A Comparison of ARA- and Protograph-Based LDPC Block and Convolutional Codes

Daniel J. Costello, Jr., Ali Emre Pusane
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556,
U.S.A.
{Costello.2, Pusane.1}@nd.edu

Christopher R. Jones, Dariush Divsalar
Jet Propulsion Laboratory
Pasadena, CA 91109-8099
U.S.A.
{crjones, dariush.divsalar}@jpl.nasa.gov

***Abstract*— ARA- and protograph-based LDPC codes are capable of achieving error performance similar to randomly constructed codes while enjoying several implementation advantages as a result of their structure. LDPC convolutional codes can be derived from these codes through an unwrapping process. In this paper, we review the unwrapping process as well as the pipeline decoder that allows continuous decoding of LDPC convolutional codes. Computer simulations are then used to demonstrate that the unwrapped convolutional codes achieve a "convolutional gain" in error performance. We conjecture that this is due to the concatenation of many constraint lengths worth of received symbols in the pipeline decoding process. The consequences of this improved performance are examined in terms of factors related to decoder implementation: processor size, memory requirements, and decoding delay (latency). Finally, given identical protograph kernels, we compare derived block and convolutional codes based on the above measures.**

I. Introduction

The convolutional counterparts of low-density parity-check (LDPC) block codes, LDPC convolutional codes, were first proposed in [1]. Analogous to LDPC block codes, LDPC convolutional codes are defined by sparse parity-check matrices, which allow them to be decoded using iterative message-passing algorithms. The so-called pipeline decoder, that is typically used to decode these codes, employs several small identical processors that perform the message-passing decoding iterations in parallel. In [2], the first two authors presented a general comparison of LDPC block and convolutional codes, investigating several practical encoding and decoding aspects of these codes. In this paper, we extend that work by looking specifically at LDPC convolutional codes derived from ARA- and Protograph-Based LDPC block codes.

II. Preliminaries

We start with a brief definition of a rate $R = b/c$ binary LDPC convolutional code $\mathcal{C}$. (A more detailed description can be found in [1].) Let

$$\mathbf{u}_{[0,t-1]} = [\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{t-1}], \tag{1}$$

where $\mathbf{u}_i = (u_i^{(1)}, u_i^{(2)}, ..., u_i^{(b)}), 0 \le i < t, t \in \mathcal{Z}^+$, and $u_i^{(\cdot)} \in GF(2)$, be an information sequence. The encoder maps this sequence into the code sequence

$$\mathbf{v}_{[0,t-1]} = [\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_{t-1}], \tag{2}$$

where $\mathbf{v}_i = (v_i^{(1)}, v_i^{(2)}, ..., v_i^{(c)}), 0 \le i < t, t \in \mathcal{Z}^+$, and $v_i^{(\cdot)} \in GF(2)$.

A code sequence $\mathbf{v}_{[0,\infty]}$ satisfies the equation

$$\mathbf{v}_{[0,\infty]}\mathbf{H}^{\mathrm{T}}_{[0,\infty]} = \mathbf{0}, \tag{3}$$

where, for all $t \le t'$,

$$\mathrm{H}^{\mathrm{T}}_{[t,t']} = \begin{bmatrix} \mathbf{H}_0^{\mathrm{T}}(t) & & \cdots & \mathbf{H}_{m_s}^{\mathrm{T}}(t+m_s) & \mathbf{0} & & \\ \mathbf{0} & \mathbf{H}_0^{\mathrm{T}}(t+1) & \cdots & & \mathbf{H}_{m_s}^{\mathrm{T}}(t+m_s+1) & \ddots & \\ \vdots & \mathbf{0} & \ddots & \vdots & \vdots & \ddots & \\ & & \ddots & \mathbf{H}_0^{\mathrm{T}}(t') & \cdots & & \mathbf{H}_{m_s}^{\mathrm{T}}(t'+m_s) \end{bmatrix}$$

is a transposed parity check matrix, also called the syndrome former of the convolutional code $\mathcal{C}$. The submatrices $\mathbf{H}_i(t)$, $i = 0, 1, \cdots, m_{\mathrm{s}}$, are binary $(c-b) \times c$ submatrices given by

$$\mathbf{H}_i(t) = \begin{bmatrix} h_i^{(1,1)}(t) & \cdots & h_i^{(1,c)}(t) \\ \vdots & & \vdots \\ h_i^{(c-b,1)}(t) & \cdots & h_i^{(c-b,c)}(t) \end{bmatrix}. \tag{4}$$

They satisfy the following properties:

1) $\mathbf{H}_i(t) = \mathbf{0},\ i < 0$ and $i > m_s,\ \forall\ t$.
2) There is a t such that $\mathbf{H}_{m_{\mathrm{s}}}(t) \neq \mathbf{0}$.

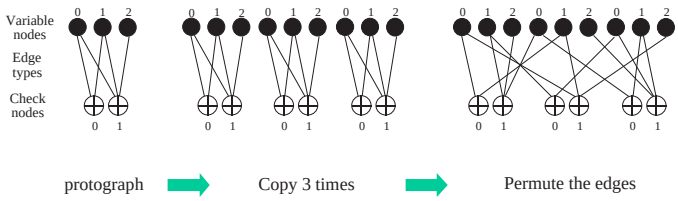


Fig. 1. Copy and permute operation for a protograph to generate larger graphs.

3) $\mathbf{H}_0(t) \neq \mathbf{0}$ and has full rank $\forall\ t$.

We call m_{s} the syndrome former memory and $\nu_{\mathrm{s}} = (m_{\mathrm{s}} + 1) \cdot c$ is the overall constraint length. These parameters determine the height of the nonzero diagonal region of $\mathbf{H}^{\mathrm{T}}_{[0,\infty]}$. The sparsity of the syndrome former is ensured by demanding that its columns have very low Hamming weight, i.e., $w_H(\mathbf{h}_i) << (m_{\mathrm{s}}+1)\cdot c,\ i \in \mathcal{Z}^+$, where $\mathbf{h}_i$ denotes the i-th column of $\mathbf{H}^{\mathrm{T}}_{[0,\infty]}$. The code is said to be regular if its syndrome former $\mathbf{H}^{\mathrm{T}}_{[0,\infty]}$ has exactly J ones in every row and K ones in every column, starting from the $(m_{\mathrm{s}} \cdot (c-b)+1)$-th column. The other entries are zeros. We refer to a code with these properties as an (m_{s}, J, K)-regular LDPC convolutional code, and we note that in general the code is time-varying and has rate $R = 1 - J/K$. An (m_{s}, J, K)-regular time-varying LDPC convolutional code is periodic with period T if $\mathbf{H}_i(t)$ is periodic, i.e., $\mathbf{H}_i(t) = \mathbf{H}_i(t+T), \forall\ i, t$, and if $\mathbf{H}_i(t) = \mathbf{H}_i, \forall\ i, t$, the code is time-invariant.

We will describe a construction technique, similar to the one given in [1], for deriving a periodically time-varying parity-check matrix $\mathbf{H}_{[0,\infty]}$ from the parity-check matrix of an LDPC block code in Section V. The LDPC convolutional codes considered in this paper are derived from the well-known classes of ARA- and protograph-based LDPC block codes.

III. ARA-BASED LDPC CODES

A protograph [3] is a Tanner graph with a relatively small number of nodes. A protograph $G = (V, C, E)$ consists of a set of variable nodes V, a set of check nodes C, and a set of edges E. Each edge $e \in E$ connects a variable node $v_e \in V$ to a check node $c_e \in C$. Parallel edges are permitted, so the mapping $e \rightarrow (v_e, c_e) \in V \times C$ is not necessarily 1:1. As a simple example, we consider the protograph shown in Fig. 1.

This graph consists of 3 variable nodes and 2 check nodes, connected by 5 edges. In this example we have 5 edge types, i.e., each edge in the base protograph represents an edge type. For multi-edge LDPC codes, a group of edges (the number of edges in each group can be different) represents an edge type. For unstructured irregular LDPC codes, there is only one edge type. Having the base protograph, we can obtain a larger graph by a "copy-and-permute" operation as shown in Fig. 1. This operation consists of first making N copies of the protograph and then permuting the endpoints of each edge type among the N variable and N check nodes connected to the set of N edges copied from the same edge type in the protograph. The derived or lifted graph is the graph of a code N times as large as the code corresponding to the protograph, with the same rate and the same distribution of variable and check node degrees.

As can be seen from the protograph representation in the figures, those variable nodes, say n_{trans} nodes, that are connected to the channel (transmitted nodes) will be shown as dark filled circles. Those variable nodes that are not connected to the channel (punctured nodes or not transmitted nodes) will be depicted by blank circles. The check nodes will be depicted by circles with a plus sign inside. The code rate for the protograph is $R = \frac{n_v - n_c}{n_{trans}}$, provided that the parity check matrix of the derived or lifted graph is full rank.

As an example, consider the rate-1/2 systematic repeat-accumulate (RA) code with repetition 3 shown in Fig. 2.

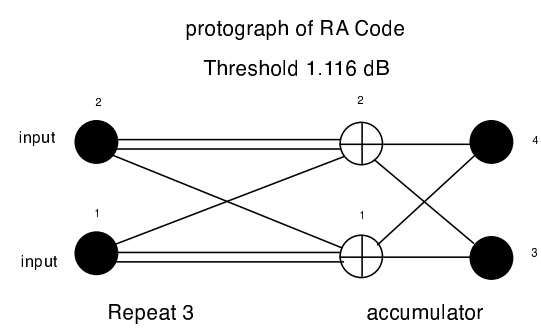


Fig. 2. Protograph for a rate 1/2 RA code.

In [4] it was shown that the threshold can be further improved by precoding the repetition code with an accumulator. The design of the precoder in [4] was guided by an analysis of the extrinsic signal-to-noise ratio (SNR) behavior of repetition codes and punctured accumulator codes using density evolution.

The use of a rate-1 accumulator as a precoder dramatically improves the extrinsic SNR behavior of a repetition 3 outer code in the high extrinsic SNR region and hence improves the iterative decoding threshold of the overall code.

An RA code with an accumulator precoder is called an Accumulate-Repeat-Accumulate (ARA) code [4]. An

example of a simple rate-1/2 ARA code and its corresponding threshold is shown in Fig. 3. The ARA encoder in Fig. 3 uses a punctured accumulator as the precoder.

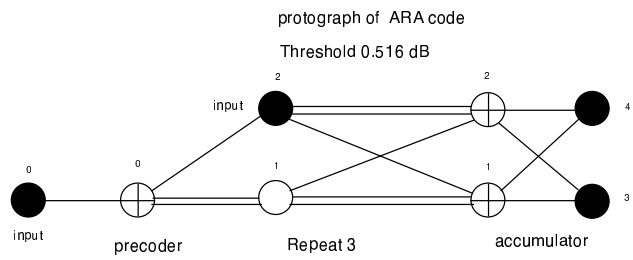


Fig. 3. Protograph for a rate-1/2 ARA code.

In an ARA code protograph the number of degree 2 variable nodes is equal to the number of inner checks (checks that are connected to these degree 2 variable nodes). If we decrease the number of degree 2 variable nodes with respect to inner checks, then the ensemble asymptotic minimum distance of the code may grow with n. For example, if we replace 50% of the degree 2 variable nodes with degree 3 variable nodes, then the minimum distance grows with n. We call such constructed codes ARJA [5] [6] codes because the inner accumulator now has a "Jagged" appearance. The protograph of a rate-1/2 AR4JA code and its corresponding threshold is shown in Fig. 4. We call this protograph AR'4'JA due to the repetition by 4 on the right hand side of the punctured degree-6 variable node.

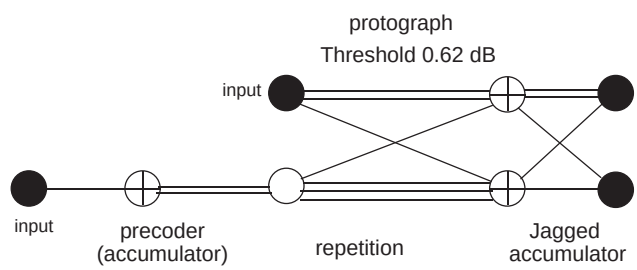


Fig. 4. The rate-1/2 AR4JA protograph.

IV. Decoding of LDPC Convolutional Codes

LDPC convolutional codes can be iteratively decoded using a message-passing algorithm. Although the corresponding Tanner graph has an infinite number of nodes, the distance between two variable nodes that are connected to the same check node is limited by the syndrome former memory. This allows continuous decoding with a decoder that operates on a finite window sliding along the received sequence, similar to a Viterbi decoder with finite path memory [7], [8]. The decoding of two variable nodes that are at least $(m_{\mathrm{s}}+1)$ time units apart can be performed independently, since the corresponding bits cannot participate in the same parity-check equation. This allows the parallelization of the I_C iterations by employing I_C independent identical processors working on different regions of the Tanner graph simultaneously. A pipeline decoder that is based on this idea was introduced by Jiménez-Felström and Zigangirov in [1]. The operation of this decoder on the Tanner graph for a simple time-invariant rate $R = 1/3$ LDPC convolutional code with $m_{\mathrm{s}} = 2$ is shown in Fig. 5.

Assuming c received symbols enter the pipeline decoder per unit time, it takes $I_C \cdot (m_{\mathrm{s}}+1)$ time units for this c-tuple to reach the output of the decoder, where a decision on these symbols is made. Thus, once a decoding delay of $I_C \cdot (m_{\mathrm{s}}+1)$ time units has elapsed, the decoder produces a continuous output stream, i.e., at each time unit, c newly received symbols enter the decoder and c decoded bits leave it. The I_C processors perform the I_C decoding iterations in parallel. At each time unit, the i-th processor begins by activating the first column of $(c-b)$ check nodes in its operating region and then proceeds to activate the last column of c variable nodes that are about to leave the operating region. A check node activation is the step where the check node collects all the incoming messages from its neighboring variable nodes, calculates the outgoing messages, and sends the new messages to each neighboring variable node. Correspondingly, during a variable node activation, all the incoming messages to a variable node from the neighboring check nodes are collected and the new outgoing messages are calculated. Then the new messages are sent to each neighboring check node.

This message passing schedule corresponds to a parallel message passing schedule of a block code decoder where all variable nodes are updated at once and then all check nodes are activated. We will discuss a basis for comparison between LDPC block and convolutional decoding in Section VI.

V. Derivation of Convolutional Codes From Block Codes

In this section, we present a graphical unwrapping procedure based on the binary parity-check matrix $\mathbf{H}$ of an arbitrary LDPC block code.

In order to derive a periodically time-varying LDPC convolutional code of rate $R = b/c$ from an LDPC block code, we cut the binary parity-check matrix $\mathbf{H}$ of the

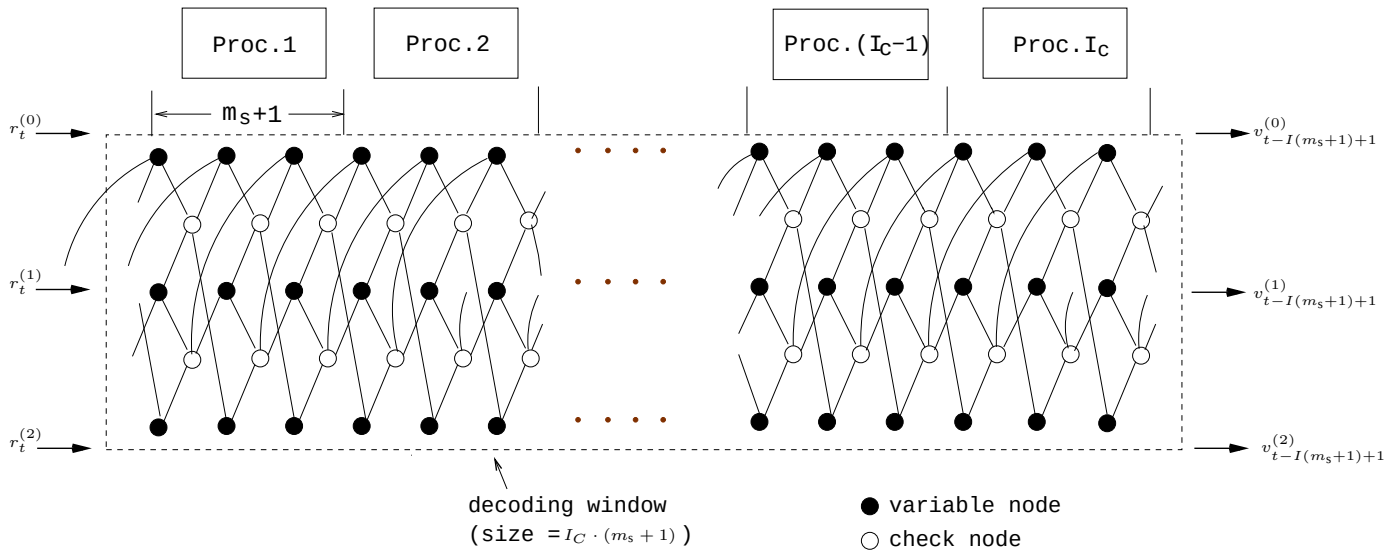


Fig. 5. The Tanner graph of an R=1/3 LDPC convolutional code and an illustration of pipeline decoding.

underlying block code along the diagonal in steps of size $(c-b)\times c$. Next, we remove the upper-diagonal portion and paste it to the bottom of the lower-diagonal portion. The resulting diagonal shaped matrix is then repeated indefinitely, giving us the parity-check matrix $\mathbf{H}_{\text{conv}}$ of a time-varying convolutional code with period $T = m_{\text{s}}+1$. We refer to this cutting and pasting operation as the unwrapping procedure. In general, if we start from an (n, J, K)-regular LDPC block code of length n and rate $R \geq 1 - J/K$, we obtain an (m_{s}, J, K)-regular LDPC convolutional code with rate $R = b/c = 1 - J/K$ and syndrome former memory $m_{\text{s}} = (n/c) - 1$, i.e., overall constraint length $\nu_{\text{s}} = n$. This procedure is illustrated in Fig. 6, where we derive a rate $R = 1/3$, $(6, 2, 3)$-regular LDPC convolutional code from a rate $R = 8/21$, $(21, 2, 3)$-regular LDPC block code, where in this case the H matrix of the block code contains one redundant row.

The unwrapping step size of $(c-b)\times c$ produces a rate $R = b/c$ convolutional code. Similarly, a step size of $(c-b)k \times ck$, where $0 < ck \leq n$ and $k \in \mathcal{Z}^+$, produces a rate bk/bc code. The special case of $ck = n$, for example, corresponds to repeating the original block code indefinitely, and is therefore of no practical significance. On the other hand, this special case helps to show the connection between the block and convolutional codes and illustrates that by decreasing the value of k we arrive at a "more convolutional" structure.

Although the above example uses regular LDPC block codes as the starting point of the unwrapping procedure, there is, in general, no constraint on the row and column weights of the parity-check matrices, and the derived time-varying convolutional code has the exact same node degree distribution as that of the underlying block code. In other words, the unwrapping procedure preserves the degree distribution during the cutting and pasting (and also the repeating) steps, so the same approach can be employed to derive irregular LDPC convolutional codes from irregular LDPC block codes.

VI. Decoding Comparisons

In this section, we compare several aspects of decoding LDPC convolutional and block codes.

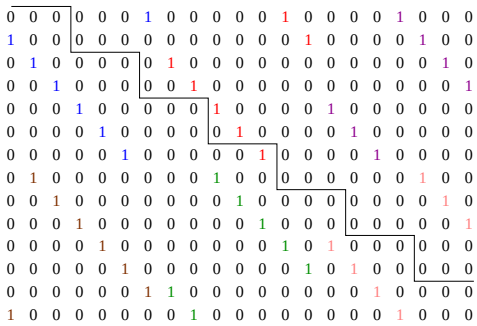
A. Computational Complexity

Let C_{check} (C_{var}) denote the number of computations required for a degree-K (J) check (variable) node update. Regardless of the code structure, C_{check} and C_{var} depend only on the values J and K.

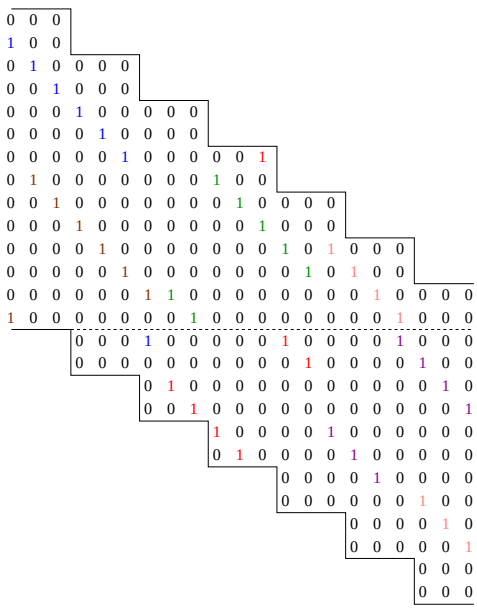
For a rate $R = b/c$, (m_{s}, J, K)-regular LDPC convolutional code and a pipeline decoder with I_C iterations/processors, at every time instant each processor activates $c-b$ check nodes and c variable nodes. The computational complexity per decoded bit is therefore given by

$$\begin{aligned} C_{bit}^{conv} &= ((c-b)\cdot C_{check} + c\cdot C_{var})\cdot I_C/c \quad (5)\\ &= ((1-R)\cdot C_{check} + C_{var})\cdot I_C, \end{aligned}$$

which is independent of the constraint length ν_{s}.



(a)



(b)

Fig. 6. Deriving a time-varying LDPC convolutional code from an LDPC block code: (a) H matrix for the block code, (b) $\mathbf{H}_{\text{conv}}$ matrix for the convolutional code after unwrapping.

Similarly, the decoding complexity for a rate $R \geq 1 - J/K$, (n, J, K)-regular LDPC block code with I_B iterations is given by

$$\begin{aligned} C_{bit}^{block} &= (n \cdot \frac{J}{K} \cdot C_{check} + n \cdot C_{var}) \cdot I_B / n \quad (6) \\ &= (\frac{J}{K} \cdot C_{check} + C_{var}) \cdot I_B \\ &= ((1 - R) \cdot C_{check} + C_{var}) \cdot I_B, \end{aligned}$$

which is again independent of the code length n. Thus the difference in computational complexity between block and convolutional codes (on a per decoded bit basis) is given by the ratio I_B / I_C.

B. Internal Observation Memory

The sliding window decoder implementation of an LDPC convolutional code requires the storage of $I_C \cdot \nu_s$ symbols. However, since decoding can be carried out by pipelining I_C identical independent parallel processors, each operating on only ν_s symbols, the smaller size of the individual processors may be useful in simplifying hardware design and reducing routing congestion. For an LDPC block code of length n, the processor must be capable of storing all n symbols.

C. Internal Edge Memory

For the pipeline decoder, we need a storage element for each edge in the corresponding Tanner graph. Thus a total of $I_C \cdot (J) \cdot \nu_s$ storage elements are required for I_C iterations of decoding. Similarly, we need $n \cdot (J)$ storage elements for the decoding of an LDPC block code of length n.

D. Decoding Delay (Latency) and External Observation Memory

Let T_s denote the time between the arrival of successive symbols, i.e., the symbol rate is $1/T_s$. Then the maximum time from the arrival of a symbol until it is decoded in the pipeline decoder is

$$\Delta_{io}^{conv} = ((c - 1) + (m_s + 1) \cdot c \cdot I_C) \cdot T_s. \quad (7)$$

The first term $(c - 1)$ in (7) represents the time between the arrival of the first and last of the c encoded symbols output by a rate $R = b/c$ convolutional encoder in each encoding interval. The dominant second term $(m_s + 1) \cdot c \cdot I_C$ is the time each symbol spends in the decoding window. Since c symbols are loaded into the decoder simultaneously, the pipeline decoder requires a buffer to hold the first $(c - 1)$ symbols.

With LDPC block codes, data is typically transmitted in a sequence of blocks. Depending on the data rate and the processor speed, several scenarios are possible. We consider the best case for block codes, i.e., each block is decoded by the time the last bit of the next block arrives. This results in an input-output delay of $\Delta_{io}^{block} = 2n \cdot T_s$. Note that the block decoder needs a buffer to hold the $n - 1$ symbols that arrive before the next block is complete. This scenario is overly optimistic since some blocks will consume significantly more than the average number of iterations to decode, and therefore more than n symbols should be accommodated in the decoder's input buffer. Fogal, Dolinar, and Andrews [9] have shown an additional block of storage reduces the likelihood of buffer overflow below typical frame error rates at operating SNR's of interest. Therefore, we choose $\Delta_{io}^{block} = 3n \cdot T_s$ as an upper bound on block code decoding latency.

VII. A Basis for Comparison

As noted in the previous section, there are many ways to compare possible decoder realizations for LDPC block and convolutional codes. However, no one approach tells the whole story. This is due in part to the very different structures of block and convolutional codes. In fact, even before the LDPC coding era, it has always been a controversial topic to determine how best to compare block and convolutional codes. Notions such as trellis complexity, minimum distance bounds, and error exponents have all been employed to this end. Similarly, it is very hard to determine an ideal basis of comparison between LDPC block and convolutional codes. In the final analysis, any comparison must be a function of the particular application, e.g., while a large storage requirement or processor size might not be problems for mobile communications, a large latency may be undesirable, especially for real-time voice transmission.

In this paper, we choose a comparison method based on decoded bit error rate (BER) performance. For each of the code constructions, we target a specific channel SNR at a fixed distance from the iterative decoding threshold of the employed code family. We then compare LDPC block and convolutional codes that achieve the same BER and/or frame error rate (FER) performance at this SNR based on the criteria presented in the previous section.

VIII. Comparison of Block and Convolutional Codes Built from the AR4JA Protograph

Our comparison begins with the AR4JA protograph of Section III. Before constructing either convolutional or block LDPC codes, we expand the graph of Fig. 4 by a factor of 4 to remove double edges. This working 'base' graph is given the name AR4JAx4. The construction of two unique[1] LDPC convolutional codes is described in Figs. 7(a)(b). In both cases, a block code is first derived via expansion by a factor of 128, Fig. 7(a), and expansion by a factor of 125, Fig. 7(b). The number of rows and columns in the transposed parity-check matrix of each of these block codes is indicated in the figure. Note that permutations associated with the aforementioned expansions (x4, x125, x128) were performed using the well-known progressive edge growth technique [10]. The performance of these two block codes is denoted by the $k = 1000$ and $k = 1024$ curves in Fig. 8.

[1]These convolutional codes are unique, but are constructed to achieve relatively 'comparable' convolutional LDPC codes

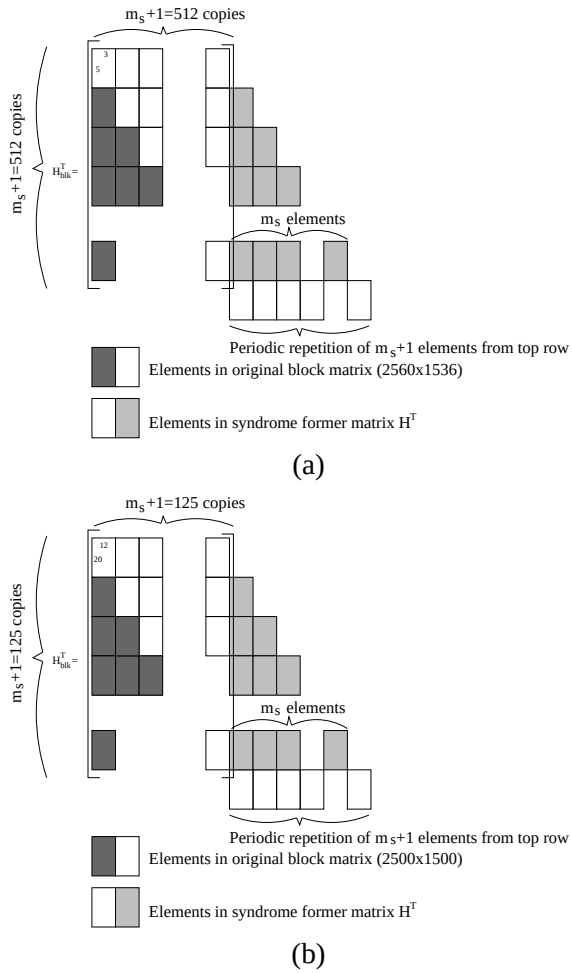


Fig. 7. Deriving an LDPC convolutional code from an LDPC block code: (a) AR4JAx4x128, k=1024, n=2048 (+ 512 punctured nodes) protograph unwrapping. (b) AR4JAx4x125 protograph unwrapping.

These two block codes (with k roughly 1000) are 'unwrapped' to form LDPC convolutional codes, again following the parameters denoted in Fig. 7. For instance, the AR4JAx4x128 (Fig. 7(a)) block code is broken into sub-blocks of size $c = 5$ and $c - b = 3$, while the AR4JAx4x125 (Fig. 7(b)) block code is broken into sub-blocks of size $c = 20$ and $c - b = 12$. Note that the latter dimensions adhere to those inherent in the AR4JAx4 base graph. An additional factor is that the AR4JA code has all degree-6 nodes punctured (see Fig. 4). Therefore, in terms of transmitted nodes, Fig. 7(a) has $c_{trans} = 4$ and rate $R = (c - (c - b))/c_{trans} = 1/2$. Similarly, Fig. 7(b) has $c_{trans} = 16$ and rate $R = (c - (c - b))/c_{trans} = 1/2$.

Much of the interest associated with LDPC convolutional codes stems from the increase in coding gain that can be obtained through simple periodic repetition, by a factor of I_C, of graphs like those shown

	AR4JAx4x125 Conv	AR4JAx4x128 Conv	AR4JAx4x2048 Blk
Complexity Per Decoded Bit [2-input Ops]	$12I_C$=1200	$12I_C$=1200	$12I_B$=436
Observation Memory [Depth]	$\nu_s I_C$=250000	$\nu_s I_C$=256000	n=40960
Edge Storage Memory [Depth]	$\nu_s J_{avg} I_C$=750000	$\nu_s J_{avg} I_C$=768000	nJ_{avg}=122880
Latency [T_s]	$\nu_s I_C$=250000	$\nu_s I_C$=256000	$3n$=122880

TABLE I

COMPLEXITY MEASURES FOR CONVOLUTIONAL AND BLOCK LDPC CODES BUILT FROM IDENTICAL PROTOGRAPHS WITH COMPARABLE BER VS. SNR PERFORMANCE.

in Figs. 7(a)(b) ($I_C = 2$ in the figure). For instance, while the performance of the block codes underlying the convolutional codes in Figs. 7(a)(b) is given by the circle and square marked lines in Fig. 8, periodic repetition by a factor of $I_C = 100$ of the unwrapped versions of these block codes yields the performance shown by the diamond and pentagram marked lines in the same figure. At a BER=10^{-6}, nearly a full decibel of performance is gained without the expense of any additional rate whatsoever. A main motivation of this paper is to examine whether or not the cost of the performance gained by convolutional constraint length repetition is more or less than the cost of the gain associated with further expansion of the underlying block code. Note that the performance of any code constructed via repetition of the AR4JA protograph can do no better than the protograph's threshold, which is $E_b/N_o = 0.62$ dB.

Based on the simulation results shown in Fig. 8, we see that the performance of AR4JAx4x2048 (the AR4JAx4 base graph expanded via circulant permutation by a factor of 2048, which has $k = 16384$ and $n_{trans} = 32768$ ($n = 40960$)) coincides nearly exactly with the performance of the $I_C = 100$ unwrapped convolutional codes at $E_b/N_o = 1.0$ dB and BER = 10^{-7}. In addition, the average number of iterations, $I_B = 36.3$, performed by the block decoder is given for this SNR and others across an operating region of interest in Fig. 9. We can apply the complexity analysis outlined for regular codes in Section VI to AR4JA based codes with a few additional assumptions. First, let the parameters K and J (the constraint and variable node degrees) be denoted by $K_{avg} = 5$ and $J_{avg} = 3$. Second, assume that a degree K_{avg} constraint node requires $C_{check} = 2K_{avg}$ 2-input operations to perform an update; similarly a degree J_{avg} variable node requires $C_{var} = 2J_{avg}$ 2-input operations (see, for instance, [11]). Given these assumptions, we refer the reader to Table I.

Without biasing results to one approach over the other, the results in the table assume that all variable nodes in

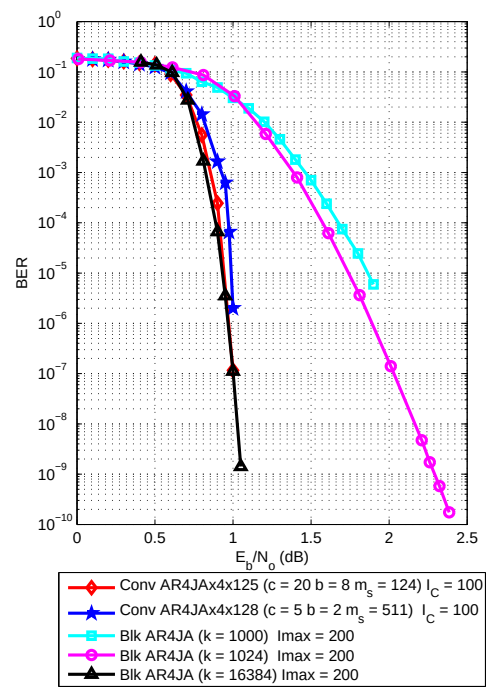


Fig. 8. Rate-1/2 convolutional and block LDPC codes based on the AR4JA protograph.

the code are transmitted and must be buffered in the decoding process. For essentially equal performance the block code requires 6 times less internal observation and edge storage memory and incurs at worst (due to the upper bound of $3n$ on block code latency) 2 times less decoding latency.

Table I also shows that the block code requires 2.57 times fewer computations per decoded bit than the convolutional code, where we note that the number of computations performed is proportional to the number of iterations. (Because of the identical graph connectivity of the LDPC block codes and the LDPC convolutional codes derived from them, they perform the same number of computations if they use the same number of iterations.) However, for the simulation results shown in Fig. 8, the LDPC convolutional codes employed a

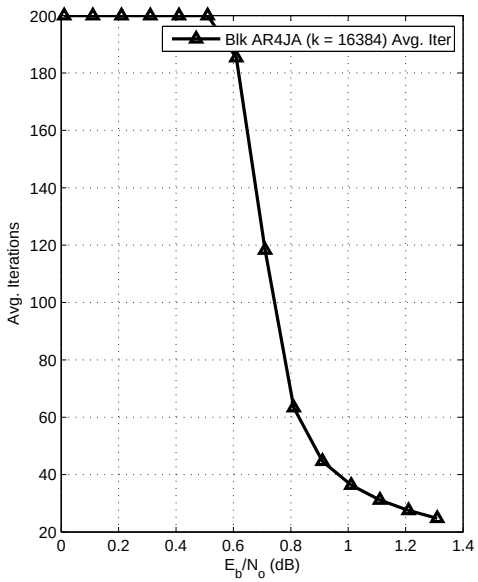


Fig. 9. Average decoding iterations vs. SNR for Rate-1/2 AR4JA with k = 16384.

constant number of $I_C = 100$ iterations, while the LDPC block codes made use of a syndrome-based stopping rule to decrease their average number of iterations to $I_B = 36.3$, with the maximum number of iterations fixed at $I_B = 200$. This leads to the difference in the number of required computations noted above.

As a result of the extended graph of LDPC convolutional codes, stopping rules are not as obvious as for LDPC block codes. One such stopping rule has been proposed, however, in [12], where the average number of iterations can be reduced without affecting the error performance. Using this rule, the independent processors in the pipeline decoder can sometimes be put into a "sleep" mode in order to save computations. Adjusting the above computational comparison to take advantage of this pipeline decoding stopping rule is currently being investigated.

We note that the size of a pipeline decoding 'processor' is roughly $1/I_C$ times that any of the convolutional decoder complexity measures in Table I. Convolutional LDPC codes utilizing a pipeline decoder therefore lend themselves to a fine granularity that allows a trade-off between performance and complexity. The simulation results of Fig. 8 indicate that it is possible to achieve low error rate performance by keeping the processor size ν_s relatively small while increasing the number of processors I_C. (In Fig. 8, for example, the convolutional code processor is only $1/16$ the size of the block code processor.) The resulting cascade of relatively small processors may allow for higher clock frequencies and provide higher throughputs than is possible with a single block decoder. We note that pipelined LDPC block decoders can also be designed to use individual processors, with each processor performing only a fraction of the total number of iterations. In this case, however, the block code would lose the storage memory and decoding latency advantages noted above compared to their convolutional counterparts.

IX. Conclusion

An advantage of the convolutional structure is that performance can easily be gained by simply adding more constraint length multiples (iterations I_C) at the decoder without changing the encoding structure at all. Also, the natural pipeline structure of the convolutional decoder facilitates the realization of low error rates and potentially high throughputs by designing a processor of relatively modest size and replicating it in the pipeline. However, if one has a particular target error rate in mind, designing a dedicated block code achieves the desired result with less latency and reduced memory requirements.

We are currently initiating a practical hardware-based complexity comparison of the LDPC codes presented in this paper. This includes realizations of the LDPC block decoding and pipelined LDPC convolutional decoding architectures on field programmable gate arrays (FPGAs) and application specific integrated circuits (ASICs). These realizations will provide further insight into the comparisons presented in this paper, including determining possible practical implementation bottlenecks in terms of memory and logic element usage.

Acknowledgment

This work was supported in part by NSF Grants CCR02-05310, CCF05-15012, NASA Grant NNG05GH73G, and at the Jet Propulsion Laboratory / California Institute of Technology under a contract with NASA.

References

[1] A. Jiménez-Feltström and K. Sh. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inform. Theory*, vol. IT-45, pp. 2181–2191, Sept. 1999.

[2] D. J. Costello, Jr., A. E. Pusane, S. Bates, and K. Sh. Zigangirov, "A comparison between LDPC block and convolutional codes," in *Proc. Information Theory and Applications Workshop*, (San Diego, CA, USA), February 2006.

[3] J. Thorpe, "Low density parity check (LDPC) codes constructed from protographs," *JPL IPN Progress Report*, pp. 42–154, August 2003.

[4] A. Abbasfar, D. Divsalar, and K. Yao, "Accumulate repeat accumulate codes," in *Proc. IEEE Int. Symp. Inf. Theory*, (Dallas, TX, USA), July 2004.

[5] D. Divsalar, S. Dolinar, and C. Jones, "Protograph based LDPC codes with minimum distance growing linearly with block size," in *Proc. IEEE Global Communications Conference (GlobeCom)*, (St. Louis, MO, USA), Nov. 2005.

[6] D. Divsalar, S. Dolinar, and C. Jones, "Low-rate LDPC codes with simple protograph structure," in *Proc. IEEE Int. Symp. Inf. Theory*, (Adelaide, South Australia, Australia), June 2005.

[7] R. Johannesson and K. Sh. Zigangirov, *Fundamentals of Convolutional Coding*. Piscataway, NJ: IEEE Press, 1999.

[8] S. Lin and D. J. Costello, Jr., *Error Control Coding*. Englewood Cliffs, NJ: Prentice-Hall, 2nd ed., 2004.

[9] S. Fogal, S. Dolinar, and K. Andrews, "Buffer analysis for LDPC decoders," in *Submitted to ISIT*, 2007.

[10] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth Tanner graphs," in *Proc. IEEE Global Telecommun. Conf.*, (San Antonio, TX), pp. 995–1001, Nov. 2001.

[11] C. Jones, E. Valles, M. Smith, and J. Villasenor, "Approximate-min* constraint node updating for LDPC code decoding," in *Proc. IEEE Military Communications Conference (MILCOM)*, (Boston, Massachusetts, USA), Oct. 2004.

[12] A. E. Pusane, A. Jiménez-Feltström, A. Sridharan, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., "Implementation aspects of LDPC convolutional codes," *submitted to IEEE Trans. Commun.*, Nov. 2005.