# Deriving Good LDPC Convolutional Codes from LDPC Block Codes

Ali E. Pusane, *Member, IEEE*, Roxana Smarandache, *Member, IEEE*, Pascal O. Vontobel, *Member, IEEE*, and Daniel J. Costello, Jr., *Life Fellow, IEEE*

*Dedicated to the memory of our dear friend and colleague Ralf Koetter (1963–2009)*

***Abstract*—Low-density parity-check (LDPC) convolutional codes are capable of achieving excellent performance with low encoding and decoding complexity. In this paper, we discuss several graph-cover-based methods for deriving families of time-invariant and time-varying LDPC convolutional codes from LDPC block codes and show how earlier proposed LDPC convolutional code constructions can be presented within this framework. Some of the constructed convolutional codes significantly outperform the underlying LDPC block codes. We investigate some possible reasons for this "convolutional gain," and we also discuss the—mostly moderate—decoder cost increase that is incurred by going from LDPC block to LDPC convolutional codes.**

***Index Terms*—Block codes, convolutional codes, low-density parity-check (LDPC) codes, message-passing iterative decoding, pseudocodewords, pseudoweights, quasi-cyclic codes, unwrapping, wrapping.**

## I. Introduction

In the last 15 years, the area of channel coding has been revolutionized by the practical realization of capacity-approaching coding schemes, initiated by the invention of turbo codes and their associated decoding algorithms in 1993 [1]. A few years after the invention of the turbo coding schemes, researchers became aware that Gallager's low-density parity-check (LDPC) block codes and message-passing iterative decoding, first introduced in [2], were also capable of capacity-approaching performance. The analysis and design of these coding schemes quickly attracted considerable attention in the literature, beginning with the work of Wiberg [3], MacKay and Neal [4], and many others. An irregular version of LDPC codes was first introduced by Luby *et al.* in [5], [6], and analytical tools were presented in [7] and [8] to obtain performance limits for graph-based message-passing iterative decoding algorithms, such as those suggested by Tanner [9]. For many classes of channels, these tools have been successfully employed to design families of irregular LDPC codes that perform very well near capacity [10], [11]. Moreover, for the binary erasure channel these tools have enabled the design of families of irregular LDPC codes that are not only capacity-approaching but in fact capacity-achieving (see [12] and references therein).

The convolutional counterparts of LDPC block codes are LDPC convolutional codes. 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. Recent studies have shown that LDPC convolutional codes are suitable for practical implementation in a number of different communication scenarios, including continuous transmission and block transmission in frames of arbitrary size [13]–[15].

Two major methods have been proposed in the literature for the construction of LDPC convolutional codes, two methods that in fact started the field of LDPC convolutional codes. The first method was proposed by Tanner [16] (see also [17] and [18]) and exploits similarities between quasi-cyclic block codes and time-invariant convolutional codes. The second method was presented by Jiménez-Feltström and Zigangirov [19] and relies on a matrix-based unwrapping procedure to obtain the parity-check matrix of a periodically time-varying convolutional code from the parity-check matrix of a block code.

The aims of this paper are threefold. First, we show that these two LDPC convolutional code construction methods, once suitably generalized, are in fact tightly connected. We establish this connection with the help of so-called graph

Manuscript received April 25, 2010; revised August 18, 2010; accepted September 21, 2010. Date of current version January 19, 2011. The work of A. E. Pusane and D. J. Costello, Jr. was supported in part by the National Science Foundation (NSF) under Grants CCR-0205310 and CCF-0830650, and by NASA under Grant NNX09AI66G. The work of A. E. Pusane was also supported by a Graduate Fellowship from the Center for Applied Mathematics, University of Notre Dame, Notre Dame, IN. The work of R. Smarandache was supported by the NSF under Grants DMS-0708033 and CCF-0830608, and in part by the NSF under Grant CCR-0205310. The material in this paper was presented in part at the IEEE International Symposium on Information Theory, Nice, France, June 2007.

This paper is part of the special issue on "Facets of Coding Theory: From Algorithms to Networks," dedicated to the scientific legacy of Ralf Koetter.

A. E. Pusane was with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA. He is now with the Department of Electrical and Electronics Engineering, Bogazici University, Bebek, Istanbul 34342, Turkey (e-mail: ali.pusane@boun.edu.tr).

R. Smarandache is with the Department of Mathematics and Statistics, San Diego State University, San Diego, CA 92182 USA (e-mail: rsmarand@sciences.sdsu.edu).

P. O. Vontobel is with Hewlett-Packard Laboratories, Palo Alto, CA 94304 USA (e-mail: pascal.vontobel@ieee.org).

D. J. Costello, Jr. is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: costello.2@nd.edu).

Communicated by G. D. Forney, Jr., Associate Editor for the special issue on "Facets of Coding Theory: From Algorithms to Networks."

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2010.2095211

$$\bar{\boldsymbol{H}}_{\text{conv}} = \begin{bmatrix} \boldsymbol{H}_0(0) & & & & & & & \\ \boldsymbol{H}_1(1) & \boldsymbol{H}_0(1) & & & & & & \\ \vdots & \vdots & \ddots & & & & & \\ \boldsymbol{H}_{m_\text{s}}(m_\text{s}) & \boldsymbol{H}_{m_\text{s}-1}(m_\text{s}) & \cdots & \boldsymbol{H}_0(m_\text{s}) & & & & \\ & \boldsymbol{H}_{m_\text{s}}(m_\text{s}+1) & \boldsymbol{H}_{m_\text{s}-1}(m_\text{s}+1) & \cdots & \boldsymbol{H}_0(m_\text{s}+1) & & & \\ & & \ddots & \ddots & & \ddots & & \\ & & & \boldsymbol{H}_{m_\text{s}}(t) & \boldsymbol{H}_{m_\text{s}-1}(t) & \cdots & \boldsymbol{H}_0(t) & \\ & & & & \ddots & & & \ddots \end{bmatrix}. \tag{1}$$

covers.[1] A second aim is to discuss a variety of LDPC convolutional code constructions. Although the underlying principles are mathematically quite straightforward, it is important to understand how they can be applied to obtain convolutional codes with good performance and attractive encoder and decoder architectures. A third aim is to make progress towards a better understanding of where the observed "convolutional gain" comes from, and what its costs are in terms of decoder complexity.

The paper is structured as follows. After some notational remarks in Section I-A, we discuss the basics of LDPC convolutional codes in Section II. In particular, in that section, we give a first exposition of the LDPC convolutional code construction methods due to Tanner and due to Jiménez-Feltström and Zigangirov. In Section III, we discuss two types of graph-cover code constructions and show how they can be used to connect the code construction methods due to Tanner and due to Jiménez-Feltström and Zigangirov. Based on these insights, Section IV presents a variety of LDPC convolutional code constructions (along with simulation results), and in Section V, we mention some similarities and differences of these constructions compared to other recent code constructions in the literature. Afterwards, in Section VI, we analyze some aspects of the constructed LDPC convolutional codes and discuss some possible reasons for the "convolutional gain," before we conclude the paper in Section VII.

### A. Notation

We use the following sets, rings, and fields: $\mathbb{Z}$ is the ring of integers; $\mathbb{Z}_{\geqslant 0}$ is the set of nonnegative integers; $\mathbb{F}_2$ is the field of size two; $\mathbb{F}_2[X]$ is the ring of polynomials with coefficients in $\mathbb{F}_2$ and indeterminate $X$; and $\mathbb{F}_2[X]/\langle X^r-1\rangle$ is the ring of polynomials in $\mathbb{F}_2[X]$ modulo $X^r-1$, where $r$ is a positive integer. We also use the notational shorthand $\mathbb{F}_2^{\langle r\rangle}[X]$ for $\mathbb{F}_2[X]/\langle X^r-1\rangle$.

[1]Note that graph covers have been used in two different ways in the LDPC code literature. On the one hand, starting with the work of Tanner [20], they have been used to construct Tanner graphs [9] of LDPC codes, and therefore parity-check matrices of LDPC codes. Codes constructed in this way are nowadays often called proto-graph-based codes, following the influential work of Thorpe [21], who formalized this code construction approach. On the other hand, starting with the work of Koetter and Vontobel [22], [23], finite graph covers have been used to analyze the behavior of LDPC codes under message-passing iterative decoding. In this paper, we will use graph covers in the first way, with the exception of some comments on pseudocodewords.

By $\mathbb{F}_2^n$ and $\mathbb{F}_2^{\langle r\rangle}[X]^n$, we mean, respectively, a row vector over $\mathbb{F}_2$ of length $n$ and a row vector over $\mathbb{F}_2^{\langle r\rangle}[X]$ of length $n$. In the following, if $\boldsymbol{M}$ is some matrix, then $[\boldsymbol{M}]_{j,i}$ denotes the entry in the $j$th row and $i$th column of $\boldsymbol{M}$. Note that we use the convention that indices of vector entries start at 0 (and not at 1), with a similar convention for row and column indices of matrix entries. (This comment applies also to semi-infinite matrices, which are defined such that the row and column index sets equal $\mathbb{Z}_{\geqslant 0}$.) The only exception to this convention are bi-infinite matrices, where the row and column index sets equal $\mathbb{Z}$. Finally, $\boldsymbol{M}_1 \otimes \boldsymbol{M}_2$ will denote the Kronecker product of the matrices $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$.

## II. LDPC Convolutional Codes

This section defines LDPC convolutional codes and discusses why they are interesting from an implementation perspective. Afterwards, we review two popular methods of obtaining LDPC convolutional codes by unwrapping block codes. Later in this paper, namely in Section III, we will use graph covers to show how these two methods are connected, and in Section IV we will see how these two methods can be implemented and combined to obtain LDPC convolutional codes with very good performance.

### A. Definition of LDPC Convolutional Codes

A semi-infinite binary parity-check matrix as in (1), shown at the top of the page, defines a convolutional code $\bar{\mathsf{C}}_{\text{conv}}$ as follows. Namely, it is the set of semi-infinite sequences given by

$$\bar{\mathsf{C}}_{\text{conv}} = \left\{ \bar{\boldsymbol{v}} \in \mathbb{F}_2^{\infty} \mid \bar{\boldsymbol{H}}_{\text{conv}} \cdot \bar{\boldsymbol{v}}^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

where $(\,\cdot\,)^{\mathsf{T}}$ denotes the transpose of a vector or of a matrix.

We comment on several important aspects and properties of the code $\bar{\mathsf{C}}_{\text{conv}}$ and its parity-check matrix $\bar{\boldsymbol{H}}_{\text{conv}}$.

- If the submatrices $\boldsymbol{H}_i(t), i = 0, 1, \ldots, m_\text{s}$, $t \in \mathbb{Z}_{\geqslant 0}$, have size $(c-b) \times c$ with $b < c$, then $\bar{\mathsf{C}}_{\text{conv}}$ is said to have (design) rate $R = b/c$.
- The parameter $m_\text{s}$ that appears in (1) is called the syndrome former memory. It is an important parameter of $\bar{\mathsf{C}}_{\text{conv}}$ because the maximal number of nonzero submatrices per block row of $\bar{\boldsymbol{H}}_{\text{conv}}$ is upper bounded by $m_\text{s}+1$.

- The quantity $\nu_{\rm s} = (m_{\rm s}+1)\cdot c$ is called the constraint length of $\bar{\mathsf{C}}_{\rm conv}$. It measures the maximal width (in symbols) of the nonzero area of $\bar{\boldsymbol{H}}_{\rm conv}$.[2]
- We do not require that for a given $i = 0, 1, \ldots, m_{\rm s}$ the submatrices $\{\boldsymbol{H}_i(t)\}_{t\in\mathbb{Z}_{\geqslant 0}}$ are independent of $t$, and so $\bar{\mathsf{C}}_{\rm conv}$ is in general a *time-varying* convolutional code.
- If there is a positive integer $T_{\rm s}$ such that $\boldsymbol{H}_i(t) = \boldsymbol{H}_i(t+T_{\rm s})$ for all $i = 0, 1, \ldots, m_{\rm s}$ and all $t \in \mathbb{Z}_{\geqslant 0}$, then $T_{\rm s}$ is called the period of $\bar{\boldsymbol{H}}_{\rm conv}$, and $\bar{\mathsf{C}}_{\rm conv}$ is *periodically time-varying*.
- If the period $T_{\rm s}$ equals 1, then $\bar{\boldsymbol{H}}_{\rm conv}$ is called *time-invariant*, and the parity-check matrix can be simply written as

$$\bar{\boldsymbol{H}}_{\rm conv} = \begin{bmatrix} \boldsymbol{H}_0 & & & & & \\ \boldsymbol{H}_1 & \boldsymbol{H}_0 & & & & \\ \vdots & \vdots & \ddots & & & \\ \boldsymbol{H}_{m_{\rm s}} & \boldsymbol{H}_{m_{\rm s}-1} & \cdots & \boldsymbol{H}_0 & & \\ & \boldsymbol{H}_{m_{\rm s}} & \boldsymbol{H}_{m_{\rm s}-1} & \cdots & \boldsymbol{H}_0 & \\ & & \ddots & \ddots & & \ddots \end{bmatrix}. \tag{2}$$

- If the number of ones in each row and column of $\bar{\boldsymbol{H}}_{\rm conv}$ is small compared to the constraint length $\nu_{\rm s}$, then $\bar{\mathsf{C}}_{\rm conv}$ is an LDPC convolutional code.
- An LDPC convolutional code $\bar{\mathsf{C}}_{\rm conv}$ is called $(m_{\rm s}, J, K)$-regular if, starting from the zeroth column, $\bar{\boldsymbol{H}}_{\rm conv}$ has $J$ ones in each column, and, starting from the $(m_{\rm s}+1)\cdot(c-b)$th row, $\bar{\boldsymbol{H}}_{\rm conv}$ has $K$ ones in each row. If, however, there are no integers $m_{\rm s}$, $J$, and $K$ such that $\bar{\mathsf{C}}_{\rm conv}$ is $(m_{\rm s}, J, K)$-regular, then $\bar{\mathsf{C}}_{\rm conv}$ is called irregular.

Of course, there is some ambiguity in the above definition. Namely, a periodically time-varying LDPC convolutional code with parameters $T_{\rm s}, b$, and $c$ can also be considered to be a periodically time-varying LDPC convolutional code with parameters $T_{\rm s}' = T_{\rm s}/\ell, b' = \ell\cdot b, c' = \ell\cdot c$, and $R' = b'/c' = b/c$ for any integer $\ell$ that divides $T_{\rm s}$. In particular, for $\ell = T_{\rm s}$ we consider the code to be a time-invariant LDPC convolutional code with parameters $b' = T_{\rm s}\cdot b$ and $c' = T_{\rm s}\cdot c$.

### *B. Implementation Aspects of LDPC Convolutional Codes*

An advantage of LDPC convolutional codes compared to their block code counterparts is the so-called "fast encoding" property. As a result of the diagonal shape of their parity-check matrices, many LDPC convolutional codes enjoy simple shift register based encoders. Even randomly constructed LDPC convolutional codes can be formed in such a way as to achieve this feature without any loss in performance (see, e.g., [19] and [24]). On the other hand, in order to have a simple encoding procedure for LDPC block codes, either the block code must have some sort of structure [25] or the parity-check matrix must be changed to a more easily "encodable" form [26].

The difficulty in constructing and decoding LDPC convolutional codes is dealing with the unbounded size of the parity-check matrix. This is overcome at the code construction step by considering only periodically time-varying or time-invariant codes. The code construction problem is therefore reduced to designing just one period of the parity-check matrix. For decoding, the most obvious approach is to terminate the encoder and to employ message-passing iterative decoding based on the complete Tanner graph representation of the parity-check matrix of the code. Although this would be straightforward to implement using a standard LDPC block code decoder, it would be wasteful of resources, since the resulting (very large) block decoder would not be taking advantage of two important aspects of the convolutional structure: namely, that decoding can be done continuously without waiting for an entire terminated block to be received and that the distance between two variable nodes that are connected to the same check node is limited by the size of the syndrome former memory.

In order to take advantage of the convolutional nature of the parity-check matrix, a continuous sliding window message-passing iterative decoder that operates on a window of size $I\cdot\nu_{\rm s}$ variable nodes, where $I$ is the number of decoding iterations to be performed, can be implemented, similar to a Viterbi decoder with finite path memory [27]. This window size is chosen since, in a single iteration, messages from variable (or check) nodes can be passed across a span of only one constraint length. Thus, in $I$ iterations, messages can propagate only over a window of size $I$ constraint length. [See also the recent paper by Papaleo *et al.* [28], which investigates further reducing the window size for codes operating on a binary erasure channel (BEC).] Another simplification is achieved by exploiting the fact that a single decoding iteration of two variable nodes that are at least $m_{\rm s}+1$ 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 $I$ iterations by employing $I$ *independent identical processors* working on different regions of the parity-check matrix simultaneously, resulting in the parallel pipeline decoding architecture introduced in [19]. The pipeline decoder outputs a continuous stream of decoded data after an initial decoding delay of $I\cdot\nu_{\rm s}$ received symbols. The operation of this decoder on the Tanner graph of a simple time-invariant rate-$1/3$ convolutional code with $m_{\rm s} = 2$ and $\nu_{\rm s} = 9$ is illustrated in Fig. 1.[3]

Although the pipeline decoder is capable of fully parallelizing the iterations by using $I$ independent identical processors, employing a large number of hardware processors might not be desirable in some applications. In such cases, fewer processors (even one processor) can be scheduled to perform subsets of iterations, resulting in a serial looping architecture [29] with reduced throughput. This ability to balance the processor load and decoding speed dynamically is especially desirable when very large LDPC convolutional codes must be decoded with limited available on-chip memory. Further discussion on the implementation aspects of the pipeline decoder can be found in [30].

[2]Strictly speaking, the above formula for $\nu_{\rm s}$ gives only an upper bound on the maximal width (in symbols) of the nonzero area of $\bar{\boldsymbol{H}}_{\rm conv}$, but this upper bound will be good enough for our purposes.

[3]For LDPC convolutional codes the parameter $\nu_{\rm s}$ is usually much larger than typical values of $\nu_{\rm s}$ for "classical" convolutional codes. Therefore, the value $\nu_{\rm s} = 9$ of the convolutional code shown in Fig. 1 is not typical for the codes considered in this paper.

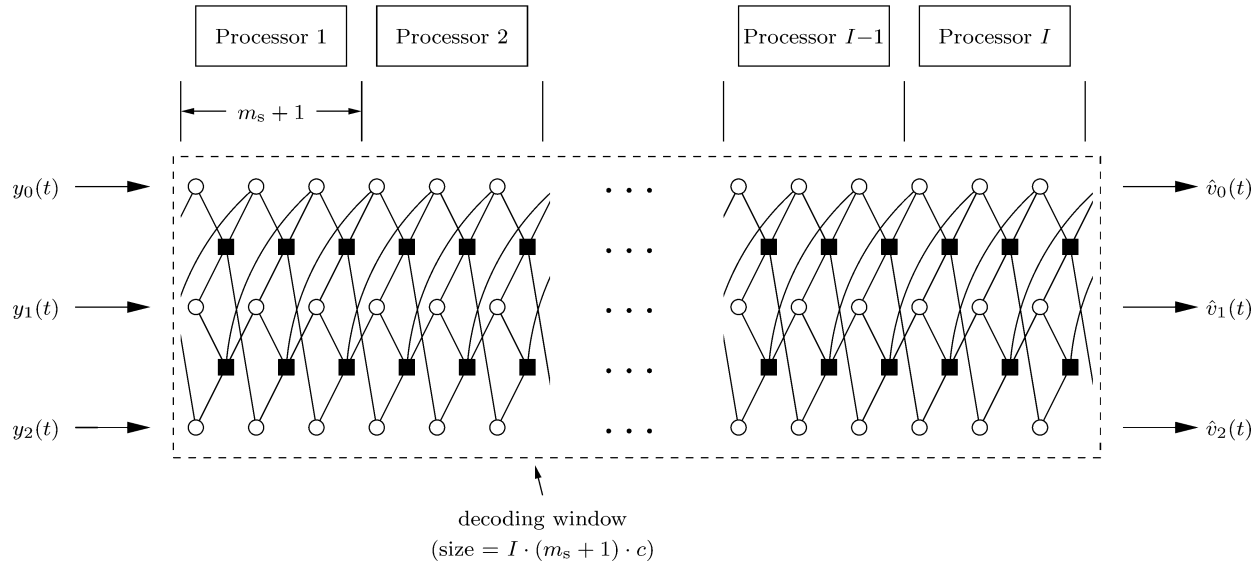


Fig. 1. Tanner graph of a rate-$1/3$ convolutional code and an illustration of pipeline decoding. Here, $y_0(t), y_1(t), y_2(t)$ denote the stream of channel output symbols, and $\hat{v}_0(t), \hat{v}_1(t), \hat{v}_2(t)$ denote the stream of decoder code bit decisions.

### C. Unwrapping Techniques due to Tanner and due to Jiménez-Feltström and Zigangirov (JFZ)

In this section, we discuss two approaches for deriving convolutional codes from block codes, in particular for deriving LDPC convolutional codes from LDPC block codes. The first technique will be the unwrapping due to Tanner and the second will be the unwrapping due to Jiménez-Feltström and Zigangirov (JFZ). In Section III, we will see, with the help of graph covers, how these two—seemingly different—unwrapping techniques are connected with each other.

The term *unwrapping*, in particular unwrapping a quasi-cyclic block code to obtain a time-invariant convolutional code, was first introduced in a paper by Tanner [17] (see also [16]). That paper describes a link between quasi-cyclic block codes and time-invariant convolutional codes and shows that the free distance of the unwrapped convolutional code cannot be smaller than the minimum distance of the underlying quasi-cyclic code. This idea was later extended in [31] and [32].

Consider the quasi-cyclic block code $\mathsf{C}_{\mathrm{QC}}^{(r)}$ defined by the polynomial parity-check matrix $\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)$ of size $m \times n$, i.e.,

$$\mathsf{C}_{\mathrm{QC}}^{(r)} = \left\{ \boldsymbol{v}(X) \in \mathbb{F}_2^{\langle r \rangle}[X]^n \mid \boldsymbol{H}_{\mathrm{QC}}^{(r)}(X) \cdot \boldsymbol{v}(X)^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}.$$

Here the polynomial operations are performed modulo $X^r - 1$. The Tanner unwrapping technique is simply based on dropping these modulo computations. More precisely, with a quasi-cyclic block code $\mathsf{C}_{\mathrm{QC}}^{(r)}$ we associate the convolutional code

$$\mathsf{C}_{\mathrm{conv}} = \left\{ \boldsymbol{v}(D) \in \mathbb{F}_2[D]^n \mid \boldsymbol{H}_{\mathrm{conv}}(D) \cdot \boldsymbol{v}(D)^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

with polynomial parity-check matrix

$$\boldsymbol{H}_{\mathrm{conv}}(D) \triangleq \boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)\Big|_{X=D}. \tag{3}$$

Here the change of indeterminate from $X$ to $D$ indicates the lack of the modulo $D^r - 1$ operations. [Note that in (3) we assume that the exponents appearing in the polynomials in $\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)$ are between 0 and $r - 1$ inclusive.]

It can easily be seen that any codeword $\boldsymbol{v}(D)$ in $\mathsf{C}_{\mathrm{conv}}$ maps to a codeword $\boldsymbol{v}(X)$ in $\mathsf{C}_{\mathrm{QC}}^{(r)}$ through

$$\boldsymbol{v}(X) \triangleq \boldsymbol{v}(D) \bmod (D^r - 1)\Big|_{D=X}$$

a process which was described in [17] as the wrapping around of a codeword in the convolutional code into a codeword in the quasi-cyclic code. The inverse process was described as unwrapping.

Having introduced the unwrapping technique due to Tanner, we move on to discuss the unwrapping technique due to JFZ [19], which is another way to unwrap a block code to obtain a convolutional code. The basic idea is best explained with the help of an example.

*Example 1:* Consider the parity-check matrix

$$\overline{\boldsymbol{H}} \triangleq \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

with size $m \times n = 5 \times 10$, of a rate-$1/2$ block code. As indicated above, we can take a pair of scissors and "cut" the parity-check matrix into two pieces, whereby the cutting pattern is such that we repeatedly move $c = 2$ units to the right and then $c - b = 1$ unit down. Having applied this "diagonal cut," we repeatedly copy and paste the two parts to obtain the bi-infinite matrix shown in Fig. 2. This new matrix can be seen as the parity-check matrix of (in general) a periodically time-varying convolutional code (here the period is $T_{\mathrm{s}} = 5$). It is worth observing that this

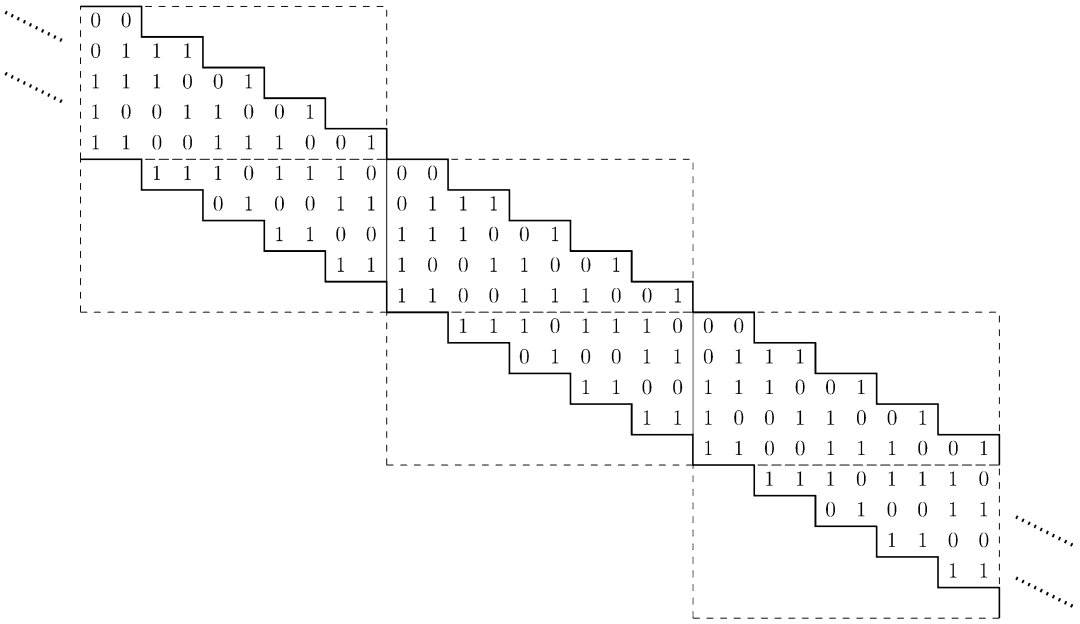

Fig. 2. Deriving a rate $R = 1/2$ periodically time-varying convolutional code with $b = 1, c = 2, m_{\mathrm{s}} = 4, \nu_{\mathrm{s}} = 10$, and $T_{\mathrm{s}} = 5$ from a rate-$1/2$ block code of length 10.

new matrix has the same row and column weights as the matrix that we started with.[4] □

This example can be formalized easily. Namely, starting with an $m \times n$ parity-check matrix $\bar{\boldsymbol{H}}$ of some block code, let $\eta \triangleq \gcd(m, n)$. Then, the "diagonal cut" is performed by alternately moving $c = n/\eta$ units to the right and then $c - b \triangleq m/\eta$ units down (i.e., $b = ((n - m)/\eta)$. The resulting convolutional code has rate $R = b/c$, syndrome former memory $m_{\mathrm{s}} = \eta - 1$, constraint length $\nu_{\mathrm{s}} = (m_{\mathrm{s}} + 1) \cdot c = \eta \cdot c = n$, and period $T_{\mathrm{s}} = m_{\mathrm{s}} + 1 = \eta$.

Analogous to the comment at the end of Section II-A, it is also possible to cut the matrix $\bar{\boldsymbol{H}}$ in larger step sizes, e.g., moving $c' = \ell \cdot c$ units to the right and $c' - b' = \ell \cdot (c - b)$ units down, for any integer $\ell$ that divides $T_{\mathrm{s}} = \eta$, thereby obtaining a periodically time-varying convolutional code with rate $R' = b'/c' = b/c$, syndrome former memory ${m_{\mathrm{s}}}' = (\eta/\ell) - 1$, constraint length ${\nu_{\mathrm{s}}}' = ({m_{\mathrm{s}}}' + 1) \cdot c' = \eta \cdot c = n$, and period ${T_{\mathrm{s}}}' = {m_{\mathrm{s}}}' + 1 = \eta/\ell$. (See also the discussion in Section IV-B.)

In the rest of this paper, the term "JFZ unwrapping technique" will also stand for the following generalization of the above procedure. Namely, starting with a length-$n$ block code $\bar{\mathsf{C}}$ defined by some size-$m \times n$ parity-check matrix $\bar{\boldsymbol{H}}$, i.e.,

$$\bar{\mathsf{C}} = \left\{ \bar{\boldsymbol{v}} \in \mathbb{F}_2^n \mid \bar{\boldsymbol{H}} \cdot \bar{\boldsymbol{v}}^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

we write $\bar{\boldsymbol{H}}$ as the sum $\bar{\boldsymbol{H}} = \sum_{\ell \in \mathcal{L}} \boldsymbol{H}_\ell$ (in $\mathbb{Z}$) of a collection of matrices $\{\boldsymbol{H}_\ell\}_{\ell \in \mathcal{L}}$. The convolutional code $\bar{\mathsf{C}}_{\mathrm{conv}}$ is then defined to be

$$\bar{\mathsf{C}}_{\mathrm{conv}} \triangleq \left\{ \bar{\boldsymbol{v}} \in \mathbb{F}_2^{\infty} \mid \bar{\boldsymbol{H}}_{\mathrm{conv}} \cdot \bar{\boldsymbol{v}}^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\} \tag{4}$$

[4]In practice, the codewords start at some time, so the convolutional parity-check matrix has effectively the semi-infinite form of (1), and the row weights of the first $\nu_{\mathrm{s}} - 1$ rows are reduced.

where

$$\bar{\boldsymbol{H}}_{\mathrm{conv}} \triangleq \begin{bmatrix} \boldsymbol{H}_0 & & & & & \\ \boldsymbol{H}_1 & \boldsymbol{H}_0 & & & & \\ \vdots & \vdots & \ddots & & & \\ \boldsymbol{H}_{|\mathcal{L}|-1} & \boldsymbol{H}_{|\mathcal{L}|-2} & \cdots & \boldsymbol{H}_0 & & \\ & \boldsymbol{H}_{|\mathcal{L}|-1} & \boldsymbol{H}_{|\mathcal{L}|-2} & \cdots & \boldsymbol{H}_0 & \\ & & \ddots & \ddots & \ddots & \ddots \end{bmatrix}.$$

Referring to the notation introduced in Section II-A, the matrix $\bar{\boldsymbol{H}}_{\mathrm{conv}}$ is the parity-check matrix of a time-invariant convolutional code. However, depending on the decomposition of $\bar{\boldsymbol{H}}$ and the internal structure of the terms in that decomposition, the matrix $\bar{\boldsymbol{H}}_{\mathrm{conv}}$ can also be (and very often is) viewed as the parity-check matrix of a time-varying convolutional code with nontrivial period $T_{\mathrm{s}}$.

In order to illustrate the generalization of the JFZ unwrapping technique that we have introduced in the last paragraph, observe that decomposing $\bar{\boldsymbol{H}}$ from Example 1 as $\bar{\boldsymbol{H}} = \boldsymbol{H}_0 + \boldsymbol{H}_1$ (in $\mathbb{Z}$) with

$$\boldsymbol{H}_0 \triangleq \left[\begin{array}{cccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

$$\boldsymbol{H}_1 \triangleq \left[\begin{array}{cccccccccc} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

yields a convolutional code with parity-check matrix $\bar{\boldsymbol{H}}_{\text{conv}}$ whose bi-infinite version equals the matrix shown in Fig. 2.

## III. Tanner Graphs From Graph Covers

Having formally introduced LDPC convolutional codes in the previous section, we now turn our attention to the main tool of this paper, namely graph covers.

*Definition 2 (see, e.g., [33]):* A *cover* of a graph $\mathsf{G}$ with vertex set $\mathcal{W}$ and edge set $\mathcal{E}$ is a graph $\tilde{\mathsf{G}}$ with vertex set $\tilde{\mathcal{W}}$ and edge set $\tilde{\mathcal{E}}$, along with a surjection $\pi : \tilde{\mathcal{W}} \to \mathcal{W}$ which is a graph homomorphism (i.e., $\pi$ takes adjacent vertices of $\tilde{\mathsf{G}}$ to adjacent vertices of $\mathsf{G}$) such that for each vertex $w \in \mathcal{W}$ and each $\tilde{w} \in \pi^{-1}(w)$, the neighborhood $\partial(\tilde{w})$ of $\tilde{w}$ is mapped bijectively to $\partial(w)$. A cover is called an $M$-*cover*, where $M$ is a positive integer, if $|\pi^{-1}(w)| = M$ for every vertex $w$ in $\mathcal{W}$.[5] □

These graph covers will be used for the construction of new Tanner graphs from old Tanner graphs, in particular for the construction of Tanner graphs that represent LDPC convolutional codes.

More specifically, this section starts by discussing two simple methods to specify a graph cover, which will be called graph-cover construction 1 (**GCC1**) and graph-cover construction 2 (**GCC2**). Although they yield isomorphic Tanner graphs, and therefore equivalent codes, it is convenient to have both methods at hand.[6] As we will see, interesting classes of Tanner graphs can be obtained by repeatedly applying these graph-cover constructions, by mixing them, and by suitably shortening the resulting codes. Moreover, these two graph-cover constructions will allow us to exhibit a connection between the Tanner and the JFZ unwrapping techniques.

### A. Graph-Cover Constructions

Let $\boldsymbol{A}$ be an $m_{\boldsymbol{A}} \times n_{\boldsymbol{A}}$ matrix over $\mathbb{Z}_{\geqslant 0}$. With such a matrix we can associate a Tanner graph $\mathsf{T}(\boldsymbol{A})$, where we draw $n_{\boldsymbol{A}}$ variable nodes, $m_{\boldsymbol{A}}$ check nodes, and where there are $[\boldsymbol{A}]_{j,i}$ edges from the $i$th variable node to the $j$th check node.[7] Given the role that the matrix $\boldsymbol{A}$ will play subsequently, we follow [21] and call the matrix $\boldsymbol{A}$ a proto-matrix and the corresponding graph $\mathsf{T}(\boldsymbol{A})$ a proto-graph.

The next definition introduces **GCC1** and **GCC2**, two ways to specify graph covers that will be used throughout the rest of the paper.[8]

*Definition 3:* For some positive integers $m_{\boldsymbol{A}}$ and $n_{\boldsymbol{A}}$, let $\boldsymbol{A} \in \mathbb{Z}_{\geqslant 0}{}^{m_{\boldsymbol{A}} \times n_{\boldsymbol{A}}}$ be a proto-matrix. We also introduce the following objects.

- For some finite set $\mathcal{L}$, let $\{\boldsymbol{A}_\ell\}_{\ell \in \mathcal{L}}$ be a collection of matrices such that $\boldsymbol{A}_\ell \in \mathbb{Z}_{\geqslant 0}{}^{m_{\boldsymbol{A}} \times n_{\boldsymbol{A}}}$, $\ell \in \mathcal{L}$, and such that $\boldsymbol{A} = \sum_{\ell \in \mathcal{L}} \boldsymbol{A}_\ell$ (in $\mathbb{Z}$).
- For some positive integer $r$, let $\{\boldsymbol{P}_\ell\}_{\ell \in \mathcal{L}}$ be a collection of size-$r \times r$ permutation matrices. For example, for every $\ell \in \mathcal{L}$, the matrix $\boldsymbol{P}_\ell$ is such that it contains one "1" per column, one "1" per row, and "0"s otherwise.

Based on the collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell \in \mathcal{L}}$ and the collection of matrices $\{\boldsymbol{P}_\ell\}_{\ell \in \mathcal{L}}$, there are two common ways of defining a graph cover of the Tanner graph $\mathsf{T}(\boldsymbol{A})$. (In the following expressions, $\boldsymbol{I}$ is the identity matrix of size $r \times r$.)

- Graph-cover construction 1 (**GCC1**). Consider the intermediary matrix
$$\boldsymbol{B}' \triangleq \boldsymbol{A} \otimes \boldsymbol{I} = \sum_{\ell \in \mathcal{L}} (\boldsymbol{A}_\ell \otimes \boldsymbol{I})$$
whose Tanner graph $\mathsf{T}(\boldsymbol{B}')$ consists of $r$ disconnected copies of $\mathsf{T}(\boldsymbol{A})$. This is an $r$-fold cover of $\mathsf{T}(\boldsymbol{A})$, albeit a rather trivial one. In order to obtain an interesting $r$-fold graph cover of $\boldsymbol{A}$, for each $\ell \in \mathcal{L}$, we replace $\boldsymbol{A}_\ell \otimes \boldsymbol{I}$ by $\boldsymbol{A}_\ell \otimes \boldsymbol{P}_\ell$, i.e., we define
$$\boldsymbol{B} \triangleq \sum_{\ell \in \mathcal{L}} (\boldsymbol{A}_\ell \otimes \boldsymbol{P}_\ell).$$
- Graph-cover construction 2 (**GCC2**) Consider the intermediary matrix
$$\bar{\boldsymbol{B}}' \triangleq \boldsymbol{I} \otimes \boldsymbol{A} = \sum_{\ell \in \mathcal{L}} (\boldsymbol{I} \otimes \boldsymbol{A}_\ell)$$
whose Tanner graph $\mathsf{T}(\bar{\boldsymbol{B}}')$ consists of $r$ disconnected copies of $\mathsf{T}(\boldsymbol{A})$. This is an $r$-fold cover of $\mathsf{T}(\boldsymbol{A})$, albeit a rather trivial one. In order to obtain an interesting $r$-fold graph cover of $\boldsymbol{A}$, for each $\ell \in \mathcal{L}$, we replace $\boldsymbol{I} \otimes \boldsymbol{A}_\ell$ by $\boldsymbol{P}_\ell \otimes \boldsymbol{A}_\ell$, i.e., we define
$$\bar{\boldsymbol{B}} \triangleq \sum_{\ell \in \mathcal{L}} (\boldsymbol{P}_\ell \otimes \boldsymbol{A}_\ell).$$

If all the matrices $\{\boldsymbol{P}_\ell\}_{\ell \in \mathcal{L}}$ are circulant matrices, then the graph covers $\mathsf{T}(\boldsymbol{B})$ and $\mathsf{T}(\bar{\boldsymbol{B}})$ will be called cyclic covers of $\mathsf{T}(\boldsymbol{A})$. □

One can verify that the two graph-cover constructions in Definition 3 are such that the matrix $\boldsymbol{B}$, after a suitable reordering of the rows and columns, equals the matrix $\bar{\boldsymbol{B}}$.[9] This implies that $\mathsf{T}(\boldsymbol{B})$ and $\mathsf{T}(\bar{\boldsymbol{B}})$ are isomorphic graphs; nevertheless, it is helpful to define both types of constructions.

### B. Graph-Cover Construction Special Cases

The following examples will help us to better understand how **GCC1** and **GCC2** can be used to obtain interesting classes of Tanner graphs, and, in particular, how the resulting graph-cover constructions can be visualized graphically. Although these examples are very concrete, they are written such that they can be easily generalized.

*Example 4 (Cyclic Cover):* Consider the proto-matrix
$$\boldsymbol{A} \triangleq \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5}$$

[5]The number $M$ is also known as the degree of the cover. (Not to be confused with the degree of a vertex.)

[6]For a formal definition of code equivalence, see, for example, [34].

[7]Note that we use a generalized notion of Tanner graphs, where parallel edges are allowed and are reflected by corresponding integer entries in the associated matrix.

[8]We leave it as an exercise for the reader to show that the graphs constructed in **GCC1** and **GCC2** are indeed two instances of the graph cover definition in Definition 2.

[9]Indeed, a possible approach to show this is to use the fact that $\boldsymbol{A}_\ell \otimes \boldsymbol{P}_\ell$ and $\boldsymbol{P}_\ell \otimes \boldsymbol{A}_\ell$ are permutation equivalent, i.e., there is a pair of permutation matrices $(\boldsymbol{Q}, \boldsymbol{Q}')$ such that $\boldsymbol{A}_\ell \otimes \boldsymbol{P}_\ell = \boldsymbol{Q} \cdot (\boldsymbol{P}_\ell \otimes \boldsymbol{A}_\ell) \cdot \boldsymbol{Q}'$. Of course, for this to work, the pair $(\boldsymbol{Q}, \boldsymbol{Q}')$ must be independent of $\ell \in \mathcal{L}$, i.e., dependent only on the size of the matrices $\{\boldsymbol{A}_\ell\}_{\ell \in \mathcal{L}}$ and $\{\boldsymbol{P}_\ell\}_{\ell \in \mathcal{L}}$. Such a $(\boldsymbol{Q}, \boldsymbol{Q}')$ pair can easily be found.

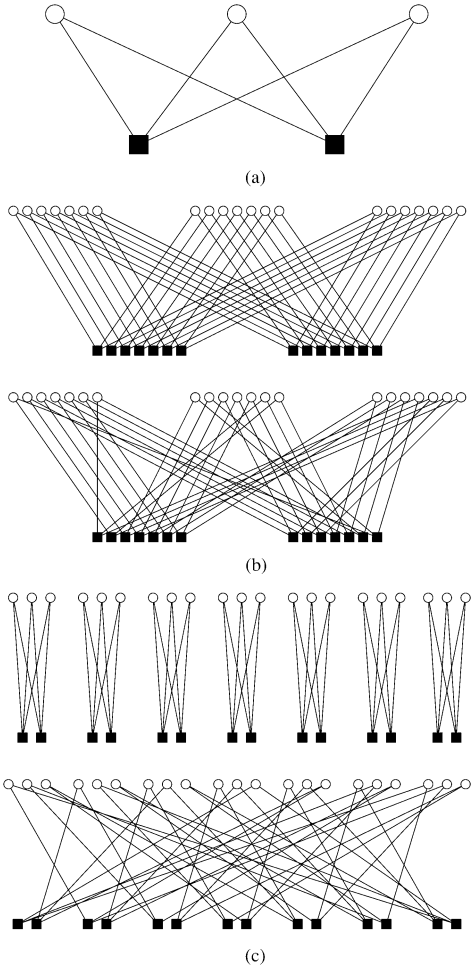


Fig. 3. Proto-graph and graph-covers for the graph-cover constructions discussed in Example 4. (Compare with the corresponding graphs in Fig. 4.) (a) Proto-graph $\mathsf{T}(\boldsymbol{A})$. (b) **GCC1** based on $\mathsf{T}(\boldsymbol{A})$. Top: $\mathsf{T}(\boldsymbol{B}')$. Bottom: $\mathsf{T}(\boldsymbol{B})$. (c) **GCC2** based on $\mathsf{T}(\boldsymbol{A})$. Top: $\mathsf{T}(\bar{\boldsymbol{B}}')$. Bottom: $\mathsf{T}(\bar{\boldsymbol{B}})$.

with $m_{\boldsymbol{A}} = 2$ and $n_{\boldsymbol{A}} = 3$, and whose Tanner graph $\mathsf{T}(\boldsymbol{A})$ is shown in Fig. 3(a). Let $\mathcal{L} \triangleq \{0,1\} \times \{0,1,2\}$, and let the collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$ be given by $\{\boldsymbol{A}_{j,i}\}_{j,i}$, where for each $j = 0,\ldots,m_{\boldsymbol{A}}-1$ and each $i = 0,\ldots,n_{\boldsymbol{A}}-1$ the matrix $\boldsymbol{A}_{j,i} \in \mathbb{Z}_{\geqslant 0}^{m_{\boldsymbol{A}}\times n_{\boldsymbol{A}}}$ is defined as follows:

$$[\boldsymbol{A}_{j,i}]_{j',i'} \triangleq \begin{cases} [\boldsymbol{A}]_{j',i'}, & \text{if } (j',i') = (j,i) \\ 0, & \text{otherwise.} \end{cases}$$

Moreover, let $r \triangleq 7$, and let the collection of matrices $\{\boldsymbol{P}_\ell\}_{\ell\in\mathcal{L}}$ be given by $\{\boldsymbol{P}_{j,i}\}_{j,i}$, where $\boldsymbol{P}_{0,0} \triangleq \boldsymbol{I}_1, \boldsymbol{P}_{0,1} \triangleq \boldsymbol{I}_2, \boldsymbol{P}_{0,2} \triangleq \boldsymbol{I}_4, \boldsymbol{P}_{1,0} \triangleq \boldsymbol{I}_6, \boldsymbol{P}_{1,1} \triangleq \boldsymbol{I}_5, \boldsymbol{P}_{1,2} \triangleq \boldsymbol{I}_3$, and where $\boldsymbol{I}_s$ is an $s$ times left-shifted identity matrix of size $r \times r$.

- Using **GCC1**, we obtain the matrices

$$\boldsymbol{B}' = \boldsymbol{A} \otimes \boldsymbol{I}_0 = \begin{bmatrix} \boldsymbol{I}_0 & \boldsymbol{I}_0 & \boldsymbol{I}_0 \\ \boldsymbol{I}_0 & \boldsymbol{I}_0 & \boldsymbol{I}_0 \end{bmatrix}$$
$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{I}_1 & \boldsymbol{I}_2 & \boldsymbol{I}_4 \\ \boldsymbol{I}_6 & \boldsymbol{I}_5 & \boldsymbol{I}_3 \end{bmatrix} \tag{6}$$

  whose Tanner graphs $\mathsf{T}(\boldsymbol{B}')$ and $\mathsf{T}(\boldsymbol{B})$, respectively, are shown in Fig. 3(b).

- Using **GCC2**, we obtain the matrices

$$\bar{\boldsymbol{B}}' = \boldsymbol{I}_0 \otimes \boldsymbol{A} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{A} \end{bmatrix}$$
$$\bar{\boldsymbol{B}} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} \\ \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} \\ \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} \\ \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} \\ \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} \\ \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{A}_{1,0} \\ \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} \end{bmatrix} \tag{7}$$

  whose Tanner graphs $\mathsf{T}(\bar{\boldsymbol{B}}')$ and $\mathsf{T}(\bar{\boldsymbol{B}})$, respectively, are shown in Fig. 3(c). Note that all the block rows and all the block columns sum (in $\mathbb{Z}$) to $\boldsymbol{A}$. (This observation holds in general, not just for this example.) □

We would like to add two comments with respect to the above example.

First, instead of defining $\boldsymbol{I}_s$ to be an $s$ times *left*-shifted identity matrix of size $r \times r$, we could have defined $\boldsymbol{I}_s$ to be an $s$ times *right*-shifted identity matrix of size $r \times r$. Compared to the matrices and graphs described above, such a change in definition would yield (in general) different matrices but isomorphic graphs.

Second, we note that **GCC2** was termed the "copy-and-permute" construction by Thorpe *et al.* This terminology stems from the visual appearance of the procedure: namely, in going from Fig. 3(a) to (c)(top), we copy the graph several times, and in going from Fig. 3(c)(top) to (c)(bottom), we permute the edges of the graph, where the permutations are done within the sets of edges that have the same preimage in Fig. 3(a).

*Remark 5 (Quasi-Cyclic Codes):* Consider again the matrices that were constructed in Example 4, in particular the matrix $\boldsymbol{A}$ in (5) and its $r$-fold cover matrix $\boldsymbol{B}$ in (6). Because all matrices in the matrix collection $\{\boldsymbol{P}_\ell\}_{\ell\in\mathcal{L}}$ are circulant, $\mathsf{T}(\boldsymbol{B})$ represents a cyclic cover of $\mathsf{T}(\boldsymbol{A})$. Clearly, when seen over $\mathbb{F}_2$, the matrix $\boldsymbol{H}_{\mathrm{QC}}^{(r)} \triangleq \boldsymbol{B}$ is the parity-check matrix of a quasi-cyclic binary linear block code

$$\mathsf{C}_{\mathrm{QC}}^{(r)} = \left\{ \boldsymbol{v} \in \mathbb{F}_2^{n_{\boldsymbol{A}}\cdot r} \,\middle|\, \boldsymbol{H}_{\mathrm{QC}}^{(r)} \cdot \boldsymbol{v}^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}.$$

Using the well-known isomorphism between the addition and multiplication of circulant matrices over $\mathbb{F}_2$ and the addition and multiplication of elements of the ring $\mathbb{F}_2^{\langle r\rangle}[X]$, this code can be written equivalently as

$$\mathsf{C}_{\mathrm{QC}}^{(r)} = \left\{ \boldsymbol{v}(X) \in \mathbb{F}_2^{\langle r\rangle}[X]^{n_{\boldsymbol{A}}} \,\middle|\, \boldsymbol{H}_{\mathrm{QC}}^{(r)}(X) \cdot \boldsymbol{v}(X)^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

with

$$\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X) \triangleq \begin{bmatrix} X^1 & X^2 & X^4 \\ X^6 & X^5 & X^3 \end{bmatrix}.$$

As noted above, the graphs $\mathsf{T}(\boldsymbol{B})$ and $\mathsf{T}(\bar{\boldsymbol{B}})$ that are constructed in Definition 3 are isomorphic. Applying this observa-

tion to Example 4, the matrix $\bar{\boldsymbol{H}}_{\mathrm{QC}}^{(r)} \triangleq \bar{\boldsymbol{B}}$ with $\bar{\boldsymbol{B}}$ from (7) is therefore the parity-check matrix of a binary linear block code

$$\bar{\mathsf{C}}_{\mathrm{QC}}^{(r)} = \left\{ \bar{\boldsymbol{v}} \in \mathbb{F}_2^{r \cdot n_{\boldsymbol{A}}} \,\middle|\, \bar{\boldsymbol{H}}_{\mathrm{QC}}^{(r)} \cdot \bar{\boldsymbol{v}}^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

that is equivalent to $\mathsf{C}_{\mathrm{QC}}^{(r)}$, i.e., the codewords of $\bar{\mathsf{C}}_{\mathrm{QC}}^{(r)}$ can be obtained from the codewords of $\mathsf{C}_{\mathrm{QC}}^{(r)}$ by a suitable reordering of the codeword components. In terms of the matrices $\{\boldsymbol{A}_{j,i}\}_{j,i}$, which also appear in the matrix $\bar{\boldsymbol{B}}$ in (7), one can verify that the polynomial parity-check matrix $\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)$ can be written as $\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X) = \boldsymbol{0}X^0 + \boldsymbol{A}_{0,0}X^1 + \boldsymbol{A}_{0,1}X^2 + \boldsymbol{A}_{1,2}X^3 + \boldsymbol{A}_{0,2}X^4 + \boldsymbol{A}_{1,1}X^5 + \boldsymbol{A}_{1,0}X^6$. □

Besides defining finite graph covers, we can also define infinite graph covers, as illustrated in the following examples. These infinite graph covers will be crucial towards defining Tanner graphs of convolutional codes.

*Example 6 (Bi-Infinite Toeplitz Covers):* We continue Example 4. However, besides keeping the proto-matrix $\boldsymbol{A}$ and the collection of matrices $\{\boldsymbol{A}_{j,i}\}_{j,i}$, we consider a different collection of matrices $\{\boldsymbol{P}_{j,i}\}_{j,i}$. Namely, we set $\boldsymbol{P}_{0,0} \triangleq \boldsymbol{T}_1$, $\boldsymbol{P}_{0,1} \triangleq \boldsymbol{T}_2, \boldsymbol{P}_{0,2} \triangleq \boldsymbol{T}_4$, $\boldsymbol{P}_{1,0} \triangleq \boldsymbol{T}_6, \boldsymbol{P}_{1,1} \triangleq \boldsymbol{T}_5$, $\boldsymbol{P}_{1,2} \triangleq \boldsymbol{T}_3$. Here $\boldsymbol{T}_s$ is a bi-infinite Toeplitz matrix with zeros everywhere except for ones in the $s$th diagonal below the main diagonal, i.e., $[\boldsymbol{T}_s]_{j,i} = 1$ if $j = i + s$ and $[\boldsymbol{T}_s]_{j,i} = 0$ otherwise. For example

$$\boldsymbol{T}_1 = \begin{bmatrix} \ddots & \ddots & \ddots & \ddots & & \\ \ddots & \underline{0} & 0 & 0 & 0 & \\ \ddots & 1 & \underline{0} & 0 & 0 & \ddots \\ \ddots & 0 & 1 & \underline{0} & 0 & \ddots \\ & 0 & 0 & 1 & \underline{0} & \ddots \\ & & \ddots & \ddots & \ddots & \ddots \end{bmatrix}$$

where for clarity we have underlined the entries of the main diagonal.

- Using **GCC1**, we obtain the matrices $\boldsymbol{B}' = \boldsymbol{A} \otimes \boldsymbol{T}_0$ and

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{T}_1 & \boldsymbol{T}_2 & \boldsymbol{T}_4 \\ \boldsymbol{T}_6 & \boldsymbol{T}_5 & \boldsymbol{T}_3 \end{bmatrix}. \tag{8}$$

  The Tanner graph $\mathsf{T}(\boldsymbol{B}')$, which is depicted in Fig. 4(b)(top), is similar to the corresponding Tanner graph in Fig. 3(b)(top), but with bi-infinitely many independent components. Analogously, the Tanner graph $\mathsf{T}(\boldsymbol{B})$, which is depicted in Fig. 4(b)(bottom), is similar to the Tanner graph shown in Fig. 3(b)(bottom), but instead of cyclically wrapped edge connections, the edge connections are infinitely continued on both sides.

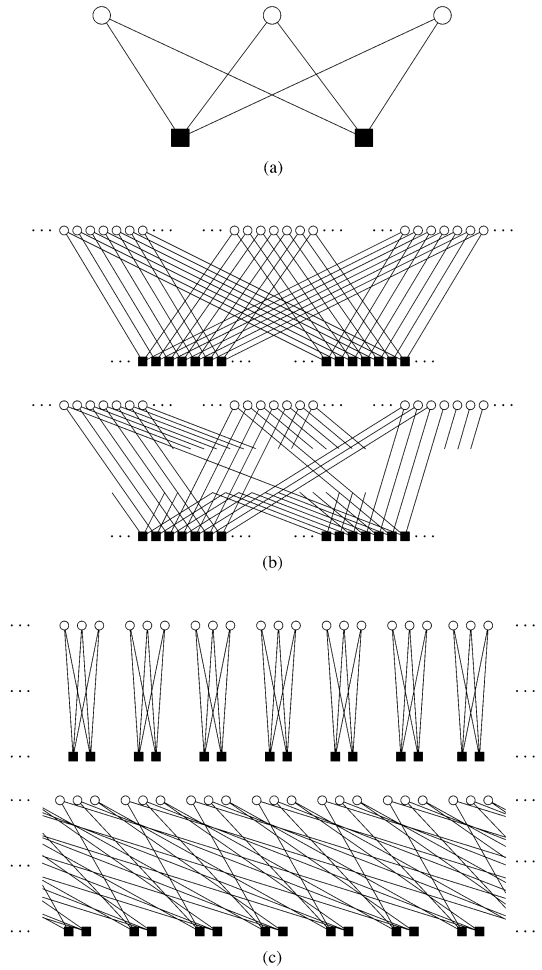


Fig. 4. Proto-graph and graph-covers for the graph-cover constructions discussed in Example 6. (Compare with the corresponding graphs in Fig. 3.) (a) Proto-graph $\mathsf{T}(\boldsymbol{A})$. (b) **GCC1** based on $\mathsf{T}(\boldsymbol{A})$. Top: $\mathsf{T}(\boldsymbol{B}')$. Bottom: $\mathsf{T}(\boldsymbol{B})$. (c) **GCC2** based on $\mathsf{T}(A)$. Top: $\mathsf{T}(\bar{\boldsymbol{B}}')$. Bottom: $\mathsf{T}(\bar{\boldsymbol{B}})$.

- Using **GCC2**, we obtain the matrices $\bar{\boldsymbol{B}}' = \boldsymbol{T}_0 \otimes \boldsymbol{A}$ and (9), shown at the bottom of the page. The Tanner graph

$$\bar{\boldsymbol{B}} = \begin{bmatrix} \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ \ddots & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \\ \ddots & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ddots \\ \ddots & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ddots \\ \ddots & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ddots \\ \ddots & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ddots \\ \ddots & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \ddots \\ \ddots & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \boldsymbol{0} & \ddots \\ & \boldsymbol{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \boldsymbol{0} & \ddots \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}. \tag{9}$$

$\mathsf{T}(\bar{\boldsymbol{B}}')$, which is depicted in Fig. 4(c)(top), is similar to the corresponding Tanner graph in Fig. 3(c)(top), but with bi-infinitely many independent components. Analogously, the Tanner graph $\mathsf{T}(\bar{\boldsymbol{B}})$, which is depicted in Fig. 4(c)(bottom), is similar to the Tanner graph shown in Fig. 3(c)(bottom), but instead of cyclically wrapped edge connections, the edge connections are infinitely continued on both sides. □

Although it is tempting to replace in Example 6 the bi-infinite Toeplitz matrices $\boldsymbol{T}_s$ (whose row and column index sets equal $\mathbb{Z}$) by semi-infinite Toeplitz matrices (whose row and column index sets equal $\mathbb{Z}_{\geqslant 0}$), note that the resulting Tanner graphs $\mathsf{T}(\boldsymbol{B})$ and $\mathsf{T}(\bar{\boldsymbol{B}})$ would then in general not be graph covers of $\mathsf{T}(\boldsymbol{A})$. This follows from the fact that semi-infinite Toeplitz matrices are not permutation matrices (except for $\boldsymbol{T}_0$), and so some vertex degrees of $\mathsf{T}(\boldsymbol{B})$ and $\mathsf{T}(\bar{\boldsymbol{B}})$ would not equal the corresponding vertex degrees in $\mathsf{T}(\boldsymbol{A})$.[10]

*Remark 7:* It turns out that the Tanner graphs in Fig. 4 are infinite graph covers of the Tanner graphs in Fig. 3. More precisely, the Tanner graphs $\mathsf{T}(\boldsymbol{B}'), \mathsf{T}(\boldsymbol{B}), \mathsf{T}(\bar{\boldsymbol{B}}'), \mathsf{T}(\bar{\boldsymbol{B}})$ in Fig. 4 are graph covers of the corresponding Tanner graphs $\mathsf{T}(\boldsymbol{B}'), \mathsf{T}(\boldsymbol{B}), \mathsf{T}(\bar{\boldsymbol{B}}'), \mathsf{T}(\bar{\boldsymbol{B}})$ in Fig. 3. For the Tanner graphs $\mathsf{T}(\boldsymbol{B}')$ in Figs. 3(b)(top) and 4(b)(top) and the Tanner graphs $\mathsf{T}(\bar{\boldsymbol{B}}')$ in Figs. 3(c)(top) and 4(c)(top), this statement is easily verified by inspection.

To verify that the Tanner graph $\mathsf{T}(\bar{\boldsymbol{B}})$ in Fig. 4(c)(bottom) is a graph cover of $\mathsf{T}(\bar{\boldsymbol{B}})$ in Fig. 3(c)(bottom), we apply **GCC2** with proto-matrix $\boldsymbol{A}$, with resulting matrix $\bar{\boldsymbol{B}}$, with the set $\mathcal{L}$, with the collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$, and with the collection of permutation matrices $\{\boldsymbol{P}_\ell\}_{\ell\in\mathcal{L}}$ as follows. Namely, we let the proto-matrix $\boldsymbol{A}$ be the matrix from (7) (there denoted by $\bar{\boldsymbol{B}}$), we let the resulting matrix $\bar{\boldsymbol{B}}$ be the matrix in (9) (there denoted by $\bar{\boldsymbol{B}}$), we define $\mathcal{L} \triangleq \{0, 1\}$, we select

$$\boldsymbol{A}_0 = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{A}_{0,0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} & \mathbf{0} \end{bmatrix} \tag{10}$$

$$\boldsymbol{A}_1 = \begin{bmatrix} \mathbf{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} & \boldsymbol{A}_{0,0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} & \boldsymbol{A}_{0,1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} & \boldsymbol{A}_{1,2} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} & \boldsymbol{A}_{0,2} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{A}_{1,0} & \boldsymbol{A}_{1,1} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{A}_{1,0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{11}$$

and we select $\boldsymbol{P}_0 = \boldsymbol{T}_0$ and $\boldsymbol{P}_1 = \boldsymbol{T}_1$, where $\boldsymbol{T}_s$ was defined in Example 6. Clearly, $\boldsymbol{A} = \boldsymbol{A}_0 + \boldsymbol{A}_1$ (in $\mathbb{Z}$).[11] With this we have

$$\begin{aligned} \bar{\boldsymbol{B}} &= \boldsymbol{P}_0 \otimes \boldsymbol{A}_0 + \boldsymbol{P}_1 \otimes \boldsymbol{A}_1 \\ &= \begin{bmatrix} \ddots & \ddots & \ddots & \ddots & & \\ \ddots & \boldsymbol{A}_0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \\ \ddots & \boldsymbol{A}_1 & \boldsymbol{A}_0 & \mathbf{0} & \mathbf{0} & \ddots \\ \ddots & \mathbf{0} & \boldsymbol{A}_1 & \boldsymbol{A}_0 & \mathbf{0} & \ddots \\ & \mathbf{0} & \mathbf{0} & \boldsymbol{A}_1 & \boldsymbol{A}_0 & \ddots \\ & & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \end{aligned}$$

and one can verify that this matrix equals the matrix in (9) (there denoted by $\bar{\boldsymbol{B}}$), which means that $\mathsf{T}(\bar{\boldsymbol{B}})$ is indeed an infinite cover of $\mathsf{T}(\boldsymbol{A})$. We remark that, interestingly, in this process we have shown how a certain **GCC2** graph cover of a proto-matrix can be written as a **GCC2** graph cover of a certain **GCC2** graph cover of that proto-matrix.

Finally, a similar argument shows that the Tanner graph $\mathsf{T}(\boldsymbol{B})$ in Fig. 4(b)(bottom) is a graph cover of the Tanner graph in Fig. 3(b)(bottom), also denoted by $\mathsf{T}(\boldsymbol{B})$. □

There are many other ways of writing a proto-matrix $\boldsymbol{A}$ as a sum of a collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$. The next example discusses two such possibilities.

*Example 8:* Consider the proto-matrix

$$\boldsymbol{A} \triangleq \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

that is shown in Fig. 5(a), and that also appeared in Example 1. Its Tanner graph $\mathsf{T}(\boldsymbol{A})$ is $(3, 6)$-regular, i.e., all variable nodes have degree 3 and all check nodes have degree 6. Let $\mathcal{L} = \{0, 1\}$, and consider the collection of matrices $\{\boldsymbol{P}_\ell\}_{\ell\in\mathcal{L}}$ with $\boldsymbol{P}_0 = \boldsymbol{T}_0$ and $\boldsymbol{P}_1 = \boldsymbol{T}_1$, where the matrices $\boldsymbol{T}_0$ and $\boldsymbol{T}_1$ are defined as in Example 6. In the following, we look at two different choices of the collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$.

- Fig. 5(c) shows a typical part of the matrix $\bar{\boldsymbol{B}}$ that is obtained when **GCC2** is used to construct a graph cover of $\boldsymbol{A}$ with the collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$ defined as shown in Fig. 5(a).
- Fig. 5(d) shows a typical part of the matrix $\bar{\boldsymbol{B}}$ when **GCC2** is used to construct a graph cover of $\boldsymbol{A}$ with the collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$ defined as shown in Fig. 5(b).

Overall, because of the choice of the collection $\{\boldsymbol{P}_\ell\}_{\ell\in\mathcal{L}}$, the support of both matrices $\bar{\boldsymbol{B}}$ possesses a banded diagonal structure. Moreover, the different choices of the collection $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$ leads to a somewhat narrower banded diagonal structure in the first case compared to the second case. □

The next example makes a crucial observation; namely, it shows that the above graph-cover constructions can be applied repeatedly to obtain additional interesting classes of Tanner graphs.

[10]As will be clear from the discussion later on, in this paper we take an approach where in a first step we construct bi-infinite Tanner graphs that are "proper" graph covers and where in a second step we obtain semi-infinite Tanner graphs by applying a "shortening" procedure to these bi-infinite Tanner graphs. Alternatively, one could also choose an approach based on "improper" graph covers. Both approaches have their advantages and disadvantages; we preferred to take the first approach.

[11]Note that a nonzero block diagonal of $\boldsymbol{A}$ would be put in $\boldsymbol{A}_0$.

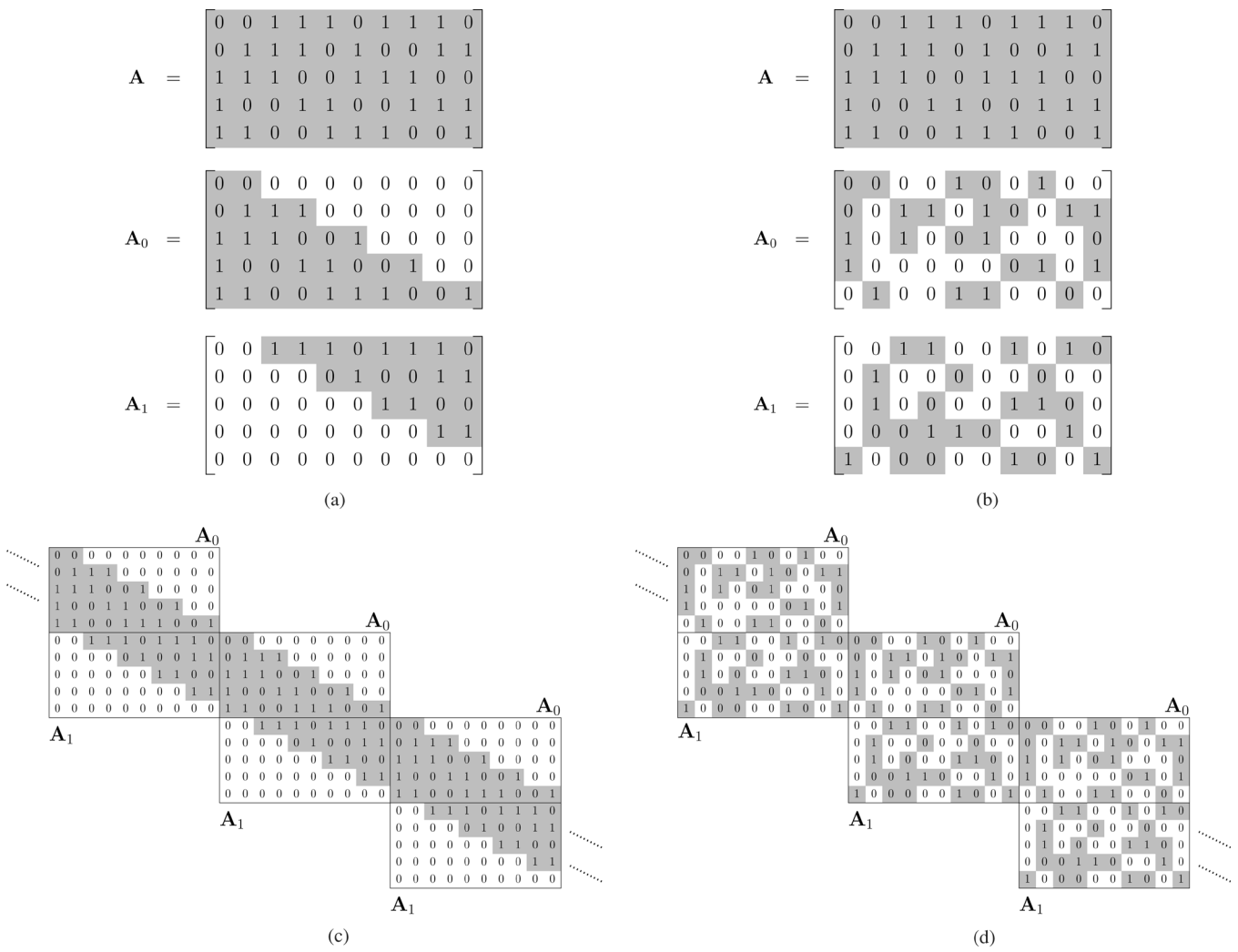


Fig. 5. Matrices appearing in Example 8. (See main text for details.) (a) First decomposition of the matrix $\boldsymbol{A}$ into the matrices $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$. (b) Second decomposition of the matrix $\boldsymbol{A}$ into the matrices $\boldsymbol{A}_0$ and $\boldsymbol{A}_1$. (c) Part of the matrix $\bar{\boldsymbol{B}}$ based on the first decomposition of $\boldsymbol{A}$. (d) Part of the matrix $\bar{\boldsymbol{B}}$ based on the second decomposition of $\boldsymbol{A}$.

*Example 9 (Iterated Graph-Cover Construction):* Starting with the proto-matrix $\boldsymbol{A}$ from Example 4, we consider two iterated graph-cover constructions. In the first case, we apply **GCC1** and then **GCC2**, and in the second case we apply **GCC2** twice.

- Consider the matrix $\boldsymbol{B}$ obtained from the matrix $\boldsymbol{A}$ using **GCC1**, like in Example 4. The resulting matrix $\boldsymbol{B}$ is shown in (6) and will be called $\boldsymbol{A}^{(1)}$ in this example, since it is considered to be a proto-matrix by itself; cf. Fig. 6(a). Based on the "cutting line" shown in Fig. 6(a), we define the matrices $\boldsymbol{A}_0^{(1)}$ and $\boldsymbol{A}_1^{(1)}$ as follows: the nonzero part of $\boldsymbol{A}_0^{(1)}$ equals the nonzero part of the lower triangular part of $\boldsymbol{A}^{(1)}$ and the nonzero part of $\boldsymbol{A}_1^{(1)}$ equals the nonzero part of the upper triangular part of $\boldsymbol{A}^{(1)}$. (Clearly, $\boldsymbol{A}^{(1)} = \boldsymbol{A}_0^{(1)} + \boldsymbol{A}_1^{(1)}$.) Applying the procedure from Example 8, Fig. 6(c) shows a typical part of the matrix $\bar{\boldsymbol{B}}^{(1)}$ that is obtained when **GCC2** is used to construct a graph cover of $\boldsymbol{A}^{(1)}$.
- Consider the graph-cover $\bar{\boldsymbol{B}}$ obtained from $\boldsymbol{A}$ using **GCC2**, like in Example 4. The resulting matrix $\bar{\boldsymbol{B}}$ is shown in (7) and will be called $\boldsymbol{A}^{(2)}$ in this example, since it is considered to be a proto-matrix by itself; cf. Fig. 6(b). Based on the "cutting line" shown in Fig. 6(b), we define the matrices $\boldsymbol{A}_0^{(2)}$ and $\boldsymbol{A}_1^{(2)}$ as follows: the nonzero part of $\boldsymbol{A}_0^{(2)}$ equals the nonzero part of the lower triangular part of $\boldsymbol{A}^{(2)}$ and the nonzero part of $\boldsymbol{A}_1^{(2)}$ equals the nonzero part of the upper triangular part of $\boldsymbol{A}^{(2)}$. (Clearly, $\boldsymbol{A}^{(2)} = \boldsymbol{A}_0^{(2)} + \boldsymbol{A}_1^{(2)}$.) Applying the procedure from Example 8, Fig. 6(d) shows a typical part of the matrix $\bar{\boldsymbol{B}}^{(2)}$ that is obtained when **GCC2** is used to construct a graph cover of $\boldsymbol{A}^{(2)}$.

We observe a large difference in the positions of the nonzero entries in $\bar{\boldsymbol{B}}^{(1)}$ and $\bar{\boldsymbol{B}}^{(2)}$.

- In the first case, the two graph-cover constructions are "incompatible" and the positions of the nonzero entries in $\bar{\boldsymbol{B}}^{(1)}$ follow a "nonsimple" or "pseudorandom" pattern. As we will see in Example 18 with the help of simulation results, such Tanner graphs can lead to time-varying LDPC convolutional codes with very good performance.
- In the second case, the two graph-cover constructions are "compatible" in the sense that $\bar{\boldsymbol{B}}^{(2)}$ can be obtained from the proto-matrix $\boldsymbol{A}$ by applying **GCC2** with suitable matrix

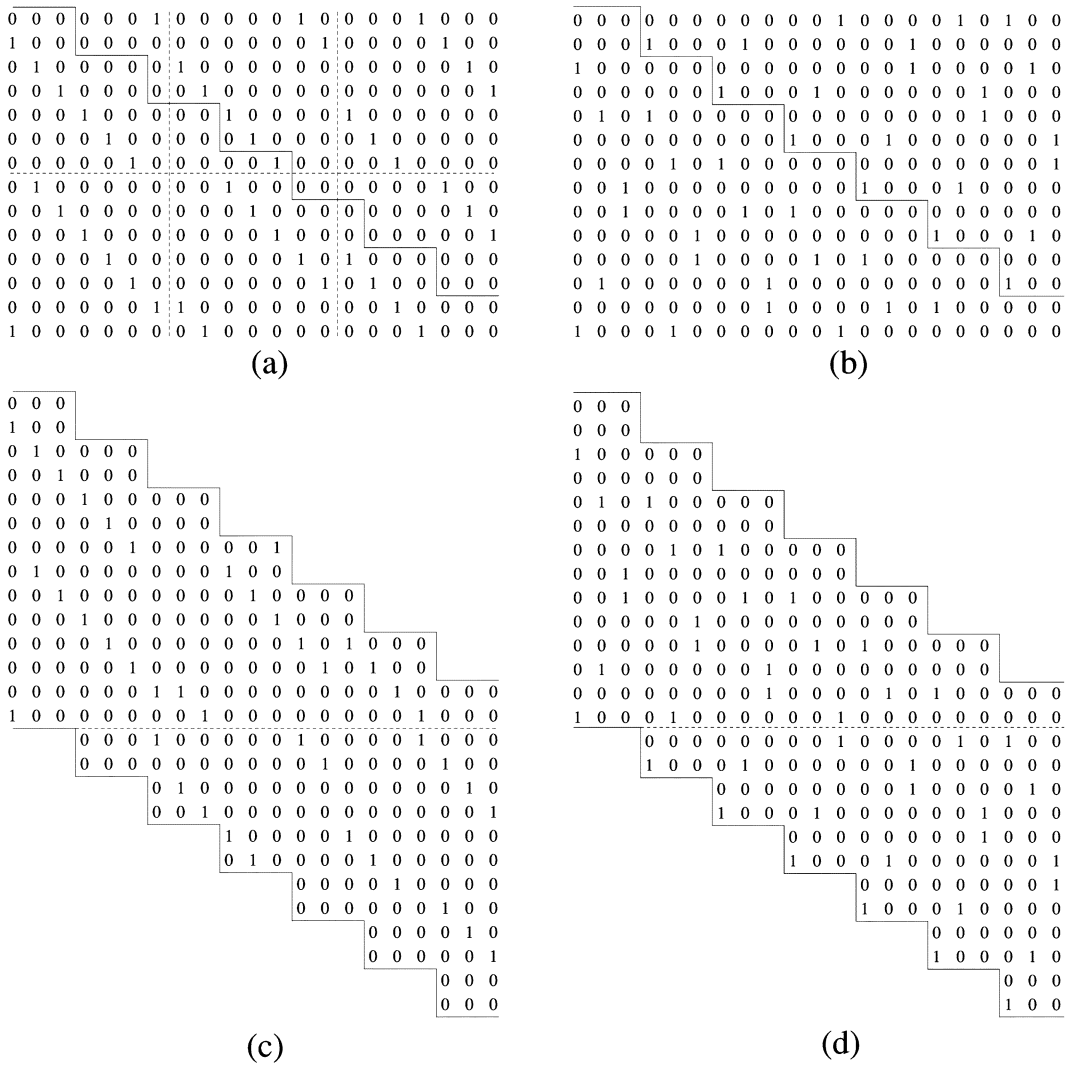


Fig. 6. Matrices appearing in Example 9. (See main text for details.) (a) Matrix $\boldsymbol{A}^{(1)}$, (b) Matrix $\boldsymbol{A}^{(2)}$, (c) Part of matrix $\bar{\boldsymbol{B}}^{(1)}$, (d) Part of matrix $\bar{\boldsymbol{B}}^{(2)}$.

collections $\{\boldsymbol{A}_\ell\}_{\ell\in\mathcal{L}}$ and $\{\boldsymbol{P}_\ell\}_{\ell\in\mathcal{L}}$. As such, the positions of the nonzero entries of $\bar{\boldsymbol{B}}^{(2)}$ follow a relatively "simple" or "nonrandom" pattern, which leads to a time-invariant LDPC convolutional code. □

The above procedure of obtaining two matrices that add up to a matrix is called "cutting a matrix." Actually, we will also use this term if there is no simple cutting line, as in the above examples, and also if the matrix is written as the sum of more than two matrices (cf. Example 1 and the paragraphs after it).

### C. Revisiting the Tanner and the JFZ Unwrapping Techniques

In Section II-C, we introduced two techniques, termed the Tanner and the JFZ unwrapping techniques, to derive convolutional codes from block codes. In this subsection we revisit these unwrapping techniques. In particular, we show how they can be cast in terms of graph covers and how the two unwrapping techniques are connected.

Because of the importance of the coding-theoretic notion of shortening [34] for this subsection, we briefly revisit this concept. Let $\boldsymbol{H}$ be a parity-check matrix that defines some length-$n$ binary code $\mathsf{C}$. We say that the length-$(n-1)$ code $\mathsf{C}'$ is obtained by shortening $\mathsf{C}$ at position $i$ if

$$\mathsf{C}' = \left\{(v_0,\ldots,v_{i-1},v_{i+1},\ldots,v_{n-1}) \in \mathbb{F}_2^{n-1} \,\middle|\, \boldsymbol{v}\in\mathsf{C}, v_i = 0\right\}.$$

In terms of parity-check matrices, a possible parity-check matrix $\boldsymbol{H}'$ of $\mathsf{C}'$ is obtained by deleting the $i$th column of $\boldsymbol{H}$. In terms of Tanner graphs, this means that the Tanner graph $\mathsf{T}(\boldsymbol{H}')$ is obtained from $\mathsf{T}(\boldsymbol{H})$ by removing the $i$th variable node, along with its incident edges. In the following, we will also use the term "shortening" to denote this graph modification procedure.

Now, to explain the Tanner unwrapping technique in terms of graph covers, consider the quasi-cyclic block code $\mathsf{C}_{\mathrm{QC}}^{(r)}$ defined by the polynomial parity-check matrix $\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)$ of size $m_{\boldsymbol{A}} \times n_{\boldsymbol{A}}$, i.e.,

$$\mathsf{C}_{\mathrm{QC}}^{(r)} = \left\{ \boldsymbol{v}(X) \in \mathbb{F}_2^{\langle r\rangle}[X]^{n_{\boldsymbol{A}}} \,\middle|\, \boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)\cdot\boldsymbol{v}(X)^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

where the polynomial operations are performed modulo $X^r - 1$ (see also Remark 5). As already mentioned in Section II-C, the Tanner unwrapping technique is simply based on dropping these

modulo computations. More precisely, with a quasi-cyclic block code $\mathsf{C}_{\mathrm{QC}}^{(r)}$, we associate the convolutional code

$$\mathsf{C}_{\mathrm{conv}} = \left\{ \boldsymbol{v}(D) \in \mathbb{F}_2[D]^{n_{\boldsymbol{A}}} \middle| \boldsymbol{H}_{\mathrm{conv}}(D) \cdot \boldsymbol{v}(D)^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

with polynomial parity-check matrix

$$\boldsymbol{H}_{\mathrm{conv}}(D) \triangleq \boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)\Big|_{X=D}.$$

Again, the change of indeterminate from $X$ to $D$ indicates the lack of modulo $D^r - 1$ operations.

In the following we will give, with the help of an example, two interpretations of the Tanner unwrapping technique in terms of graph covers.

*Example 10:* Unwrapping the quasi-cyclic block code $\mathsf{C}_{\mathrm{QC}}^{(r)}$ that was considered in Remark 5, we obtain a rate-$1/3$ time-invariant convolutional code

$$\mathsf{C}_{\mathrm{conv}} = \left\{ \boldsymbol{v}(D) \in \mathbb{F}_2[D]^{n_{\boldsymbol{A}}} \middle| \boldsymbol{H}_{\mathrm{conv}}(D) \cdot \boldsymbol{v}(D)^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\}$$

with polynomial parity-check matrix

$$\boldsymbol{H}_{\mathrm{conv}}(D) \triangleq \begin{bmatrix} D^1 & D^2 & D^4 \\ D^6 & D^5 & D^3 \end{bmatrix}.$$

Consider now the infinite graph covers that were constructed in Example 6 using **GCC1**, in particular $\mathsf{T}(\boldsymbol{B})$. Let $\mathsf{C}(\mathsf{T}(\boldsymbol{B}))$ be the set of codewords defined by the Tanner graph $\mathsf{T}(\boldsymbol{B})$. Then, the convolutional code $\mathsf{C}_{\mathrm{conv}}$ is a shortened version of $\mathsf{C}(\mathsf{T}(\boldsymbol{B}))$ where all codeword bits corresponding to negative time indices have been shortened. Therefore, the Tanner graph of $\mathsf{C}_{\mathrm{conv}}$ is given by the Tanner graph in Fig. 4(b)(bottom), where all bit nodes with negative time indices, along with their incident edges, are removed. Clearly, this bit-node and edge removal process implies decreasing the degrees of some check nodes. In fact, some check nodes become obsolete, because their degree is decreased to zero. □

Therefore, one interpretation of the Tanner unwrapping technique in terms of graph covers is that the Tanner graph of the convolutional code is obtained by taking a suitable graph cover of the same proto-graph that was used to construct the quasi-cyclic LDPC code, along with some suitable shortening.

*Example 11:* We continue Remark 5 and Example 10. Clearly, in the same way as the block code $\bar{\mathsf{C}}_{\mathrm{QC}}^{(r)}$ is equivalent to the block code $\mathsf{C}_{\mathrm{QC}}^{(r)}$, we can define a code $\bar{\mathsf{C}}_{\mathrm{conv}}$ (with parity-check matrix $\bar{\boldsymbol{H}}_{\mathrm{conv}}$) that is equivalent to $\mathsf{C}_{\mathrm{conv}}$. The observations in Remark 7 and Example 10 can then be used to show that the Tanner graph of $\bar{\boldsymbol{H}}_{\mathrm{conv}}$ equals a graph cover of the Tanner graph $\bar{\boldsymbol{H}}_{\mathrm{QC}}^{(r)}$, along with some suitable shortening. □

Therefore, the second interpretation of the Tanner unwrapping in terms of graph covers is that the Tanner graph of the convolutional code is obtained by taking a suitable graph cover of the Tanner graph of the quasi-cyclic code, along with some suitable shortening.

Now turning our attention to the JFZ unwrapping technique, recall from Section II-C that this method is based on writing a parity-check matrix $\bar{\boldsymbol{H}}$ of some block code $\bar{\mathsf{C}}$ as the sum $\bar{\boldsymbol{H}} = \sum_{\ell \in \mathcal{L}} \boldsymbol{H}_\ell$ (in $\mathbb{Z}$) of a collection of matrices $\{\boldsymbol{H}_\ell\}_{\ell \in \mathcal{L}}$. The convolutional code is then defined to be

$$\bar{\mathsf{C}}_{\mathrm{conv}} \triangleq \left\{ \bar{\boldsymbol{v}} \in \mathbb{F}_2^{\infty} \middle| \bar{\boldsymbol{H}}_{\mathrm{conv}} \cdot \bar{\boldsymbol{v}}^{\mathsf{T}} = \boldsymbol{0}^{\mathsf{T}} \right\} \tag{12}$$

where

$$\bar{\boldsymbol{H}}_{\mathrm{conv}} \triangleq \begin{bmatrix} \boldsymbol{H}_0 & & & & & \\ \boldsymbol{H}_1 & \boldsymbol{H}_0 & & & & \\ \vdots & \vdots & \ddots & & & \\ \boldsymbol{H}_{|\mathcal{L}|-1} & \boldsymbol{H}_{|\mathcal{L}|-2} & \cdots & \boldsymbol{H}_0 & & \\ & \boldsymbol{H}_{|\mathcal{L}|-1} & \boldsymbol{H}_{|\mathcal{L}|-2} & \cdots & \boldsymbol{H}_0 & \\ & & \ddots & \ddots & \ddots & \ddots \end{bmatrix}. \tag{13}$$

With the help of an example, we now explain how the JFZ unwrapping technique can be cast in terms of graph-covers.

*Example 12:* Consider the infinite graph covers that were constructed using **GCC2** in Example 6, in particular $\mathsf{T}(\bar{\boldsymbol{B}})$. Let $\mathsf{C}(\mathsf{T}(\bar{\boldsymbol{B}}))$ be the set of valid assignments to the Tanner graph $\mathsf{T}(\bar{\boldsymbol{B}})$. Moreover, let $\bar{\boldsymbol{H}} \triangleq \boldsymbol{H}_0 + \boldsymbol{H}_1 + \cdots + \boldsymbol{H}_6 \triangleq \boldsymbol{0} + \boldsymbol{A}_{0,0} + \boldsymbol{A}_{0,1} + \boldsymbol{A}_{1,2} + \boldsymbol{A}_{0,2} + \boldsymbol{A}_{1,1} + \boldsymbol{A}_{1,0}$, and let $\bar{\mathsf{C}}_{\mathrm{conv}}$ be defined as in (12). Then, the code $\bar{\mathsf{C}}_{\mathrm{conv}}$ is a shortened version of $\mathsf{C}(\mathsf{T}(\bar{\boldsymbol{B}}))$, where all codeword bits corresponding to negative time indices have been shortened. Therefore, the Tanner graph of $\bar{\mathsf{C}}_{\mathrm{conv}}$ is given by the Tanner graph in Fig. 4(c)(bottom), where all the bit nodes with negative time indices are shortened. □

In order to connect the unwrapping techniques due to Tanner and due to JFZ, we show now, with the help of an example, that in fact the unwrapping technique due to Tanner can be seen as a *special case* of the unwrapping technique due to JFZ.[12]

*Example 13:* Consider the quasi-cyclic block code defined by the parity-check matrix $\bar{\boldsymbol{H}}_{\mathrm{QC}}^{(r)} \triangleq \bar{\boldsymbol{B}}$, where $\bar{\boldsymbol{B}}$ was defined in (7). Applying the JFZ unwrapping technique with the matrix decomposition $\bar{\boldsymbol{H}}_{\mathrm{QC}}^{(r)} = \boldsymbol{A}_0 + \boldsymbol{A}_1$ (in $\mathbb{Z}$), with $\boldsymbol{A}_0$ defined in (10) and $\boldsymbol{A}_1$ defined in (11), $\bar{\boldsymbol{H}}_{\mathrm{conv}}$ turns out to equal a submatrix of $\bar{\boldsymbol{B}}$ in (9), namely the submatrix of $\bar{\boldsymbol{B}}$ where the row and column index set are equal to $\mathbb{Z}_{\geq 0}$. However, the code defined by $\bar{\boldsymbol{H}}_{\mathrm{conv}}$ is equivalent to the code defined by the Tanner unwrapping technique applied to the quasi-cyclic code defined by $\bar{\boldsymbol{H}}_{\mathrm{QC}}^{(r)}$. □

Therefore, the unwrapping technique due to JFZ is more general. In fact, whereas the Tanner unwrapping technique leads to *time-invariant* convolutional codes, the unwrapping technique due to JFZ can, depending on the parity-check matrix decomposition and the internal structure of the terms in the decomposition, lead to *time-varying* convolutional codes with nontrivial period.[13]

Despite the fact that the unwrapping technique due to Tanner is a special case of the unwrapping technique due to JFZ, it is nevertheless helpful to have both unwrapping techniques at

[12]We leave it as an exercise for the reader to show the validity of this connection beyond this specific example.

[13]Of course, if the underlying quasi-cyclic block code is suitably chosen, then also the Tanner unwrapping technique can yield a time-varying convolutional code; however, we do not consider this option here.

hand, because sometimes one framework can be more convenient than the other. We will use both perspectives in the next section.

We conclude this section with the following remarks.

- Although most of the examples in this section have regular bit node degree 2 and regular check node degree 3, there is nothing special about this choice of bit and check node degrees; any other choice would work equally well.
- Although all polynomial parity-check matrices that appear in this section contain only monomials, this is not required, i.e., the developments in this section work equally well for polynomial parity-check matrices containing the zero polynomial, monomials, binomials, trinomials, and so on.
- It can easily be verified that if the matrix $\boldsymbol{A}$ in Definition 3 contains only zeros and ones, then the graph covers constructed in **GCC1** and **GCC2** never have parallel edges. In particular, if $\boldsymbol{A}$ is the parity-check matrix of a block code (like in most examples in this paper), then the constructed graph covers never have parallel edges. However, if $\boldsymbol{A}$ contains entries that are larger than one, then there is the potential for the constructed graph covers to have parallel edges; if parallel edges really appear depends then critically on the choice of the decomposition $\boldsymbol{A} = \sum_{\ell \in \mathcal{L}} \boldsymbol{A}_\ell$ (in $\mathbb{Z}$) and the choice of the permutation matrices $\{\boldsymbol{P}_\ell\}_{\ell \in \mathcal{L}}$. An example of such a case is the Tanner graph construction in Section V-C, where $\boldsymbol{A} \triangleq [3\ 3]$ and where $\{\boldsymbol{A}_\ell\}_{\ell \in \mathcal{L}}$ and $\{\boldsymbol{P}_\ell\}_{\ell \in \mathcal{L}}$ are chosen such that parallel edges are avoided in the constructed graph cover. We note that in the case of iterated graph-cover constructions it can make sense to have parallel edges in the intermediate graph covers. However, in the last graph-cover construction stage, parallel edges are usually avoided, because parallel edges in Tanner graphs typically lead to a weakening of the code and/or of the message-passing iterative decoder.

## IV. Graph-Cover-Based Constructions of LDPC Convolutional Codes

Although the graph-cover constructions and unwrapping techniques that were discussed in Sections II and III are mathematically quite straightforward, it is important to understand how they can be applied to obtain LDPC convolutional codes with good performance and attractive encoder and decoder architectures. To that end, this section explores a variety of code design options and comments on some practical issues. It also proposes a new "random" unwrapping technique which leads to convolutional codes whose performance compares favorably to other codes with the same parameters. Of course, other variations than the ones presented here are possible, in particular, by suitably combining some of the example constructions.

The simulation results for the codes in this section plot the decoded bit error rate (BER) versus the signal-to-noise ratio (SNR) $E_\mathrm{b}/N_0$ and were obtained by assuming binary phase-shift keying (BPSK) modulation and an additive white Gaussian noise channel (AWGNC). All decoders were based on the sum-product algorithm [35] and were allowed a maximum of 100 iterations, with the block code decoders employing a syndrome-check based stopping rule. For comparing the performance of unwrapped convolutional codes with their underlying block codes we will use the following metric.

*Definition 14:* For a convolutional code constructed from an underlying block code, we define its "convolutional gain" to be the difference in SNR required to achieve a particular BER with the convolutional code compared to achieving the same BER with the block code. □

The rest of this section is structured as follows. First, we discuss the construction of some *time-invariant* LDPC convolutional codes based on the Tanner unwrapping technique. In this context, we make a simple observation about how the syndrome former memory can sometimes be reduced without changing the convolutional code. Second, we present a construction of *time-varying* LDPC convolutional codes based on iterated graph-cover constructions. An important subtopic here will be an investigation of the influence of the "diagonal cut" (which is used to define a graph cover) on the decoding performance.

### A. Construction of Time-Invariant LDPC Convolutional Codes Based on the Tanner Unwrapping Technique

In this section, we revisit a class of quasi-cyclic LDPC codes and their associated convolutional codes that were studied in [36]. As we will see, they are instances of the quasi-cyclic code construction in Example 4 and Remark 5, and the corresponding convolutional code construction based on Tanner's unwrapping technique in Example 10.

*Example 15:* Consider the regular proto-matrix

$$\boldsymbol{A} \triangleq \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{14}$$

with $m_{\boldsymbol{A}} = 3$ and $n_{\boldsymbol{A}} = 5$. We apply **GCC1**, as in Example 4 and Remark 5, with an interesting choice of permutation matrices first suggested by Tanner [37] that yields the parity-check matrix

$$\boldsymbol{H}_{\mathrm{QC}}^{(r)} \triangleq \begin{bmatrix} \boldsymbol{I}_1 & \boldsymbol{I}_2 & \boldsymbol{I}_4 & \boldsymbol{I}_8 & \boldsymbol{I}_{16} \\ \boldsymbol{I}_5 & \boldsymbol{I}_{10} & \boldsymbol{I}_{20} & \boldsymbol{I}_9 & \boldsymbol{I}_{18} \\ \boldsymbol{I}_{25} & \boldsymbol{I}_{19} & \boldsymbol{I}_7 & \boldsymbol{I}_{14} & \boldsymbol{I}_{28} \end{bmatrix} \tag{15}$$

where as before $\boldsymbol{I}_s$ is an $s$ times left-circularly shifted identity matrix of size $r \times r$ and $r > 28$. The corresponding polynomial parity-check is

$$\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X) \triangleq \begin{bmatrix} X^1 & X^2 & X^4 & X^8 & X^{16} \\ X^5 & X^{10} & X^{20} & X^9 & X^{18} \\ X^{25} & X^{19} & X^7 & X^{14} & X^{28} \end{bmatrix}.$$

The resulting quasi-cyclic $(3, 5)$-regular LDPC block codes have block length $n = 5 \cdot r$. In particular, for $r = 31, r = 48$, and $r = 80$, we obtain codes of length 155, 240, and 400, respectively, whose simulated BER performance results are shown in Fig. 7. The choice $r = 31$ yields the well-known length-155 quasi-cyclic block code that was first introduced by Tanner [37] (see also the discussion in [18]).

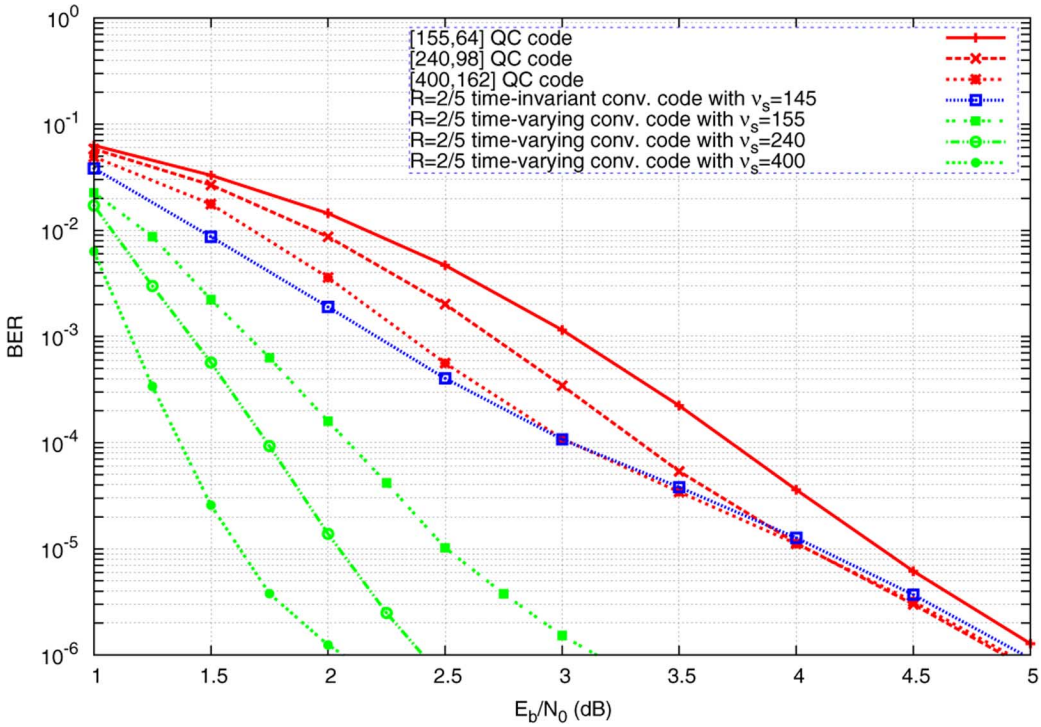


Fig. 7. Performance of three $(3, 5)$-regular quasi-cyclic LDPC block codes and their associated time-invariant and time-varying LDPC convolutional codes. (Note that the small gaps that appear between the second, third, and fourth curves for high signal-to-noise ratios are caused by a slight difference in code rates due to the existence of redundant rows in the block code parity-check matrices.)

Unwrapping these codes by the Tanner unwrapping technique as in Example 10, we obtain a rate-$2/5$ time-invariant convolutional code with $\nu_{\rm s} = 145$ defined by the polynomial parity-check matrix

$$\boldsymbol{H}_{\rm conv}(D) \triangleq \begin{bmatrix} D^1 & D^2 & D^4 & D^8 & D^{16} \\ D^5 & D^{10} & D^{20} & D^9 & D^{18} \\ D^{25} & D^{19} & D^7 & D^{14} & D^{28} \end{bmatrix}.$$

Its decoding performance is also shown in Fig. 7 under the label "$R = 2/5$ time-invariant conv. code with $\nu_{\rm s} = 145$." We conclude this example with a few remarks.

- Fig. 7 shows that the convolutional code exhibits a "convolutional gain" of between 0.5 and 0.7 dB compared to the $[155, 64]$ quasi-cyclic LDPC block code at moderate BERs and that the gain remains between 0.15 and 0.3 dB at lower BERs.
- Note that the polynomial parity-check matrix $\boldsymbol{H}_{\rm conv}(D)$ that is obtained by the Tanner unwrapping technique is independent of the parameter $r$ of the polynomial parity-check matrix $\boldsymbol{H}^{(r)}_{\rm QC}(X)$, as long as $r$ is strictly larger than the largest exponent appearing in $\boldsymbol{H}^{(r)}_{\rm QC}(X)$. Moreover, for $r \to \infty$, the Tanner graph of $\boldsymbol{H}^{(r)}_{\rm QC}(X)$ is closely related to the Tanner graph of $\boldsymbol{H}_{\rm conv}(D)$, and so it is not surprising to see that, for larger $r$, the decoding performance of quasi-cyclic LDPC block codes based on $\boldsymbol{H}^{(r)}_{\rm QC}(X)$ tends to the decoding performance of the LDPC convolutional based on $\boldsymbol{H}_{\rm conv}(D)$, as illustrated by the two curves labeled "$[240, 98]$ QC code" and "$[400, 162]$ QC code" in Fig. 7.
- The permutation matrices (more precisely, the circulant matrices) that were used for constructing the quasi-cyclic codes in this example were *not* chosen to optimize the Hamming distance or the pseudoweight properties of the code. In particular, a different choice of circulant matrices may result in better high-SNR performance, i.e., in the so-called "error floor" region of the BER curve. For choices of codes with better Hamming distance properties, we refer the reader to [38].
- The remaining curves in Fig. 7 will be discussed in Example 18. □

We conclude this section with some comments on the syndrome former memory $m_{\rm s}$ of the convolutional codes obtained by the Tanner unwrapping technique, in particular how this syndrome former memory $m_{\rm s}$ can sometimes be reduced without changing the convolutional code.

Assume that we have obtained a polynomial parity-check matrix $\boldsymbol{H}_{\rm conv}(D)$ from $\boldsymbol{H}^{(r)}_{\rm QC}(X)$ according to the Tanner method. Clearly, the syndrome former memory $m_{\rm s}$ is given by the largest exponent that appears in $\boldsymbol{H}_{\rm conv}(D)$. In some instances there is a simple way of reducing $m_{\rm s}$ without changing the convolutional code. Namely, if $e$ is the *minimal* exponent that appears in the polynomials of a given row of $\boldsymbol{H}_{\rm conv}(D)$, then the polynomials in this row of $\boldsymbol{H}_{\rm conv}(D)$ can be divided by $D^e$. We illustrate this syndrome former memory reduction for the small convolutional code that appeared in Example 10.

*Example 16:* Applying the Tanner unwrapping technique to the polynomial parity-check matrix $\boldsymbol{H}^{(r)}_{\rm QC}(X)$ of the quasi-cyclic LDPC code with $r = 7$ in Remark 5, we obtain

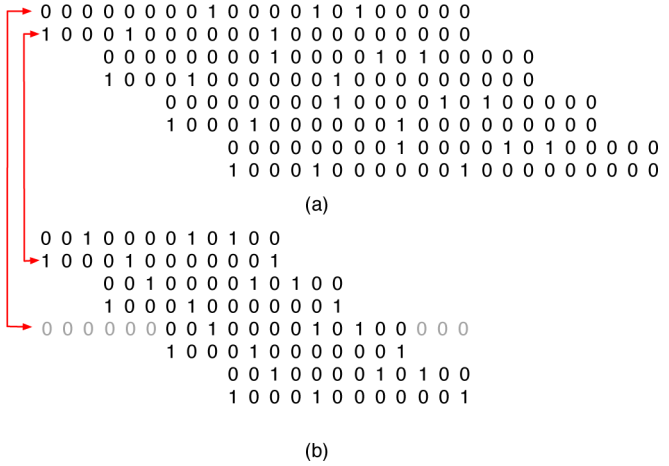


Fig. 8. Parts of the scalar parity-check matrices [see (2)] corresponding to the two equivalent LDPC convolutional codes with syndrome former memories (a) $m_{\mathrm{s}} = 6$ and (b) $m_{\mathrm{s}} = 3$.

$\boldsymbol{H}_{\mathrm{conv}}(D)$ of a rate-$1/3$ time-invariant LDPC convolutional code, as shown in Example 10, with syndrome former memory $m_{\mathrm{s}} = 6$. Following the procedure discussed in the paragraph above, the first and second rows of $\boldsymbol{H}_{\mathrm{conv}}(D)$ can be divided by $D^1$ and $D^3$, respectively, to yield an equivalent convolutional code with syndrome former memory $m_{\mathrm{s}} = 3$ and polynomial parity-check matrix

$$\boldsymbol{H}_{\mathrm{conv}}(D) = \begin{bmatrix} D^0 & D^1 & D^3 \\ D^3 & D^2 & D^0 \end{bmatrix}. \tag{16}$$

Fig. 8 shows parts of the corresponding scalar parity-check matrix $\tilde{\boldsymbol{H}}_{\mathrm{conv}}$ for $m_{\mathrm{s}} = 3$, together with the original scalar parity-check matrix for $m_{\mathrm{s}} = 6$, and illustrates the equivalence of the two matrices in the sense that only the ordering of the rows is different, which does not affect the corresponding convolutional code. In this example, the order of the even-numbered rows stays the same, while the odd-numbered rows are shifted by four positions. The equivalence of the two parity-check matrices can be seen by noting that the parity-check matrix, outside of the diagonal structure, is filled with zeros. □

### *B. Construction of Time-Varying LDPC Convolutional Codes Based on Iterated Graph-Cover Constructions*

As was seen in Example 9, interesting graph covers can be obtained by combining **GCC1** with **GCC2**, or *vice versa*. Inspired by that example, this subsection considers iterated graph-cover constructions for constructing Tanner graphs of LDPC convolutional codes, in particular of time-varying LDPC convolutional codes.

*Definition 17:* Based on a combination of **GCC1** and **GCC2**, and the code-shortening concept introduced in Section III-C, we propose the following construction of LDPC convolutional codes.

1) We start with a proto-matrix $\boldsymbol{A}$ of size $m_{\boldsymbol{A}} \times n_{\boldsymbol{A}}$.
2) We apply **GCC1** to $\boldsymbol{A}$ with finite-size permutation matrices and obtain the matrix $\boldsymbol{A}'$.
3) We apply **GCC2** to $\boldsymbol{A}'$ with permutation matrices that are bi-infinite Toeplitz matrices and obtain the matrix $\boldsymbol{A}''$.
4) Finally, looking at $\boldsymbol{A}''$ as the parity-check matrix of a bi-infinite convolutional code, we obtain the parity-check matrix of a convolutional code by shortening the code bit positions corresponding to negative time indices.

Here, Steps 3 and 4 can be seen as an application of the JFZ unwrapping method. □

The following example shows how this construction can be applied to obtain LDPC convolutional codes with excellent performance. (In the example, where suitable, we will refer to the analogous matrices of Example 9 and Fig. 6 that were used to illustrate the iterated graph-cover construction.)

*Example 18:* Based on Definition 17, we construct an LDPC convolutional code by performing the following steps.

1) We start with the same regular proto-matrix $\boldsymbol{A}$ as in Example 15, for which $m_{\boldsymbol{A}} = 3$ and $n_{\boldsymbol{A}} = 5$.
2) We apply **GCC1** to $\boldsymbol{A}$ with permutation matrices that are circulant matrices of size $r \times r$ and obtain the parity-check matrix $\boldsymbol{A}' = \boldsymbol{H}_{\mathrm{QC}}^{(r)}$ shown in (15), which is the analogue of $\boldsymbol{A}^{(1)}$ in Fig. 6(a).
3) We apply **GCC2** to $\boldsymbol{A}' = \boldsymbol{H}_{\mathrm{QC}}^{(r)}$ with permutation matrices that are bi-infinite Toeplitz matrices and obtain a new parity-check matrix $\boldsymbol{A}''$. This is analogous to the transition of the matrix $\boldsymbol{A}^{(1)}$ in Fig. 6(a) to the matrix $\overline{\boldsymbol{B}}^{(1)}$ in Fig. 6(c). The "diagonal cut" is obtained by alternately moving $n_{\boldsymbol{A}} = 5$ units to the right and then $m_{\boldsymbol{A}} = 3$ units down.
4) Finally, we obtain the desired convolutional code by shortening the code bit positions corresponding to negative time indices.

For the choices $r = 31, 48, 80$, this construction results in rate-$2/5$ time-varying convolutional codes with syndrome former memory $m_{\mathrm{s}} = 30, 47, 79$, respectively, and with constraint length $\nu_{\mathrm{s}} = (m_{\mathrm{s}} + 1) \cdot n_{\boldsymbol{A}} = 155, 240, 400$, respectively. The label "time-varying" is indeed justified because the convolutional codes constructed here can be expressed in the form of the parity-check matrix in (1) with a suitable choice of syndrome former memory $m_{\mathrm{s}}$, nontrivial period $T_{\mathrm{s}}$, and submatrices $\{\boldsymbol{H}_i(t)\}_i$.

The decoding performance of these codes is shown in Fig. 7, labeled "$R = 2/5$ time-varying conv. code with $\nu_{\mathrm{s}} = \ldots$." As originally noted in [39], we observe that these three LDPC convolutional codes achieve significantly better performance at a BER of $10^{-6}$ than the other codes shown in this plot, namely with "convolutional gains" of 2.0 dB for the $\nu_{\mathrm{s}} = 155$ convolutional code, 2.4 dB for the $\nu_{\mathrm{s}} = 240$ convolutional code, and 2.8 dB for the $\nu_{\mathrm{s}} = 400$ convolutional code, compared to the three respective underlying LDPC block codes.

In order to compare these codes based on a given decoding processor (hardware) complexity, we consider a block code of length $n = \nu_{\mathrm{s}}$ (see [40] and [41]). The above time-varying convolutional code for $r = 31$ has constraint length $\nu_{\mathrm{s}} = (m_{\mathrm{s}} + 1) \cdot c = 155$, and hence approximately the same processor complexity as the quasi-cyclic block code of length $n = 155$ in Fig. 7 and the time-invariant convolutional code with $\nu_{\mathrm{s}} = 145$ in Fig. 7, but it achieves large gains compared to both of these codes. We note, in addition, that the performance

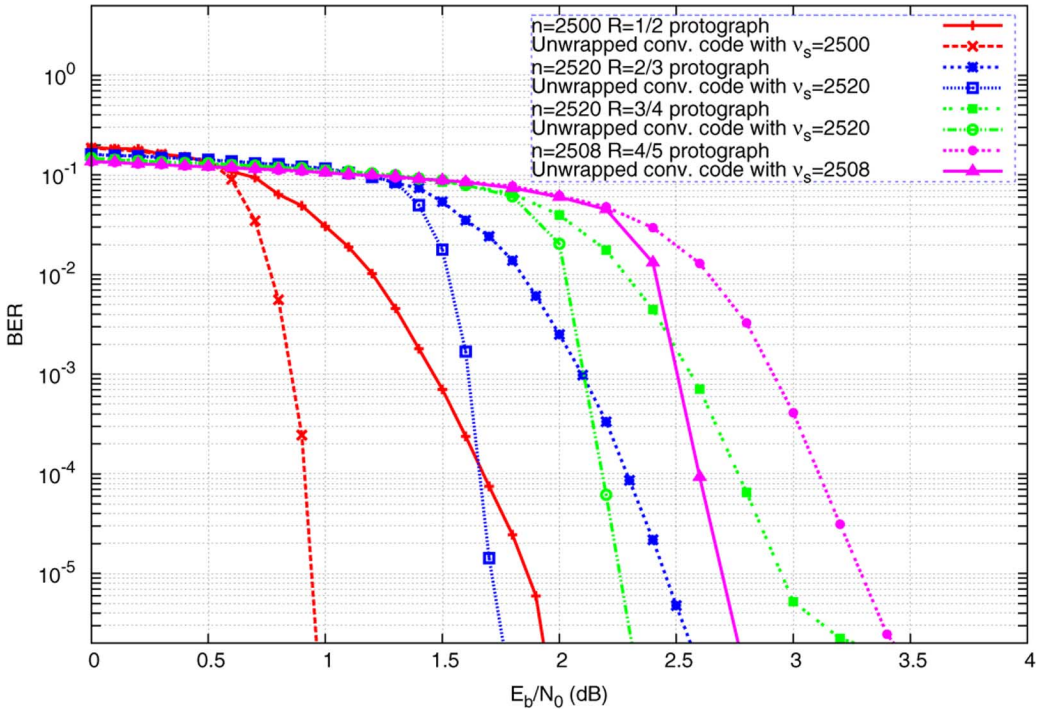


Fig. 9. Performance of a family of irregular proto-graph-based LDPC block codes and the associated time-varying LDPC convolutional codes.

of the time-varying convolutional code with constraint length $\nu_s = 400$ is quite remarkable, since, at a BER of $10^{-5}$, it performs within 1 dB of the iterative decoding threshold of 0.965 dB, while having the same processor complexity as a block code of length only $n = 400$. In Section VI-C, we discuss some possible reasons for these "convolutional gains," along with their associated implementation costs in terms of decoder memory and decoding delay. □

We make the following observations with respect to the above definition and example.

- The LDPC code construction in the above example yields time-varying LDPC convolutional codes with syndrome former memory $m_s \leq r-1$ and period $T_s = r$. Most importantly, varying $r$ in the above construction leads to different LDPC convolutional codes. This is in contrast to the Tanner unwrapping technique discussed in Section IV-A, where the obtained LDPC convolutional code is independent of the parameter $r$, as long as $r$ is strictly larger than the largest exponent in $\boldsymbol{H}_{\mathrm{QC}}^{(r)}(X)$.
- As mentioned previously in Example 9, the iterated graph-cover construction based on the combination of **GCC1** and **GCC2** yields Tanner graphs that have a "pseudorandom" structure, a structure that seems to be beneficial as indicated by the above simulation results. (We remark that the improved performance of the time-varying LDPC convolutional codes obtained by unwrapping a randomly constructed LDPC block code was first noted by Lentmaier *et al.* [42].)
- Instead of constructing a first parity-check matrix as in Step 2 of Definition 17, one can also start with any other (randomly or nonrandomly constructed, regular or irregular) parity-check matrix, and still achieve a "convolutional gain." The next example is an illustration of this point.

*Example 19:* As was done in [41], one can replace the parity-check matrix that was constructed in Step 2 of Definition 17 by an irregular LDPC block code with optimized iterative decoding thresholds. In particular, one can start with the parity-check matrix of the rate-$1/2$ irregular proto-graph-based code from [43] with an iterative decoding threshold of 0.63 dB, and several of its punctured versions. Fig. 9 shows simulation results for the obtained block and convolutional codes. Each simulated block code had a block length of about 2500, with code rates ranging from $1/2$ to $4/5$. We see that "convolutional gains" ranging from 0.6 to 0.9 dB at a BER of $10^{-5}$ were obtained.

Similarly, it was shown in [24] that an LDPC convolutional code derived from a randomly constructed rate-$1/2$ irregular LDPC block code with block length 2400 outperformed the underlying code by almost 0.8 dB at a BER of $10^{-5}$. The degree distribution of the underlying LDPC block code was fully optimized and had an iterative decoding threshold of 0.3104 dB [11]. □

Of course, there are other ways of applying the "diagonal cut" in Step 3 of Example 18, and so it is natural to investigate the influence of different "diagonal cuts" on the decoding performance. We will do this in the next few paragraphs by extending the discussion that was presented right after Example 1.

We start by assuming that the matrix after Step 2 of Definition 17 has size $m \times n$, and define $\eta \triangleq \gcd(m,n)$. Then, for any positive integer $\ell$ that divides $\eta$, we can perform a "diagonal cut" where we alternately move $c' = \ell \cdot (n/\eta)$ units to the right and then $c' - b' \triangleq \ell \cdot (m/\eta)$ units down [i.e., $b' = \ell \cdot ((n-m)/\eta)$]. With this, the obtained convolutional code is a

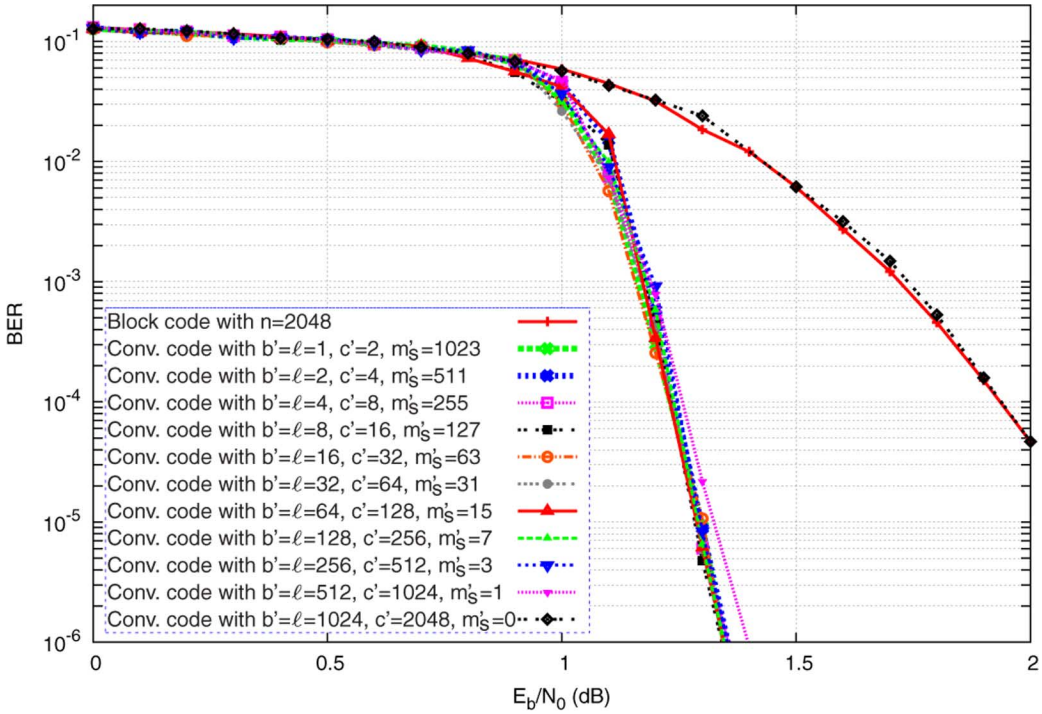


Fig. 10. Performance of a family of LDPC convolutional codes obtained from a $(3, 6)$-regular LDPC block code using different step sizes.

periodically time-varying LDPC convolutional code with rate $R' = b'/c' = b/c$, syndrome former memory $m_s' = (n/c') - 1 = (\eta/\ell) - 1$, period $T_s' = m_s' + 1 = n/c' = \eta/\ell$, and constraint length $\nu_s' = c' \cdot (m_s' + 1) = n$. (Note that the syndrome former memory $m_s'$ depends on $\ell$, but the constraint length $\nu_s'$ is independent of $\ell$.)

*Example 20:* Here we simulate the performance of some LDPC convolutional codes obtained according to the above generalization of the "diagonal cut." Namely, we start with a randomly-constructed $(3, 6)$-regular LDPC block code based on a parity-check matrix of size $1024 \times 2048$. Therefore $m = 1024, n = 2048$, and $\eta \triangleq \gcd(m, n) = 1024$. (Note that $c' = \ell \cdot (n/\eta) = 2\ell$ and $b' = \ell \cdot ((n - m)/\eta) = \ell$ in this case.) Fig. 10 shows the performance of the resulting family of LDPC convolutional codes, where $\ell$ varies in powers of 2 from 1 to 1024, each with constraint length $\nu_s' = 2048$. We make the following observations. First, the case $\ell = 1024$ is not interesting because it results in $m_s' = 0$, i.e., it is a trivial concatenation of copies of the block code, and so the BER is the same as for the underlying block code. Secondly, for all other choices of $\ell$, the constructed codes perform very similarly, each exhibiting a sizable "convolutional gain" compared to the block code, although the syndrome former memory $m_s'$ is different in each case. □

A special case of the above code construction deserves mention. When $\eta = 1$, i.e., $m$ and $n$ are relatively prime, the only possible step size is obtained by choosing $\ell = \eta = 1$, which results in the above-mentioned uninteresting case of trivial concatenations of copies of the block code. However, all-zero columns can be inserted in the parity-check matrix such that a value of $\eta > 1$ is obtained, which allows a step size to be chosen that results in a convolutional code with $m_s' > 0$. The variable nodes corresponding to the all-zero columns are not transmitted, i.e., they are punctured, so that the rate corresponds to the size of the original parity-check matrix.

For the "diagonal cut" LDPC convolutional code constructions discussed above, the unwrapped convolutional codes have the minimum possible constraint length $\nu_s'$, which is equal to the block length of the underlying block code. Although this is a desirable property for practical implementation, we do not need to limit ourselves to diagonal cuts in general.

Inspired by the graph-cover construction of Fig. 5(b) and (d) in Example 8, instead of a "diagonal cut" we now consider a "random cut," which we define as a partition of the parity-check matrix into two matrices that add up (over $\mathbb{Z}$) to the parity-check matrix. Despite the randomness of this approach, several of the key unwrapping properties of the "diagonal cut" are preserved. For example, the computational complexity per decoded bit does not change, since the degree distributions of the resulting codes are all equal.[14] However, the LDPC convolutional codes based on a "random cut" typically require larger decoding processor sizes as a result of increased code constraint lengths.

*Example 21:* We continue Example 20; however, instead of performing "diagonal cuts," we perform "random cuts." Fig. 11 shows the performance of five such LDPC convolutional codes, each with rate $1/2$ and constraint length $\nu_s' = 4096$, compared to the underlying block code and the LDPC convolutional code constructed in Example 20 (with parameters $\ell = 1, b' = 1, c' = 2$, and $\nu_s' = 2048$). We note that the increase in constraint length from $\nu_s' = 2048$ to $\nu_s' = 4096$ due to the "random cut" results

[14]This is guaranteed by choosing a random partition of the block code parity-check matrix and then using this partition to construct one period of the time-varying convolutional code parity-check matrix.

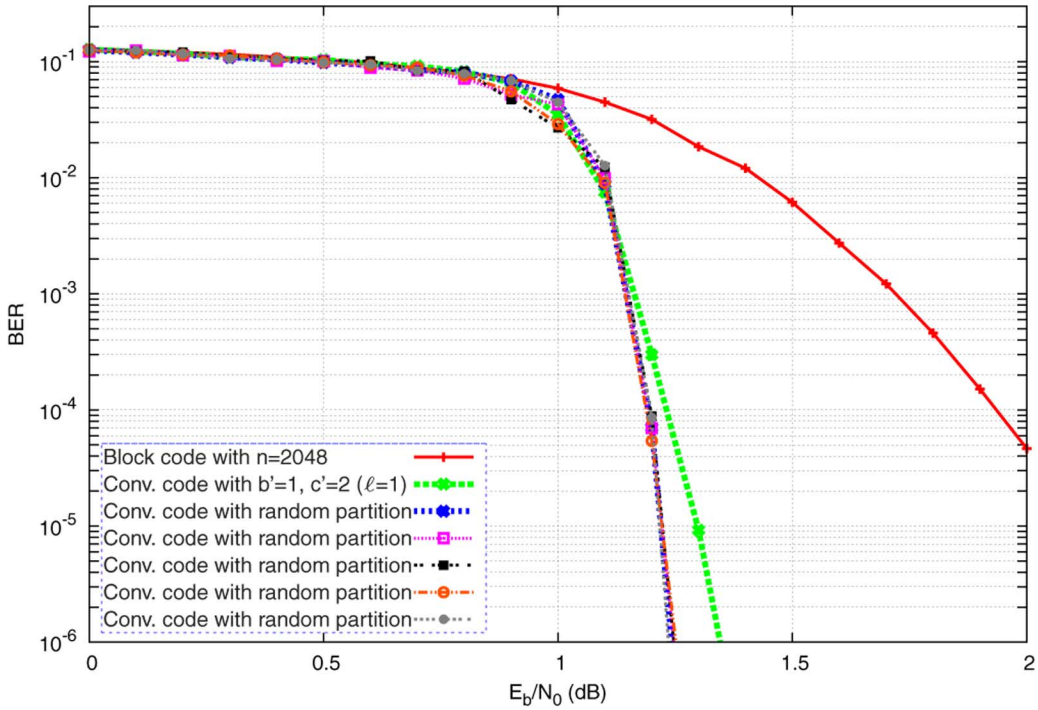


Fig. 11. Performance of "randomly unwrapped" LDPC convolutional codes obtained from a $(3, 6)$-regular LDPC block code using random partitions.

in a small additional coding gain in exchange for the larger decoding processor size. □

Finally, we note that, for a size $m \times n$ sparse parity-check matrix $\boldsymbol{H}$ with $p$ nonzero entries, there are a total of $2^{mn}$ possible ways of choosing a random cut. However, due to the sparsity, there are only $2^p$ distinct random cuts, where $p \ll m \cdot n$.

## V. Connections to Other LDPC Codes Based on Graph-Cover Constructions

In this section, we briefly discuss some other graph-cover-based LDPC code constructions proposed in the literature, namely by Ivkovic *et al.* [44], Divsalar *et al.* [43], [45], Lentmaier *et al.* [46], [47], and Kudekar *et al.* [48].

### A. LDPC Code Construction by Ivkovic et al.

The LDPC code construction by Ivkovic *et al.* [44] can be seen as an application of the graph-cover construction in Figs. 5(b) and (d) in Example 8. Namely, in terms of our notation, Ivkovic *et al.* [44] start with a parity-check matrix $\boldsymbol{H}$, choose the set $\mathcal{L} \triangleq \{0, 1\}$, a collection of zero-one matrices $\{\boldsymbol{H}_0, \boldsymbol{H}_1\}$ such that $\boldsymbol{H} = \boldsymbol{H}_0 + \boldsymbol{H}_1$ (in $\mathbb{Z}$), and the collection of permutation matrices

$$\{\boldsymbol{P}_0, \boldsymbol{P}_1\}_{\ell \in \mathcal{L}} \triangleq \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \right\}.$$

Most importantly, the decomposition of $\boldsymbol{H}$ into $\boldsymbol{H}_0$ and $\boldsymbol{H}_1$ is done such that trapping sets that were present in the Tanner graph of $\boldsymbol{H}$ are not present in the Tanner graph of the new parity-check matrix. In addition, Ivkovic *et al.* give guarantees on the relationship between the minimum Hamming distances of the old and new code.[15]

### B. LDPC Code Construction by Divsalar et al.

One of the LDPC code constructions by Divsalar *et al.* [43], [45] is the so-called rate-$1/2$ AR4JA LDPC code construction, which was also considered earlier in Example 19. A particularly attractive, from an implementation perspective, version of this code construction is obtained by an iterated graph-cover construction procedure, where each graph-cover construction is based on a cyclic cover, as in the application of **GCC1** in Example 4. Although cyclic covers result in simplified encoding and decoding circuitry, codes based on cyclic covers are known to have the disadvantage that the minimum Hamming distance is upper bounded by a number that is a function of the proto-graph structure [49], [50]. However, because the cyclic cover of a cyclic cover of the proto-graph is *not necessarily* a cyclic cover of the proto-graph, such disadvantages are avoided to a certain extent in the AR4JA LDPC code construction. Nevertheless, ultimately the minimum Hamming distance of such codes will also be upper bounded by some number; however, these bounds usually become relevant only beyond the code length of interest.[16]

### C. LDPC Code Construction by Lentmaier et al. and Kudekar et al.

The LDPC code constructions by Lentmaier *et al.* [46], [47] and Kudekar *et al.* [48] can also be seen as iterated graph-cover constructions. We now describe a specific instance of this construction.

- It starts with a proto-matrix $\boldsymbol{A} \triangleq [3\ 3]$.
- The first graph-cover construction is very similar to the bi-infinite graph-cover construction in Example 6 and Fig. 4. Namely, in terms of our notation, we define the set $\mathcal{L} \triangleq \{0, 1, 2, 3, 4, 5\}$, the collection of matrices $\{\boldsymbol{A}_\ell\}_{\ell \in \mathcal{L}}$

[15] See also the discussion of similar results in [49, Appendix J].

[16] For this statement we assume that the degree of the first cover is fixed.

with $\boldsymbol{A}_0 = \boldsymbol{A}_1 = \boldsymbol{A}_2 = [1\ 0]$ and $\boldsymbol{A}_3 = \boldsymbol{A}_4 = \boldsymbol{A}_5 = [0\ 1]$, and the collection of permutation matrices $\{\boldsymbol{P}_\ell\}_{\ell \in \mathcal{L}}$ with $\boldsymbol{P}_0 \triangleq \boldsymbol{T}_0, \boldsymbol{P}_1 \triangleq \boldsymbol{T}_1, \boldsymbol{P}_2 \triangleq \boldsymbol{T}_2, \boldsymbol{P}_3 \triangleq \boldsymbol{T}_0, \boldsymbol{P}_4 \triangleq \boldsymbol{T}_1, \boldsymbol{P}_5 \triangleq \boldsymbol{T}_2$, where as before $\boldsymbol{T}_s$ is a bi-infinite Toeplitz matrix with zeros everywhere except for ones in the $s$th diagonal below the main diagonal.
- The second graph-cover construction is a random graph-cover construction of cover-degree $M$.
- The code is shortened. Namely, for some positive integer $L$ all codeword indices corresponding to values outside the range $[-LM, LM]$ are shortened.[17]

We now point out some differences between this code construction and the LDPC convolutional code construction in Definition 17. Namely, the LDPC code ensemble constructed above has the following properties.

- The first graph-cover construction is based on bi-infinite Toeplitz permutation matrices, and the second graph-cover construction is based on finite-size permutation matrices.
- The analysis focuses on the case where $M$ and $L$ go to infinity (in that order), i.e., for a fixed $L$ the parameter $M$ tends to infinity. Afterwards, $L$ tends to infinity.
- The number of check nodes with degree smaller than 6 in the Tanner graph is proportional to $M$.
- In [48], for the binary erasure channel, when $M$ and $L$ go to infinity (in that order), Kudekar *et al.* prove that the sum-product algorithm decoding threshold for a slight variation of the above-mentioned ensemble of codes equals the maximum a-posteriori decoding threshold for the ensemble of $(3, 6)$-regular LDPC codes. This is a very remarkable property. (In [51], using density evolution methods, Lentmaier *et al.* give numerical evidence that this statement might also hold for binary-input output-symmetric channels beyond the binary erasure channel.)

On the other hand, the codes constructed in Definition 17 have the following properties. [We assume that the underlying block code is a $(3, 6)$-regular LDPC code.]

- The first graph-cover construction is based on finite-size permutation matrices, and the second graph-cover construction is based on bi-infinite Toeplitz permutation matrices.
- In a typical application of this construction, $r$ is fixed.
- The number of check nodes with degree smaller than 6 in the Tanner graph of the LDPC convolutional code is proportional to $r$.
- For a binary-input output-symmetric channel, the performance of the unterminated LDPC convolutional code under the continuous sliding window sum-product algorithm decoding discussed in Section II-B improves with increasing $r$ (see, e.g., Fig. 7), but the ultimate asymptotic threshold of such unterminated decoding is unknown.[18]

[17]Although this code construction method could be presented such that the shortening is done between the two graph-cover construction steps, namely by shortening all codeword indices that correspond to values outside the range $[-L, L]$, we have opted to present the code construction such that the shortening is done after the two graph-cover construction steps. In this way, the structure of the code construction description matches better the description in Definition 17.

[18]Lentmaier *et al.* have shown in [46] and [47] that properly terminated LDPC convolutional codes become equivalent to the LDPC block codes constructed by Kudekar *et al.* in [48] and inherit their excellent asymptotic threshold properties, but whether this is true for unterminated LDPC convolutional codes is still an open question.

The differences between these two code families come mainly from the fact that the codes constructed by Lentmaier *et al.* and Kudekar *et al.* are essentially block codes, although sophisticated ones, whereas the codes in Definition 17 are convolutional codes, along with their advantages and disadvantages. In particular, the way the limits of the parameters are taken, there is a significant difference in the fraction of check nodes with degree strictly smaller than 6. Namely, in the case of the codes by Lentmaier *et al.* and Kudekar *et al.* this fraction is a fixed nonzero function of $L$ (here we assume fixed $L$ and $M \to \infty$), whereas in the case of the codes considered in this paper, this fraction is zero (here we assume fixed $r$ and an unterminated convolutional code).

We conclude this section with the following remarks. Namely, although the convolutional codes in Definition 17 may not enjoy the same asymptotic thresholds as the block code constructions by Lentmaier *et al.* and by Kudekar *et al.*, they lend themselves to a continuous decoding architecture, as described in Section II-B, which can be advantageous in certain applications, such as data streaming, without a predetermined frame structure. More importantly, however, it is very encouraging that the simulation results reported in this paper indicate that sizable "convolutional gains" are already visible for very reasonable constraint/code lengths. In the next section, we discuss some possible reasons for these gains. Finally, it is worth noting that, as the block lengths and associated constraint lengths of the constructions presented in this section become larger, the observed "convolutional gains" will become smaller since the block code results will approach their respective thresholds.

## VI. Analysis of Derived LDPC Convolutional Codes

This section collects some analytical results about LDPC convolutional codes. In particular, we compare the existence/nonexistence of cycles in LDPC block and LDPC convolutional codes, we present some properties of pseudocodewords, and we discuss the—mostly moderate—cost increase in decoder complexity that is incurred by going from LDPC block to LDPC convolutional codes.

### A. Graph-Cycle Analysis

It is well known that cycles in the Tanner graph representation of a sparse code affect message-passing iterative decoding algorithms, with short cycles generally pushing the performance further away from optimum. (Indeed, attempts to investigate and minimize these effects have been made in [52] and [53], where the authors propose LDPC code construction procedures to maximize the connectivity of short cycles to the rest of the graph, thus also maximizing the independence of the messages flowing through a cycle.) Hence it is common practice to design codes that do not contain short cycles, so as to obtain independent messages in at least the initial iterations of the decoding process.

Avoiding cycles in Tanner graphs also has the benefit of avoiding pseudocodewords.[19] To see this, let the active

[19]Here and in the following, pseudocodewords refer to pseudocodewords as they appear in linear programming (LP) decoding [54], [55] and in the graph-cover-based analysis of message-passing iterative decoding in [22], [23]. For other notions of pseudocodewords, in particular computation tree pseudocodewords, we refer to the discussion in [56].

TABLE I
AVERAGE (PER BIT NODE) NUMBER $\bar{N}_\ell$ OF CYCLES OF LENGTH $\ell$ FOR THE TANNER GRAPHS OF THE BLOCK CODES (BCS) OF BLOCK LENGTH $n$ AND CONVOLUTIONAL CODES (CCS) OF CONSTRAINT LENGTH $\nu_{\mathrm{s}}$ DISCUSSED IN EXAMPLE 22. (ALL TANNER GRAPHS HAVE GIRTH 8.)

| Code | $\bar{N}_8$ | $\bar{N}_{10}$ | $\bar{N}_{12}$ |
|---|---|---|---|
| BC ($n = 155$) | 3.000 | 24.000 | 146.000 |
| BC ($n = 240$) | 2.600 | 14.000 | 93.400 |
| BC ($n = 400$) | 2.200 | 12.400 | 70.600 |
| Time-invariant CC ($\nu_{\mathrm{s}} = 145$) | 2.200 | 12.400 | 70.200 |
| Time-varying CC ($\nu_{\mathrm{s}} = 155$) | 0.910 | 8.342 | 44.813 |
| Time-varying CC ($\nu_{\mathrm{s}} = 240$) | 0.917 | 5.338 | 30.242 |
| Time-varying CC ($\nu_{\mathrm{s}} = 400$) | 0.675 | 4.705 | 24.585 |

TABLE II
AVERAGE (PER BIT NODE) NUMBER $\bar{N}_\ell$ OF CYCLES OF LENGTH $\ell$ FOR THE TANNER GRAPHS OF THE BLOCK CODES (BCS) OF BLOCK LENGTH $n$ AND CONVOLUTIONAL CODES (CCS) OF CONSTRAINT LENGTH $\nu_{\mathrm{s}}$ DISCUSSED IN EXAMPLE 23. (ALL TANNER GRAPHS HAVE GIRTH 4.)

| Code | $\bar{N}_4$ | $\bar{N}_6$ |
|---|---|---|
| Rate-1/2 BC ($n = 2500$) | 0.013 | 0.120 |
| Rate-2/3 BC ($n = 2520$) | 0.065 | 0.839 |
| Rate-3/4 BC ($n = 2520$) | 0.136 | 2.710 |
| Rate-4/5 BC ($n = 2508$) | 0.250 | 6.544 |
| Rate-1/2 time-varying CC ($\nu_{\mathrm{s}} = 2500$) | 0.010 | 0.064 |
| Rate-2/3 time-varying CC ($\nu_{\mathrm{s}} = 2520$) | 0.044 | 0.483 |
| Rate-3/4 time-varying CC ($\nu_{\mathrm{s}} = 2520$) | 0.091 | 1.465 |
| Rate-4/5 time-varying CC ($\nu_{\mathrm{s}} = 2508$) | 0.173 | 3.622 |

part of a pseudocodeword be defined as the set of bit nodes corresponding to the support of the pseudocodeword, along with the adjacent edges and check nodes. With this, it holds that the active part of any pseudocodeword contains at least one cycle and/or at least one bit node of degree one. And so, given that the typical Tanner graph under consideration in this paper does not contain bit nodes of degree one, the active part of a pseudocodeword must contain at least one cycle. Therefore, avoiding cycles implicitly means avoiding pseudocodewords.[20]

Let $\tilde{\boldsymbol{H}}$ and $\boldsymbol{H}$ be two parity-check matrices such that $\mathsf{T}(\tilde{\boldsymbol{H}})$ is a graph cover of $\mathsf{T}(\boldsymbol{H})$. It is a well-known result that any cycle in $\mathsf{T}(\tilde{\boldsymbol{H}})$ can be mapped into a cycle in $\mathsf{T}(\boldsymbol{H})$. This has several consequences. In particular, the girth of $\mathsf{T}(\tilde{\boldsymbol{H}})$ is at least as large as the girth of $\mathsf{T}(\boldsymbol{H})$, and more generally, $\mathsf{T}(\tilde{\boldsymbol{H}})$ contains fewer short cycles than $\mathsf{T}(\boldsymbol{H})$.[21] For the codes constructed in this paper, this means that the unwrapping process (from block code to convolutional code) can "break" some cycles in the Tanner graph of the block code.

We now revisit some codes that were discussed in earlier sections and analyze their graph cycle structure using a brute-force search algorithm.[22] Note that, in order to accurately compare the graph cycle distributions of two codes with different block/constraint lengths, we compute the total number of cycles of a given cycle length per block/constraint length, and divide this number by the block/constraint length.[23]

*Example 22:* Consider the LDPC block and convolutional codes that were constructed in Examples 15 and 18 and whose BER performance was plotted in Fig. 7. Table I shows the average number of cycles of certain lengths for the Tanner graphs of the quasi-cyclic block codes, for the Tanner graph of the corresponding time-invariant convolutional code, and for the Tanner graph of the time-varying convolutional codes. □

[20]Note that the support of any pseudocodeword is a stopping set [22], [23], [57].

[21]This observation has been used in many different contexts over the past ten years in the construction of LDPC and turbo codes; in particular, it was used in [42], where the authors dealt with bounding the girth of the resulting LDPC convolutional codes.

[22]The search technique that we used is based on evaluating the diagonal entries of the powers of the matrix $\boldsymbol{M}$ defined in [33, eq. (3.1)]. Note that this search technique works only for counting cycles of length smaller than twice the girth of the graph. For searching longer cycles, more sophisticated algorithms are needed.

[23]For LDPC convolutional codes, we have made use of the periodicity of the parity-check matrices in order to complete the search in a finite number of steps.

*Example 23:* Table II shows the cycle analysis results for the rate-$1/2$ proto-graph-based codes that were discussed in Example 19 and whose BER performance was plotted in Fig. 9. □

From Examples 22 and 23, we see that many of the short cycles in the Tanner graphs of the LDPC block codes are "broken" to yield cycles of larger length in the Tanner graphs of the derived LDPC convolutional codes.

### B. Pseudocodeword Analysis

This section collects some comments concerning the pseudocodewords of the parity-check matrices under consideration in this paper.

We start by observing that many of the statements that were made in [36] about pseudocodewords can be extended to the setup of this paper. In particular, if some parity-check matrices $\tilde{\boldsymbol{H}}$ and $\boldsymbol{H}$ are such that $\mathsf{T}(\tilde{\boldsymbol{H}})$ is a graph cover of $\mathsf{T}(\boldsymbol{H})$, then a pseudocodeword of $\tilde{\boldsymbol{H}}$ can be "wrapped" to obtain a pseudocodeword of $\boldsymbol{H}$, as is formalized in the next lemma.

*Lemma 24:* Let the parity-check matrices $\tilde{\boldsymbol{H}}$ and $\boldsymbol{H}$ be such that $\mathsf{T}(\tilde{\boldsymbol{H}})$ is an $M$-fold graph cover of $\mathsf{T}(\boldsymbol{H})$. More precisely, let $\tilde{\boldsymbol{H}} = \sum_{\ell\in\mathcal{L}} \boldsymbol{H}_\ell \otimes \boldsymbol{P}_\ell$ for some set $\mathcal{L}$, for some collection of parity-check matrices $\{\boldsymbol{H}_\ell\}_{\ell\in\mathcal{L}}$ such that $\boldsymbol{H} = \sum_{\ell\in\mathcal{L}} \boldsymbol{H}_\ell$ (in $\mathbb{Z}$), and for some collection of $M \times M$ permutation matrices $\{\boldsymbol{P}_\ell\}_{\ell\in\mathcal{L}}$. Moreover, let $\mathcal{I}$ be the set of column indices of $\boldsymbol{H}$ and let $\mathcal{I} \times \mathcal{M}$ with $\mathcal{M} \triangleq \{0, 1, \ldots, M-1\}$ be the set of column indices of $\tilde{\boldsymbol{H}}$. With this, if $\tilde{\boldsymbol{\omega}} = (\tilde{\omega}_{(i,m)})_{(i,m)\in\mathcal{I}\times\mathcal{M}}$ is a pseudocodeword of $\tilde{\boldsymbol{H}}$, then $\boldsymbol{\omega} = (\omega_i)_{i\in\mathcal{I}}$ with

$$\omega_i \triangleq \frac{1}{M} \sum_{m\in\mathcal{M}} \tilde{\omega}_{(i,m)} \quad (\text{in } \mathbb{R}) \tag{17}$$

is a pseudocodeword of $\boldsymbol{H}$.

*Proof:* (Sketch.) There are different ways to verify this statement. One approach is to show that, based on the fact that $\tilde{\boldsymbol{\omega}}$ satisfies the inequalities that define the fundamental polytope of $\tilde{\boldsymbol{H}}$ [22], [23], [54], [55], $\boldsymbol{\omega}$ satisfies the inequalities that define the fundamental polytope of $\boldsymbol{H}$. (We omit the details.) Another approach is to use the fact that pseudocodewords with rational entries are given by suitable projections of codewords in graph covers [22], [23]. So, for every pseudocodeword $\tilde{\boldsymbol{\omega}}$ of $\tilde{\boldsymbol{H}}$ with rational entries, there is some graph cover of $\mathsf{T}(\tilde{\boldsymbol{H}})$ with a codeword in it, which, when projected down to $\mathsf{T}(\tilde{\boldsymbol{H}})$, gives $\tilde{\boldsymbol{\omega}}$. However, that graph cover of $\mathsf{T}(\tilde{\boldsymbol{H}})$ is also a graph cover of $\mathsf{T}(\boldsymbol{H})$,

and so this codeword, when projected down to $\mathsf{T}(\boldsymbol{H})$, gives $\boldsymbol{\omega}$ as defined in (17). (We omit the details; see [36] for a similar, but less general, result.) □

One can then proceed as in [36] and show that the AWGNC, the BSC, and the BEC pseudoweights [3], [22], [23], [54], [55], [58] of $\tilde{\boldsymbol{\omega}}$ will be at least as large as the corresponding pseudoweights of $\boldsymbol{\omega}$. As a corollary, the minimum AWGNC, BSC, and BEC pseudoweights of $\tilde{\boldsymbol{H}}$ are, respectively, at least as large as the corresponding minimum pseudoweights of $\boldsymbol{H}$. Similar results can also be obtained for the minimum Hamming distance.

Because the high-SNR behavior of linear programming decoding is dominated by the minimum pseudoweight of the relevant parity-check matrix, the high-SNR behavior of linear programming decoding of the code defined by $\tilde{\boldsymbol{H}}$ is at least as good as the high-SNR behavior of linear programming decoding of the code defined by $\boldsymbol{H}$.[24]

In general, because of the observations made in Section VI-A about the "breaking" of cycles and the fact that the active part of a pseudocodeword must contain at least one cycle, it follows that the unwrapping process is beneficial for the pseudocodeword properties of an unwrapped code, in the sense that many pseudocodewords that exist in the base code do not map to pseudocodewords in the unwrapped code. It is an intriguing challenge to better understand this process and its influence on the low-to-medium SNR behavior of linear programming and message-passing iterative decoders, in particular, to arrive at a better analytical explanation of the significant gains that are visible in the simulation plots that were shown in Section IV. To this end, the results of [46] and [48] with respect to some related code families (see the discussion in Section V) will be very helpful, since they indicate that some of the features of the fundamental polytope deserve further analysis.

### C. Cost of the "Convolutional Gain"

In this section, we investigate the cost of the convolutional gain by comparing several aspects of decoders for LDPC block and convolutional codes. In particular, we consider the computational complexity, hardware complexity, decoder memory requirements, and decoding delay. More details on the various comparisons described in this section can be found in [30], [40], and [41].

LDPC block code decoders and LDPC convolutional code decoders have the same computational complexity per decoded bit and per iteration since LDPC convolutional codes derived from LDPC block codes have the same node degrees (row and column weights) in their Tanner graph representations, which determines the number of computations required for message-passing decoding.

We adopt the notion of *processor size* to characterize the hardware complexity of implementing the decoder. A decoder's processor size is proportional to the maximum number of variable nodes that can participate in a common check equation. This is the block length $n$ for a block code, since any two variable nodes in a block can participate in the same check equation. For a convolutional code, this is the constraint length $\nu_{\mathrm{s}}$, since no two variable nodes that are more than $\nu_{\mathrm{s}}$ positions apart can participate in the same check equation. The constraint lengths of the LDPC convolutional codes derived from LDPC block codes of length $n$ satisfy $\nu_{\mathrm{s}} \leq n$. Therefore, the convolutional codes have a processor size less than or equal to that of the underlying block code.

On the other hand, the fully parallel pipeline decoding architecture penalizes LDPC convolutional codes in terms of decoder memory requirements (and decoding delay/latency) as a result of the $I$ iterations being multiplexed in space rather than in time. The pipeline decoder architecture of Fig. 1 consists of $I$ identical processors of size $\nu_{\mathrm{s}}$ performing $I$ decoding iterations simultaneously on independent sections of a decoding window containing $I$ constraint lengths of received symbols. This requires $I$ times more decoder memory elements than an LDPC block code decoder that employs a single processor of size $n = \nu_{\mathrm{s}}$ performing $I$ decoding iterations successively on the same block of received symbols. Therefore, the decoder memory requirements and the decoding delay of the pipeline decoder are proportional to $\nu_{\mathrm{s}} \cdot I$, whereas the block decoder's memory and delay requirements are only proportional to $n$. Another way of comparing the two types of codes, preferred by some researchers, is to equate the block length of a block code to the memory/delay requirements, rather than the processor size, of a convolutional code, i.e., to set $n = \nu_{\mathrm{s}} \cdot I$. In this case the block code, now having a block length many times larger than the constraint length of the convolutional code, will typically (depending on $I$) outperform the convolutional code, but at a cost of a much larger hardware processor. Finally, as noted in Section II, the parallel pipeline decoding architecture for LDPC convolutional codes can be replaced by a serial looping decoding architecture, resulting in fewer processors but a reduced throughput along with the same memory and delay requirements.

In summary, the convolutional gain achieved by LDPC convolutional codes derived from LDPC block codes comes at the expense of increased decoder memory requirements and decoding delays. Although this does not cause problems for some applications that are not delay-sensitive (e.g., deep-space communication), for other applications that are delay-sensitive (e.g., real-time voice/video transmission), design specifications may be met by deriving LDPC convolutional codes from shorter LDPC block codes, thus sacrificing some coding gain, but reducing memory and delay requirements, or by employing a reduced window size decoder, as suggested in the recent paper by Papaleo *et al.* [28], with a resulting reduction in the "convolutional gain."

## VII. Conclusion

In this paper, we showed that it is possible to connect two known techniques for deriving LDPC convolutional codes from LDPC block codes, namely the techniques due to Tanner and due to Jiménez-Feltström and Zigangirov. This connection was explained with the help of graph covers, which were also used as a tool to present a general approach for constructing interesting classes of LDPC convolutional codes. Because it is important to understand how the presented code construction methods can be used—and in particular combined—we then

[24]We neglect here the influence of the multiplicity of the minimum pseudoweight pseudocodewords.

discussed a variety of LDPC convolutional code constructions, along with their simulated performance results.

In the future, it will be worthwhile to extend the presented analytical results, in particular to obtain a better quantitative understanding of the low-to-medium SNR behavior of LDPC convolutional codes. In that respect, the insights in the papers by Lentmaier *et al.* [46], [47] and Kudekar *et al.* [48] on the behavior of related code families will be valuable guidelines for further investigation.

## ACKNOWLEDGMENT

The authors would like to thank C. Jones, M. Lentmaier, D. Mitchell, R. M. Tanner, and K. Zigangirov for their valuable discussions and comments. They also acknowledge the constructive comments made by the reviewers.

## REFERENCES

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

[2] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[3] N. Wiberg, "Codes and decoding on general graphs," Ph.D. dissertation, Dept. Electr. Eng., Linköping Univ., Linköping, Sweden, 1996.

[4] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 32, no. 18, pp. 1645–1646, Aug. 1996.

[5] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 150–159.

[6] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[7] S. Y. Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 657–670, Feb. 2001.

[8] T. J. Richardson and R. L. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[9] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.

[10] S. Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. L. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[11] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[12] P. Oswald and A. Shokrollahi, "Capacity-achieving sequences for the erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 12, pp. 3017–3028, Dec. 2002.

[13] S. Bates, D. Elliot, and R. Swamy, "Termination sequence generation circuits for low-density parity-check convolutional codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 53, no. 9, pp. 1909–1917, Sep. 2006.

[14] S. Bates, Z. Chen, and X. Dong, "Low-density parity check convolutional codes for Ethernet networks," in *Proc. IEEE Pacific Rim Conf. Commun. Comput. Signal Process.*, Victoria, BC, Canada, Aug. 2005, pp. 85–88.

[15] S. Bates, L. Gunthorpe, A. E. Pusane, Z. Chen, K. Sh. Zigangirov, and D. J. Costello, Jr., "Decoders for low-density parity-check convolutional codes with large memory," in *Proc. 12th NASA Symp. VLSI Design*, Coeur d'Alene, ID, Oct. 2005.

[16] R. M. Tanner, "Error-correcting coding system," U.S. Patent # 4 295 218, Oct. 1981.

[17] R. M. Tanner, "Convolutional codes from quasi-cyclic codes: A link between the theories of block and convolutional codes," Univ. California, Santa Cruz, CA, Tech Rep. UCSC-CRL-87-21, Nov. 1987.

[18] R. M. Tanner, D. Sridhara, A. Sridharan, T. E. Fuja, and D. J. Costello, Jr., "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[19] A. Jiménez-Feltström and K. Sh. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2181–2191, Sep. 1999.

[20] R. M. Tanner, "On quasi-cyclic repeat-accumulate codes," in *Proc. 37th Allerton Conf. Commun. Control Comput.*, Monticello, IL, Sep. 22–24, 1999, pp. 249–259.

[21] J. Thorpe, "Low-density parity-check (LDPC) codes constructed from protographs," JPL INP Progr. Rep., Aug. 2003, pp. 42–154.

[22] R. Koetter and P. O. Vontobel, "Graph covers and iterative decoding of finite-length codes," in *Proc. 3rd Int. Symp. Turbo Codes Related Topics*, Brest, France, Sep. 1–5, 2003, pp. 75–82.

[23] P. O. Vontobel and R. Koetter, "Graph-cover decoding and finite-length analysis of message-passing iterative decoding of LDPC codes," Dec. 2005 [Online]. Available: http://www.arxiv.org/abs/cs.IT/0512078

[24] A. E. Pusane, K. Sh. Zigangirov, and D. J. Costello, Jr., "Construction of irregular LDPC codes with fast encoding," in *Proc. IEEE Int. Conf. Commun.*, Istanbul, Turkey, Jun. 11–15, 2006, vol. 3, pp. 1160–1165.

[25] L. Zongwang, C. Lei, Z. Lingqi, S. Lin, and W. H. Fong, "Efficient encoding of quasi-cyclic low-density parity-check codes," *IEEE Trans. Commun.*, vol. 54, no. 1, pp. 71–81, Jan. 2006.

[26] T. J. Richardson and R. L. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638–656, Feb. 2001.

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

[28] M. Papaleo, A. R. Iyengar, P. H. Siegel, J. Wolf, and G. Corazza, "Windowed erasure decoding of LDPC convolutional codes," in *Proc. IEEE Inf. Theory Workshop*, Cairo, Egypt, Jan. 6–8, 2010, pp. 78–82.

[29] S. Bates, Z. Chen, L. Gunthorpe, A. E. Pusane, K. Sh. Zigangirov, and D. J. Costello, Jr., "A low-cost serial decoder architecture for low-density parity-check convolutional codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 55, no. 7, pp. 1967–1976, Aug. 2008.

[30] A. E. Pusane, A. Jiménez-Feltström, A. Sridharan, M. Lentmaier, K. Sh. Zigangirov, and D. J. Costello, Jr., "Implementation aspects of LDPC convolutional codes," *IEEE Trans. Commun.*, vol. 56, no. 7, pp. 1060–1069, Jul. 2008.

[31] Y. Levy and D. J. Costello, Jr., "An algebraic approach to constructing convolutional codes from quasi-cyclic codes," in *Coding and Quantization*, ser. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Providence, RI: AMS, 1993, vol. 14, pp. 189–198.

[32] M. Esmaeili, T. A. Gulliver, N. P. Secord, and S. A. Mahmoud, "A link between quasi-cyclic codes and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 44, no. 1, pp. 431–435, Jan. 1998.

[33] H. M. Stark and A. A. Terras, "Zeta functions of finite graphs and coverings," *Adv. Math.*, vol. 121, no. 1, pp. 124–165, Jul. 1996.

[34] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York: North-Holland, 1977.

[35] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[36] R. Smarandache, A. E. Pusane, P. O. Vontobel, and D. J. Costello, Jr., "Pseudo-codeword performance analysis of LDPC convolutional codes," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2577–2598, Jun. 2009.

[37] R. M. Tanner, "A [155, 64, 20] sparse graph (LDPC) code," presented at the IEEE Int. Symp. Inf. Theory, Sorrento, Italy, Jun. 2000.

[38] F. Hug, I. Bocharova, R. Johannesson, B. Kudryashov, and R. Satyukov, "New low-density parity-check codes with large girth based on hypergraphs," in *Proc. IEEE Int. Symp. Inf. Theory*, Austin, TX, Jun. 13–18, 2010, pp. 819–823.

[39] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, Jr., "On deriving good LDPC convolutional codes from QC LDPC block codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Nice, France, Jun. 24–29, 2007, pp. 1221–1225.

[40] D. J. Costello, Jr., A. E. Pusane, S. Bates, and K. Sh. Zigangirov, "A comparison between LDPC block and convolutional codes," in *Proc. Inf. Theory Appl. Workshop*, San Diego, CA, Feb. 6–10, 2006.

[41] D. J. Costello, Jr., A. E. Pusane, C. R. Jones, and D. Divsalar, "A comparison of ARA- and protograph-based LDPC block and convolutional codes," in *Proc. Inf. Theory Appl. Workshop*, San Diego, CA, Jan. 29–Feb. 2 2007, pp. 111–119.

[42] M. Lentmaier, D. V. Truhachev, and K. Sh. Zigangirov, "On the theory of low-density convolutional codes II," *Probl. Inf. Transm. (Problemy Peredachy Informatsii)*, vol. 37, pp. 288–306, Oct.–Dec. 2001.

[43] D. Divsalar, C. R. Jones, S. Dolinar, and J. Thorpe, "Protograph based LDPC codes with minimum distance linearly growing with block size," in *Proc. IEEE Global Telecommun. Conf.*, St. Louis, MO, Nov. 28—Dec. 5 2005, vol. 3, DOI: 10.1109/GLOCOM.2005.1577834.

[44] M. Ivkovic, S. K. Chilappagari, and B. Vasic, "Eliminating trapping sets in low-density parity-check codes by using Tanner graph covers," *IEEE Trans. Inf. Theory*, vol. 54, no. 8, pp. 3763–3768, Aug. 2008.

[45] The Consultative Committee for Space Data Systems (CCSDS), "Low density parity check codes for use in near-earth and deep space applications," Experimental Specification CCSDS 131.1-O-2, Sep. 2007 [Online]. Available: http://public.ccsds.org/publications/archive/131x1o2e2.pdf
[46] M. Lentmaier, G. P. Fettweis, K. Sh. Zigangirov, and D. J. Costello, Jr., "Approaching capacity with asymptotically regular LDPC codes," in *Proc. Inf. Theory Appl. Workshop*, San Diego, CA, Feb. 8–13, 2009, pp. 173–177.
[47] M. Lentmaier, D. G. M. Mitchell, G. P. Fettweis, and D. J. Costello, Jr., "Asymptotically regular LDPC codes with linear distance growth and thresholds close to capacity," in *Proc. Inf. Theory Appl. Workshop*, San Diego, CA, Jan. 31–Feb. 5 2010, DOI: 10.1109/ITA.2010.5454141.
[48] S. Kudekar, T. Richardson, and R. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, Feb. 2011.
[49] R. Smarandache and P. O. Vontobel, "Quasi-cyclic LDPC codes: Influence of proto- and Tanner-graph structure on minimum Hamming distance upper bounds," *IEEE Trans. Inf. Theory* Jan. 2009 [Online]. Available: http://arxiv.org/abs/0901.4129, submitted for publication
[50] B. K. Butler and P. H. Siegel, "On distance properties of quasi-cyclic protograph-based LDPC codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Austin, TX, Jun. 13–18, 2010, pp. 809–813.
[51] M. Lentmaier, A. Sridharan, D. J. Costello, Jr., and K. S. Zigangirov, "Iterative decoding threshold analysis for LDPC convolutional codes," *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 5274–5289, Oct. 2010.
[52] T. Tian, C. R. Jones, J. D. Villasenor, and R. D. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 8, pp. 1242–1247, Aug. 2004.
[53] A. Ramamoorthy and R. D. Wesel, "Analysis of an algorithm for irregular LDPC code construction," in *Proc. IEEE Int. Symp. Inf. Theory*, Chicago, IL, Jun. 27–Jul. 2 2004, DOI: 10.1109/ISIT.2004.1365107.
[54] J. Feldman, "Decoding error-correcting codes via linear programming," Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, 2003.
[55] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 954–972, Mar. 2005.
[56] N. Axvig, D. Dreher, K. Morrison, E. Psota, L. C. Perez, and J. L. Walker, "Analysis of connections between pseudocodewords," *IEEE Trans. Inf. Theory*, vol. 55, no. 9, pp. 4099–4107, Sep. 2009.
[57] C. A. Kelley and D. Sridhara, "Pseudocodewords of Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 53, no. 11, pp. 4013–4038, Nov. 2007.
[58] G. D. Forney, Jr., R. Koetter, F. Kschischang, and A. Reznik, "On the effective weights of pseudocodewords for codes defined on graphs with cycles," in *Codes, Systems, and Graphical Models*, B. Marcus and J. Rosenthal, Eds. New York: Springer-Verlag, 2001, vol. 123, pp. 101–112.

**Ali E. Pusane** (S'99–M'08) received the B.Sc. and M.Sc. degrees in electronics and communications engineering from Istanbul Technical University, Istanbul, Turkey, in 1999 and 2002, respectively, and the M.Sc. degree in electrical engineering, the M.Sc. degree in applied mathematics, and the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 2004, 2006, and 2008, respectively.

He was a Visiting Assistant Professor at the Department of Electrical Engineering, University of Notre Dame, during 2008–2009, after which he joined the Department of Electrical and Electronics Engineering, Bogazici University, Istanbul, Turkey, as an Assistant Professor. His research is in coding theory.

**Roxana Smarandache** (S'96–A'01–M'04) completed her undergraduate studies in mathematics at the University of Bucharest, Bucharest, Romania, in 1996, with a B.S. thesis on number theory, and received the Ph.D. degree in mathematics from the University of Notre Dame, Notre Dame, IN, in July 2001. Her dissertation was in coding theory, with the subject of algebraic convolutional codes.

She is an Associate Professor at the Department of Mathematics and Statistics, San Diego State University, San Diego, CA. During the academic year 1999–2000, she was for six months a visiting scholar at the Department of Communication Systems, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland. During the academic year 2005–2006, she was on a leave at the University of Notre Dame, on a visiting assistant professor position in the Department of Mathematics. During 2008–2009, she was on sabbatical leave in which she visited partly the Mathematics Department at the University of Zurich, Zurich, Switzerland, and partly the Mathematics and Electrical Engineering Departments at the University of Notre Dame. Her research topics are mainly related to coding theory. Her recent interests include low-density parity-check codes, iterative and linear programming decoding, and convolutional codes.

**Pascal O. Vontobel** (S'96–M'97) received the Diploma degree in electrical engineering, the Post-Diploma degree in information techniques, and the Ph.D. degree in electrical engineering from ETH Zurich, Switzerland, in 1997, 2002, and 2003, respectively.

From 1997 to 2002, he was a Research and Teaching Assistant at the Signal and Information Processing Laboratory, ETH Zurich. After being a Postdoctoral Research Associate at the University of Illinois at Urbana-Champaign, Urbana, the University of Wisconsin—Madison, Madison (Visiting Assistant Professor), and the Massachusetts Institute of Technology, Cambridge, he joined the Information Theory Research Group, Hewlett-Packard Laboratories, Palo Alto, CA, in summer 2006 as a Research Scientist. His research interests lie in information theory, communications, and signal processing.

Dr. Vontobel is an Associate Editor for the IEEE TRANSACTIONS ON INFORMATION THEORY and has been on the technical program committees of several international conferences. Recently, he has co-organized a BIRS workshop in Banff on "Applications of matroid theory and combinatorial optimization to information and coding theory" and a workshop at Tel Aviv University on "Linear programming and message-passing approaches to high-density parity-check codes and high-density graphical models." He has been three times a plenary speaker at international information and coding theory conferences and has been awarded the ETH medal for his Ph.D. dissertation.

**Daniel J. Costello, Jr.** (S'62–M'69–SM'78–F'86–LF'08) was born in Seattle, WA, on August 9, 1942. He received the B.S.E.E. degree from Seattle University, Seattle, WA, in 1964 and the M.S. and Ph.D. degrees in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 1966 and 1969, respectively.

He joined the faculty of the Illinois Institute of Technology, Chicago, in 1969, as an Assistant Professor of Electrical Engineering. He was promoted to Associate Professor in 1973, and to Full Professor in 1980. In 1985, he became Professor of Electrical Engineering at the University of Notre Dame, and from 1989 to 1998, served as Chair of the Department of Electrical Engineering. His research interests are in the area of digital communications, with special emphasis on error control coding and coded modulation. He has numerous technical publications in his field, and he coauthored a textbook entitled *Error Control Coding: Fundamentals and Applications* (Englewood Cliffs, NJ: Prentice-Hall, 1983), the second edition of which was published in 2004.

Dr. Costello was selected, in 1991, as one of 100 Seattle University alumni to receive the Centennial Alumni Award in recognition of alumni who have displayed outstanding service to others, exceptional leadership, or uncommon achievement. In 1999, he received a Humboldt Research Prize from the Alexander von Humboldt Foundation in Germany. In 2000, he was named the Leonard Bettex Professor of Electrical Engineering at Notre Dame. He was a member of the Information Theory Society Board of Governors (BoG) in 1983–1988, 1990–1995, 2005–2010, and in 1986 he served as President of the BoG. He has also served as Associate Editor for Communication Theory for the IEEE TRANSACTIONS ON COMMUNICATIONS, Associate Editor for Coding Techniques for the IEEE TRANSACTIONS ON INFORMATION THEORY, and Co-Chair of the IEEE International Symposia on Information Theory in Kobe, Japan (1988), Ulm, Germany (1997), and Chicago, IL (2004). In 2000, the IEEE Information Theory Society selected him as a recipient of a Third-Millennium Medal. He was corecipient of the 2009 IEEE Donald G. Fink Prize Paper Award, which recognizes an outstanding survey, review, or tutorial paper in any IEEE publication issued during the previous calendar year.