iterations versus performance for an $M = 128$ LDPC-CC decoder.

TABLE 1: Complexity comparisons for (3,6) rate-1/2 LDPC block and convolutional codes corresponding to Figure 10.

	LDPC-BC, $N = 1024$	LDPC-CC, $M = 128$
Processor complexity	12,288 updates	1,000 updates
Memory requirements	13,312 messages	103,200 messages
Latency	50 time steps	6,450 time steps

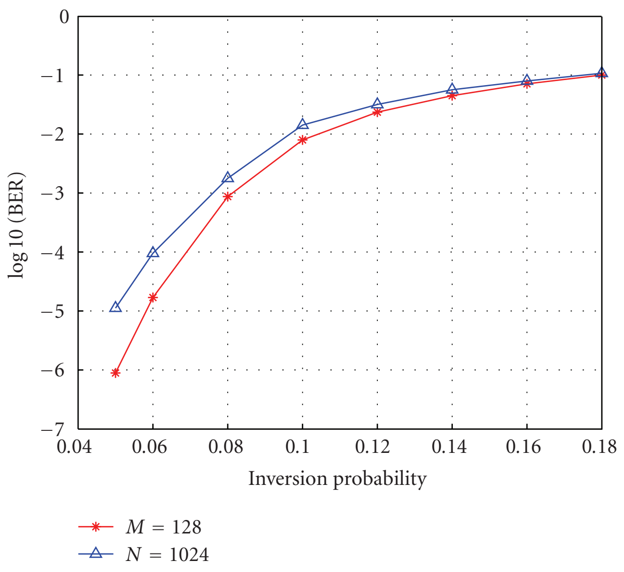

Figure 10: $M = 128$ LDPC-CC and $N = 1024$ LDPC-BC BER performance over the same GE channel.

(1) Processor complexity

During a time index, each LDPC-CC processor completes an iteration on c symbol nodes, each of which requires J SPA updates during the horizontal phase and K SPA updates during the vertical phase. Then there are 3 edges to be updated per symbol node in the Markov subgraph, and two of those edges have S number of messages on each. Like for the GE channel, the forward and backward messages have values for the good and bad states of the channel. Hence $(2S + 1)c$ SPA updates are needed (the APP edge does not need to be computed until the very end). Therefore, we have $I * [c * (J + 2S + 1) + K]$ SPA updates during any time index in the LDPC-CC decoder.

In the LDPC-BC case, there is one time index per iteration, in which all messages are updated throughout the code's graph. There are J edges per symbol node in the LDPC subgraph and two SPA updates per edge. The N symbol nodes result in a total of $2N * (J + 2S + 1)$ SPA updates per time index.

(2) Hardware memory requirements

In the LDPC-CC case, for each symbol node, we need to store $(2S + 1)$ messages for the Markov subgraph, 1 channel message, and J messages in the LDPC subgraph. There are $(M + 1)c$ symbol nodes per processor, and I processors, for a total of $I * [(M + 1)(J + 2S + 2)c]$ messages.

The LDPC-BC decoder needs memory for each SPA update, and memory to store the channel information, resulting in $N * [2 * (J + 2S + 1) + 1]$ messages to be stored.

(3) Decoder delay

The LDPC-CC latency is $I * (M + 1)$ time steps, the time it takes for symbols to traverse the entire pipeline. The latency of the LDPC-BC decoder is the time taken to decode the first frame. If the decoder uses I iterations, then the latency is simply I.

For similar performing block and convolutional codes, LDPC-CCs need a simpler processing unit, that is, the area of an LDPC-CC decoder chip involved in active computations is much less than for the LDPC-BC case, hence bringing down the power usage, but needs more memory. On general purpose architectures with multiple memory banks, the LDPC-CC decoder realizations can be very efficient. Apart from the initial latency, the pipelined LDPC-CC decoder architecture is well suited for high throughput, low-power hardware implementations.

5. CONCLUSIONS

This article described a serial, pipelined algorithm for joint decoding and channel-state estimation of LDPC convolutional decoders over Markov channels. While the general principles of joint channel state estimation and decoding are widely known, they were previously applied only to LDPC block codes. Our work is the first investigation of joint estimation and decoding for LDPC convolutional codes. We found that the conventional pipelined LDPC-CC decoder requires only a few extra operations to implement joint state estimation, which is a much smaller overhead than what is required for joint state estimation in known LDPC block decoding algorithms. Our LDPC-CC algorithm requires considerably fewer active operations than LDPC block decoders, and hence is better suited for low-power implementation of high-performance error control over Markov channels.

ACKNOWLEDGMENT

The authors would like to extend their gratitude to Dr. Stephen Bates and Ramkrishna Swamy for providing access to their LDPC-CC code designs and simulation source code, which were very helpful to our research.

REFERENCES

[1] A. J. Feltström and K. S. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 2181–2191, 1999.

[2] H. S. Wang and N. Moayeri, "Finite-state Markov channel-a useful model for radio communication channels," *IEEE Transactions on Vehicular Technology*, vol. 44, no. 1, pp. 163–171, 1995.

[3] J. Garcia-Frias, "Decoding of low-density parity-check codes over finite-state binary Markov channels," *IEEE Transactions on Communications*, vol. 52, no. 11, pp. 1840–1843, 2004.

[4] A. W. Eckford, "Low-density parity-check codes for Gilbert-Elliott and Markov-modulated channels," Ph.D. dissertation, University of Toronto, Toronto, Ontario, Canada, 2004, http://www.cse.yorku.ca/ aeckford/pubs/phd-thesis.pdf.

[5] E. Elliott, "Estimates of error rates for codes on burst-noise channels," *Bell Systems Technical Journal*, vol. 42, no. 9, pp. 1977–1997, 1968.

[6] J. Garcia-Frias and J. Villasenor, "Turbo decoding of Gilbert-Elliott channels," *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 357–363, 2002.

[7] M. Mushkin and I. Bar-David, "Capacity and coding for the Gilbert-Elliot channels," *IEEE Transactions on Information Theory*, vol. 35, no. 6, pp. 1277–1290, 1989.

[8] M. S. Arvind Sridharan, "Design and analysis of LDPC convolutional codes," Ph.D. dissertation, University of Notre Dame, Notre Dame, Ind, USA, 2005, http://etd.nd.edu/ETD-db/theses/available/etd-02202005-181524/unrestricted/SridharanA0202005.pdf.

[9] F. R. Kschischang, B. J. Frey, and H.-H. Loeliger, "Factor graphs and the sumproduct algorithm," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[10] S. Bates and Z. Cheng, "Stephen Bate's online LDPC convolutional codes tutorial," http://www.ece.ualberta.ca/ sbates/LdpcWeb/perf.html.

[11] A. W. Eckford, F. R. Kschischang, and S. Pasupathy, "On designing good LDPC codes for Markov channels," *IEEE Transactions on Information Theory*, vol. 53, no. 1, pp. 5–21, 2007.

[12] A. W. Eckford, F. R. Kschischang, and S. Pasupathy, "Analysis of low-density parity-check codes for the Gilbert-Elliott channel," *IEEE Transactions on Information Theory*, vol. 51, no. 11, pp. 3872–3889, 2005.

[13] S. Bates and G. Block, "A memory-based architecture for FPGA implementations of low-density parity-check convolutional decoders," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 1, pp. 336–339, Kobe, Japan, May 2005.

[14] R. Swamy, S. Bates, and T. L. Brandon, "Architectures for ASIC implementations of low-density parity-check convolutional encoders and decoders," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 5, pp. 4513–4516, Kobe, Japan, May 2005.