Encoding Spatially Coupled LDPC codes with Polynomial Generator Matrices

Massimo Battaglioni, *Member, IEEE*, Franco Chiaraluce, *Senior Member, IEEE*, Marco Baldi, *Senior Member, IEEE*

***Abstract*—Spatially coupled low-density parity-check (SC-LDPC) codes have recently attracted a lot of attention because of their optimal asymptotic performance. Time-invariant and periodically time-varying SC-LDPC codes are particularly interesting code families, since they have good finite-length performance and allow for a very compact representation. However, one of the key challenges to make SC-LDPC codes practical consists in finding efficient techniques for their encoding. In this paper, we propose a method to construct polynomial generator matrices for time-invariant and periodically time-varying SC-LDPC codes, based on their polynomial parity-check matrix. We show that it is always possible to find a polynomial generator matrix in quasi-standard form, where the systematic part consists of a diagonal matrix with identical entries, enabling efficient and non-catastrophic encoding. We also show that it is always possible to compute an equivalent rational function generator matrix in standard form that guarantees recursive, systematic (and therefore also non-catastrophic) encoding. Finally, to illustrate the method, we introduce a family of time-invariant SC-LDPC codes characterized by a binary parity-check matrix associated to a Tanner graph with girth larger than 4 and by good asymptotic and finite length performance. These codes naturally admit a quasi-standard polynomial generator matrix that, depending on the number of control symbols per period, either can be efficiently converted into polynomial standard form or remains in quasi-standard form but is relatively sparse. Both these features enable fast and efficient encoding.**

***Index Terms*—Generator matrix, LDPC codes, spatially coupled codes.**

I. Introduction

THE family of spatially coupled low-density parity-check (SC-LDPC) codes has been introduced in the late '90s, as LDPC convolutional codes [1]. Regular SC-LDPC codes have been shown to approach and even asymptotically achieve the channel capacity under belief propagation decoding over binary memoryless symmetric (BMS) channels [2], [3]. Also for irregular SC-LDPC code ensembles, as proven in [4], the belief propagation threshold saturates to the conjectured maximum a posteriori (MAP) threshold of the underlying irregular ensembles over BMS channels. SC-LDPC codes can be classified as time-invariant, periodically time-varying, or fully time-varying. Although, generally speaking, performance improves across these categories in the given order, having a simpler representation is a key advantage of time-invariant [5], [6] and periodically time-varying (with small period) SC-LDPC codes [7], [8]. The structure and regularity of time-invariant codes enables fully parallel decoding, with throughput in the range of hundreds of Gbit/s [9]. For all these reasons, SC-LDPC codes stand out as promising error-correction alternatives in optical communications [10] and next-generation wireless communications [11].

In digital communication systems and networks, channel coding must not only ensure reliable transmission but also operate with minimal delay to match stringent processing requirements. In modern communication scenarios, the importance of fast and efficient encoding is therefore amplified by the tight latency and complexity constraints of emerging technologies. Applications such as massive machine-type communications (mMTC) and ultra reliable low-latency communications (URLLC) also demand extremely fast and lightweight processing at the device level. In these contexts, the encoding stage can become a bottleneck, making structured and hardware-friendly implementations essential. Consequently, the design of encoding strategies that balance low complexity with good error-rate performance is gaining increasing relevance for next-generation networks. In this regard, an important yet relatively underexplored aspect in the literature concerns the encoding of SC-LDPC codes (see [12, Section VI-M]). In [13], two encoding methods – namely the syndrome former and partial syndrome former realizations – are proposed, both based on the parity-check matrix. These methods require the top-right portion of the transposed syndrome former matrix to be an identity matrix, which reduces the degrees of freedom in the code design. Moreover, these techniques are general, work well with sparse matrices, but do not take into account the potential representation and structural advantage of codes which are not fully time-varying. The recent paper [14] also describes encoding based on the partial syndrome former realization. Instead, very little attention has been paid to the generator matrix of SC-LDPC codes. Some examples are shown in [15] but, to the best of our knowledge, a formalized way to obtain a structured generator matrix is missing in the literature, and this motivates our work.

A. Our contribution

In this paper, we propose a method to construct the generator matrix (with entries being polynomials) of time-invariant

Massimo Battaglioni, Franco Chiaraluce, and Marco Baldi are with Dipartimento di Ingegneria dell'Informazione, Università Politecnica delle Marche, 60131 Ancona, Italy. Franco Chiaraluce is also with the Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), 43124 Parma, Italy (Corresponding author: Massimo Battaglioni. E-mail: m.battaglioni@univpm.it).
Their work was partially supported by the European Union - Next Generation EU under the Italian National Recovery and Resilience Plan (NRRP), Mission 4, Component 2, Investment 1.3, CUP J33C22002880001, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

Manuscript received XXX; revised XXX.

and periodically time-varying SC-LDPC codes, and show that these families of SC-LDPC codes can be easily encoded using their generator matrix, which is in *quasi-standard* form. By quasi-standard, we mean that the polynomial generator matrix contains a diagonal submatrix in which all diagonal elements are identical (and therefore not necessarily an identity submatrix). This structure preserves the systematic nature of the encoding while introducing a uniform "scaling" factor across all information bits. The method we propose uses minors of the polynomial parity-check matrix to construct the polynomial generator matrix of the code. The algebraic description of SC-LDPC codes is very useful in terms of the realization and classification of the encoder. More importantly, our method is hardware-friendly, since it is well known that polynomial multiplication and division can be efficiently implemented in practice. We also show how to always obtain the standard form of the generator matrix, which enables systematic (but recursive) encoding. Beyond being a desirable property for (n, k) error-correcting codes in general, since it allows the inclusion of k information symbols without additional overhead, systematic encoding is particularly crucial for convolutional codes. This is because it guarantees that encoding remains non-catastrophic, meaning that any input sequence with a finite number of nonzero symbols produces an output sequence with a finite number of nonzero symbols. On the one hand, without systematic encoding, certain encoder realizations may cause small errors to produce arbitrarily long error sequences, severely degrading error correction performance. On the other hand, non-systematic encoders are not necessarily catastrophic. We will show that, in the context of SC-LDPC codes, a polynomial generator matrix which is not in standard form (and therefore does not allow systematic encoding) might be sparse and non-catastrophic. In some applications, sparsity may be a more desirable feature than encoding systematicity. The computational complexity of the proposed methods is also assessed.

We also comment on the binary representation of the generator matrix. Moreover, we consider both conventional and tail-biting termination methods for SC-LDPC codes, showing that the latter approach yields remarkable advantages from the encoding standpoint.

Finally, to validate our theoretical analysis, we introduce a family of time-invariant SC-LDPC codes, called polynomial generator sequentially multiplied columns (PG-SMC) SC-LDPC codes, which serve as a concrete example illustrating the applicability of our approach. For the sake of readability, we refer to them simply as PG-SC codes. These codes are characterized by a polynomial generator matrix in quasi-standard form and, depending on the number of control symbols per period, the latter can either be efficiently converted into standard form or remains in quasi-standard form but is relatively sparse. Beyond encoding efficiency, PG-SC codes can be designed with girth larger than 4. Moreover, they exhibit good asymptotic protograph extrinsic information transfer (PEXIT) thresholds [16], and their finite-length error-rate performance is comparable to that of other codes with similar parameters.

B. Related works

Polynomial generator matrices based on minors were first constructed for quasi-cyclic LDPC (QC-LDPC) codes in [17]. This idea was later developed also in [18] for QC-LDPC codes, whereas [19] studies encoding of generalized QC-LDPC codes through the polynomial generator matrix. All these papers are based on the codeword construction technique proposed in [20, Lemma 6]. None of them deals with SC-LDPC codes, though. We already mentioned [13], where encoding of SC-LDPC codes based on the binary parity-check matrix is proposed. Our approach is different, since we focus on the generator, which results in a distinct encoding perspective and different implementation complexity compared to parity-check-based methods. Tanner *et al.* show some generator matrices of LDPC convolutional codes in [15], found through Gaussian elimination. Instead, we propose to construct generator matrices based on the permanent of minors of the polynomial parity-check matrix. In certain cases, this approach offers advantageous properties, such as the ability to work with polynomial generator matrices instead of rational function matrices, as well as possible sparsity, which is extremely rare to obtain through Gaussian elimination. Both characteristics are particularly beneficial for hardware implementations.

Recently, SC low-density generator matrix (SC-LDGM) codes have attracted growing interest [4], [21]–[23]. In this work, we take a first step toward developing a framework in which both the generator and parity-check matrices are sparse and structured which, besides the applications considered in the aforementioned papers (i.e., lossy compression and write-once memories), might also be of interest for the construction of Calderbank-Shor-Steane (CSS) quantum codes. Although these are undoubtedly active and stimulating areas of research, they fall beyond the scope of this paper.

C. Paper outline

The paper is organized as follows. In Section II, we introduce the notation and provide some background. In Section III, we propose a construction for the generator matrix of SC-LDPC codes. In Section IV, we provide some insight into the asymptotic complexity of the proposed approach. In Section V we introduce a family of time-invariant codes enabling efficient encoding. In Section VI, we assess the asymptotic and finite-length error rate performance of the proposed family of codes. In Section VII, we provide some concluding remarks.

II. Notation and preliminaries

Given two integers a and b, we denote by $[a, b]$ the set of integers $\{y | a \leq y \leq b\}$. The symbol $\otimes_N$ represents multiplication mod N.

We use upper case and lower case bold letters to denote matrices and vectors, respectively. For a vector $\mathbf{v}$ of length n, the i-th entry is indicated as v_i. For an $m \times n$ matrix $\mathbf{M}$, the entry at position (i, j) is indicated as $m_{i,j}$, the ith row as $\mathbf{M}_{i,:}$ and the j-th column as $\mathbf{M}_{:,j}$. Given an $m \times n$ matrix $\mathbf{M}$, and a set $S \subset [0, n-1]$, $\mathbf{M}_{\{S\}}$ is the submatrix of $\mathbf{M}$ formed by the columns of $\mathbf{M}$ with indexes in S. Given a set S, $|S|$ represents its cardinality. The all-zero matrix of size $m \times n$ is

denoted as $\mathbf{0}_{m\times n}$. Transposition is denoted with $^\top$. The $n\times n$ identity matrix is denoted as $\mathbf{I}_n$, whereas the $n\times n$ lower shift matrix, which is a binary matrix with ones only on the subdiagonal, is denoted as $\mathbf{L}_n$. Fraktur-style letters are used for rational functions and polynomials. The operator $\mathrm{perm}(\cdot)$ returns the permanent of its argument, that is,

$$\mathrm{perm}(\mathbf{Q}) \triangleq \sum_{\sigma\in S_n}\prod_{j=0}^{n-1} q_{j,\sigma(j)},$$

where $\mathbf{Q}$ is an $n\times n$ matrix and S_n is the symmetric group, i.e., the set of all $n!$ bijections from $[1,n]$ to $[1,n]$.

$\mathbb{Z}/N\mathbb{Z}$ denotes the ring of integers mod N. $\mathbb{F}_2[\mathrm{D}]$ denotes the ring of polynomials with coefficients in the Galois field $\mathbb{F}_2$. $\mathbb{F}_2[\mathrm{D}]/\langle \mathrm{D}^L-1\rangle$ denotes the ring of polynomials in $\mathbb{F}_2[\mathrm{D}]$ mod $\left(\mathrm{D}^L-1\right)$. $\mathbb{F}_2(\mathrm{D})$, instead, denotes the field of rational functions over $\mathbb{F}_2$. Moreover, $\mathbb{F}_2\langle\mathrm{D}\rangle$ denotes the field of Laurent series over $\mathbb{F}_2$. Finally, $\mathbb{F}_2^n$ denotes the n-dimensional vector space over $\mathbb{F}_2$.

The function returning the degree of a polynomial is denoted as $\deg(\cdot)$. The polynomial $\sum_{z=i}^{j}\mathrm{D}^z$, is denoted as $\mathfrak{U}_{j,i}(\mathrm{D})$. We define $\mathcal{W}(\cdot)$ as an operator that, given a polynomial as input, returns the number of its nonzero coefficients, which we call weight of the polynomial. When applied to a polynomial matrix (or vector), $\mathcal{W}(\cdot)$ acts entrywise, producing a matrix (or vector) whose elements correspond to the weights of the corresponding polynomial entries.

A. Background on LDPC and SC-LDPC codes

A binary linear block code $\mathcal{C}$ with length n is a k-dimensional linear subspace of $\mathbb{F}_2^n$. The code can be represented by a parity-check matrix $\mathbf{H}$, such that its null space is the code itself. In other words, $\mathcal{C}=\{\mathbf{c}\,|\,\mathbf{c}\mathbf{H}^\top=\mathbf{0}\}$, being $\mathbf{0}$ an all-zero vector, and $\mathbf{c}$ are codewords. The code is also represented by the generator matrix $\mathbf{G}$, whose rows form a basis for the code, i.e., the codewords can be obtained as $\mathbf{c}=\mathbf{u}\mathbf{G}$, where $\mathbf{u}\in\mathbb{F}_2^k$. The code rate is $R\triangleq\frac{k}{n}$. A code is said to be an LDPC code if its parity-check matrix is sparse.

The parity-check matrix can be written as

$$\mathbf{H}^\top_{[0,L-1]}=\begin{bmatrix}\mathbf{H}_0^\top(0) & \cdots & \mathbf{H}_{m_s}^\top(m_s) & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbf{H}_0^\top(L-1) & \cdots & \mathbf{H}_{m_s}^\top(L+m_s-1)\end{bmatrix}, \tag{1}$$

which describes a terminated SC-LDPC code with *frame length* $n=La$ and coupling length L. When the coupling length tends to infinity, i.e., for $L\to\infty$, the matrix represents an *unterminated* SC-LDPC code, defined as

$$\left\{\mathbf{c}\in\mathbb{F}_2^{La}\;\middle|\;\mathbf{c}\,\mathbf{H}^\top_{[0,L-1]}=\mathbf{0}\right\}.$$

Each block $\mathbf{H}_i(t)\in\mathbb{F}_2^{c\times a}$, $i=[0,m_s]$ is given as

$$\mathbf{H}_i(t)=\begin{bmatrix}h_i^{(0,0)}(t) & \dots & h_i^{(0,a-1)}(t)\\ \vdots & \ddots & \vdots\\ h_i^{(c-1,0)}(t) & \dots & h_i^{(c-1,a-1)}(t)\end{bmatrix}. \tag{2}$$

The design rate of an SC-LDPC code is $R\geq R_\infty-\frac{m_s c}{aL}$, where $R_\infty=\frac{a-c}{a}$ is its asymptotic rate.[1] An SC-LDPC code terminated in tail-biting fashion, or briefly a tail-biting SC-LDPC code, with coupling length $L>m_s$, is obtained by wrapping back the last $m_s c$ columns of (1) after L times instants.

We also define

$$\left[\mathbf{H}_{m_s}^\top(t)\,|\,\mathbf{H}_{m_s-1}^\top(t)\,|\,\cdots\,|\,\mathbf{H}_0^\top(t)\right]$$

as the t-th syndrome former matrix. The variable m_s is the syndrome former memory order (sometimes simply addressed as the *memory*) of the code and $\nu_s=(m_s+1)a$ is the syndrome former constraint length. If $\mathbf{H}_i(t)=\mathbf{H}_i(t+T)$ for a finite value of T, the corresponding code is said to be *periodically time-varying* with *period* T. When $T=1$, we say that the code is *time-invariant*. Time-invariant codes are characterized by a unique syndrome former matrix. If the above conditions are not met, the code is fully time-varying.

For time-invariant and periodically time-varying codes it is convenient to represent the following matrix

$$(\boldsymbol{\mathcal{H}}^{(t)})^\top=\begin{bmatrix}\mathbf{H}_t(t)\\ \mathbf{H}_{t+1}(t+1)\\ \vdots\\ \mathbf{H}_{t+m_s}(t+m_s)\end{bmatrix},$$

where $0\leq t\leq T-1$, in polynomial form. We get

$$\boldsymbol{\mathfrak{H}}^{(t)}(\mathrm{D})=\begin{bmatrix}\mathfrak{h}_{0,0}^{(t)}(\mathrm{D}) & \dots & \mathfrak{h}_{0,a-1}^{(t)}(\mathrm{D})\\ \vdots & \ddots & \vdots\\ \mathfrak{h}_{c-1,0}^{(t)}(\mathrm{D}) & \dots & \mathfrak{h}_{c-1,a-1}^{(t)}(\mathrm{D})\end{bmatrix},$$

where $\mathfrak{h}_{i,j}^{(t)}(\mathrm{D})\in\mathbb{F}_2[\mathrm{D}]$. To go from the binary form to the polynomial form, apply

$$\mathfrak{h}_{i,j}^{(t)}(\mathrm{D})=\sum_{z=0}^{m_s}h_z^{(i,j)}(t+z)\mathrm{D}^z.$$

Codewords, in polynomial form, can thus be expressed as $\mathbf{c}(\mathrm{D})$, and the code is

$$\{\mathbf{c}(\mathrm{D})\in\mathbb{F}_2\langle\mathrm{D}\rangle^a\,|\,\mathbf{c}(\mathrm{D})\boldsymbol{\mathfrak{H}}(\mathrm{D})^\top=\mathbf{0}\}, \tag{3}$$

where

$$\boldsymbol{\mathfrak{H}}(\mathrm{D})\triangleq\left[\;\boldsymbol{\mathfrak{H}}^{(0)}(\mathrm{D})\;\middle|\;\mathrm{D}\boldsymbol{\mathfrak{H}}^{(1)}(\mathrm{D})\;\middle|\;\dots\;\middle|\;\mathrm{D}^{T-1}\boldsymbol{\mathfrak{H}}^{(T-1)}(\mathrm{D})\;\right].$$

When dealing with time-invariant codes, we will drop the dependence on t.

We restrict our attention to codes for which

$$\mathcal{W}\big(\boldsymbol{\mathfrak{H}}^{(i)}(\mathrm{D})\big)=\mathcal{W}\big(\boldsymbol{\mathfrak{H}}^{(j)}(\mathrm{D})\big),\quad\forall\, i\neq j.$$

Then, for some t, we define the *base matrix* of the code as

$$\mathbf{B}\triangleq\mathcal{W}\big(\boldsymbol{\mathfrak{H}}^{(t)}(\mathrm{D})\big).$$

We then define an *ensemble of codes* $\mathcal{E}(\mathbf{B})$ as the collection of all codes characterized by the same base matrix $\mathbf{B}$. In the

[1]The inequality sign in the definition of rate for terminated SC-LDPC codes is due to the fact that the parity-check matrix may not have full rank.

time-varying case, if the above condition holds, without loss of generality we assume that $\mathbf{B} = \mathcal{W}\big(\boldsymbol{\mathfrak{H}}^{(0)}(\mathrm{D})\big)$.

For example, if $\boldsymbol{\mathfrak{H}}^{(t)}(\mathrm{D})$ contains only nonzero monomial entries $\forall t$, i.e.,

$$\mathbf{B}^{\mathrm{mon}} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & 1 & \ddots & 1 \\ \vdots & \ddots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{4}$$

it defines a *fully monomial code*. The set of all such codes defines the *fully monomial ensemble*, which has been extensively investigated in the literature (see, for instance, [24] for the block case and [15] for the convolutional case).

We anticipate here that the base matrix of the *polynomial generator sequentially multiplied columns (PG-SMC) ensemble* proposed in Section V of this paper is

$$\mathbf{B}^{\mathrm{PG}} = \left[\begin{array}{ccccc|cccc} 1 & 0 & \cdots & 0 & 1 & 1 & 1 & \cdots & 1 \\ 1 & 1 & 0 & \cdots & 0 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 1 & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & 1 & 1 & \cdots & 1 \\ 0 & \cdots & 0 & 1 & 1 & 1 & 1 & \cdots & 1 \end{array}\right]. \tag{5}$$

More details are given in the mentioned section.

Remark 1 If we apply the tail-biting termination to a time-invariant SC-LDPC code, we can see the corresponding parity-check matrix as that of a QC-LDPC code with block length aL, as defined in [25]. The well-known size-L circulant-block form of the parity-check matrix of this QC-LDPC code introduced in [26] is easily obtained by reordering the rows and columns of the parity-check matrix.

Analogously to the definition of $\boldsymbol{\mathfrak{H}}(\mathrm{D})$, one can define the polynomial generator matrix of a time-invariant code[2]

$$\boldsymbol{\mathfrak{G}}(\mathrm{D}) = \begin{bmatrix} \mathfrak{g}_{0,0}(\mathrm{D}) & \dots & \mathfrak{g}_{0,a-1}(\mathrm{D}) \\ \vdots & \ddots & \vdots \\ \mathfrak{g}_{a-c-1,0}(\mathrm{D}) & \dots & \mathfrak{g}_{a-c-1,a-1}(\mathrm{D}) \end{bmatrix}.$$

Definition 1 An $(a-c)\times a$ generator matrix $\boldsymbol{\mathfrak{G}}(\mathrm{D})$ is said to be in *standard form* if its rightmost $(a-c)\times(a-c)$ submatrix is the identity matrix. It is said to be in *quasi-standard form* if that submatrix is diagonal and all its elements are equal.

We loosely define the girth g of a code (or equivalently, of its binary parity-check matrix $\mathbf{H}$) as the girth – that is, the length of the shortest cycle – in the Tanner graph [27] corresponding to $\mathbf{H}$.

[2]Note that it is not necessary to define a polynomial generator matrix specifically for time-varying codes, as the codes we consider are either time-invariant or time-varying with a small period. In the latter case, we will show that the definition of a polynomial generator matrix for time-invariant codes remains sufficient.

III. Generator matrix of SC-LDPC codes

In this section, we first deal with the rational function representation of the generator matrix of time-invariant and periodically time-varying SC-LDPC codes; then, we shift to the binary representation, which is important to understand the role of termination for encoding.

A. Generator matrix of SC-LDPC codes based on minors

Let us consider the following result.

Lemma 1 [20, Lemma 6] Given a time-invariant SC-LDPC code described by $\boldsymbol{\mathfrak{H}}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]^{c\times a}$, there exist $\binom{a}{c+1}$ codewords denoted as

$$\boldsymbol{\mathfrak{m}}^{(S)}(\mathrm{D}) = \begin{bmatrix} \mathfrak{m}_0^{(S)}(\mathrm{D}) & \dots & \mathfrak{m}_{a-1}^{(S)}(\mathrm{D}) \end{bmatrix},$$

whose entries are

$$\mathfrak{m}_i^{(S)}(\mathrm{D}) = \begin{cases} \mathrm{perm}(\boldsymbol{\mathfrak{H}}_{\{S\setminus i\}}(\mathrm{D})) & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}, \tag{6}$$

where S is the s-th size-$(c+1)$ subset of $[0, a-1]$, $s \in [0, \binom{a}{c+1} - 1]$.

Note that, since we assume $\boldsymbol{\mathfrak{H}}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]^{c\times a}$, it follows by construction that $\mathfrak{m}^{(S)}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]^a$, $\forall S$. These polynomial vectors represent only a subset of all possible codewords, which, in general, belong to $\mathbb{F}_2\langle\mathrm{D}\rangle^a$, as previously noted.

Corollary 1 Given a polynomial codeword $\boldsymbol{\mathfrak{m}}(\mathrm{D})$ and a non-zero $\mathfrak{p}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]$ such that $\boldsymbol{\mathfrak{m}}(\mathrm{D}) = \hat{\boldsymbol{\mathfrak{m}}}(\mathrm{D})\mathfrak{p}(\mathrm{D})$, then also $\hat{\boldsymbol{\mathfrak{m}}}(\mathrm{D})$ is a codeword.

Proof: We have $\boldsymbol{\mathfrak{m}}(\mathrm{D})\boldsymbol{\mathfrak{H}}(\mathrm{D}) = \mathfrak{p}(\mathrm{D})\hat{\boldsymbol{\mathfrak{m}}}(\mathrm{D})\boldsymbol{\mathfrak{H}}(\mathrm{D}) = \mathbf{0}$. Since $\mathfrak{p}(\mathrm{D})$ is non-zero, $\hat{\boldsymbol{\mathfrak{m}}}(\mathrm{D})\boldsymbol{\mathfrak{H}}(\mathrm{D}) = \mathbf{0}$, which implies that $\hat{\boldsymbol{\mathfrak{m}}}(\mathrm{D})$ is also a codeword. ∎

Starting from Lemma 1, for time-invariant SC-LDPC codes we are able to construct a polynomial generator matrix as follows.

Theorem 1 Given a time-invariant SC-LDPC code $\mathcal{C}$ described by $\boldsymbol{\mathfrak{H}}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]^{c\times a}$, with $a > c$, then

$$\boldsymbol{\mathfrak{G}}(\mathrm{D}) = \begin{bmatrix} \boldsymbol{\mathfrak{m}}^{([0,c-1]\cup\{c\})}(\mathrm{D}) \\ \boldsymbol{\mathfrak{m}}^{([0,c-1]\cup\{c+1\})}(\mathrm{D}) \\ \vdots \\ \boldsymbol{\mathfrak{m}}^{([0,c-1]\cup\{a-1\})}(\mathrm{D}) \end{bmatrix} \in \mathbb{F}_2[\mathrm{D}]^{(a-c)\times a}, \tag{7}$$

is a (polynomial) generator matrix for $\mathcal{C}$.

Proof: We need to prove that $\boldsymbol{\mathfrak{G}}(\mathrm{D})\boldsymbol{\mathfrak{H}}(\mathrm{D})^\top = \mathbf{0}$. This is immediate, since according to Lemma 1 each of the rows of (7) is in $\mathbb{F}_2[\mathrm{D}]^a$ and is a codeword, therefore satisfying (3). ∎

The codewords to be included in (7) are chosen in such a way that the rightmost $(a-c)\times(a-c)$ submatrix of $\boldsymbol{\mathfrak{G}}(\mathrm{D})$ is diagonal. This potentially enables systematic encoding.

Remark 2 Suppose that for each row i of $\mathfrak{G}(\mathrm{D})$, there exists a polynomial $\mathfrak{p}_i(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]$ such that every nonzero entry $\mathfrak{g}_{i,j}(\mathrm{D})$ in that row can be written as

$$\mathfrak{g}_{i,j}(\mathrm{D}) = \mathfrak{p}_i(\mathrm{D})\hat{\mathfrak{g}}_{i,j}(\mathrm{D}).$$

Then, according to Corollary 1, the matrix formed by the polynomials $\hat{\mathfrak{g}}_{i,j}(\mathrm{D})$ is also a valid polynomial generator matrix.

Let us recall some fundamental results from classical convolutional coding theory, which establish whether a given generator matrix leads to catastrophic encoding or not. For brevity, we omit their proofs, as they are well known [28]–[30].

Definition 2 A convolutional encoder is said to be *catastrophic* if there exists an input sequence with infinite weight that produces a codeword of finite weight.

In other words, a catastrophic encoder allows error propagation: a small number of channel errors can cause the decoder to produce an output sequence that differs from the transmitted information sequence in infinitely many positions. For example, suppose an information sequence with infinite Hamming weight is mapped by the encoder to a codeword of finite Hamming weight, for instance, with only a few nonzero symbols. If this codeword is transmitted over a noisy channel, such as a binary symmetric channel, and the channel flips exactly the few nonzero bits, the received sequence may coincide with another valid codeword (precisely, the all-zero codeword). The decoder may thus interpret the received sequence as having originated from the all-zero input. As a result, it outputs an information sequence (in this case, the all-zero sequence) that differs from the transmitted one in infinitely many positions, even though the channel introduced only a small number of errors.

Proposition 1 [28, Chapter 2] A generator matrix $\mathfrak{G}(\mathrm{D})$ ensures *non-catastrophic encoding* if and only if the greatest common divisor of the permanents $\Delta_i(\mathrm{D})$ of all $\binom{a}{a-c}$ distinct $(a-c)\times(a-c)$ submatrices of $\mathfrak{G}(\mathrm{D})$ satisfies

$$\gcd\left\{\Delta_i(\mathrm{D}) \mid i = 1, 2, \ldots, \binom{a}{a-c}\right\} = \mathrm{D}^l \tag{8}$$

for some integer $l \geq 0$.

Corollary 2 [29, Section 11.1] A generator matrix in standard form guarantees non-catastrophic encoding.

Definition 3 A convolutional encoder is said to be *minimal* if it has the smallest possible number of memory elements among all encoders that generate the same code.

Corollary 3 [29, Section 11.1] A minimal non-systematic encoder is non-catastrophic.

The construction proposed in Theorem 1 is particularly useful, because it enables efficient encoding. The following corollary also holds.

Corollary 4 Given a time-invariant SC-LDPC code represented by $\mathfrak{G}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]^{(a-c)\times a}$ obtained as described in Theorem 1, suppose that

$$\mathfrak{g}_{i,j}(\mathrm{D}) = \mathfrak{g}_{0,c}(\mathrm{D})\hat{\mathfrak{g}}_{i,j}(\mathrm{D}),\ \forall\, 0 \leq i \leq a-c-1,\ 0 \leq j \leq c-1.$$

Then, $\mathfrak{G}(\mathrm{D})$ admits an equivalent generator matrix in standard form in $\mathbb{F}_2[\mathrm{D}]^{(a-c)\times a}$ as

$$[\hat{\mathfrak{G}}(\mathrm{D}) \,|\, \mathbf{I}_{a-c}].$$

Proof: We have established in Corollary 1 that it is possible to derive one valid codeword from another by "factoring out" the common polynomials in its entries. It can be easily shown that $\mathfrak{g}_{i_0,c+i_0}(\mathrm{D}) = \mathfrak{g}_{i_1,c+i_1}(\mathrm{D})$ for all $i_0 \neq i_1$. In fact, for any pair $0 \leq i_0, i_1 \leq a-c-1$, we have

$$\begin{aligned}\mathfrak{g}_{i_0,c+i_0}(\mathrm{D}) &= \mathfrak{m}_{c+i_0}^{[0,c-1]\cup\{c+i_0\}}(\mathrm{D}) = \mathrm{perm}(\mathfrak{H}_{\{[0,c-1]\}}(\mathrm{D})) \\ &= \mathfrak{m}_{c+i_1}^{[0,c-1]\cup\{c+i_1\}}(\mathrm{D}) = \mathfrak{g}_{i_1,c+i_1}(\mathrm{D}).\end{aligned}$$

Given that, by construction, the submatrix $\big(\mathfrak{G}(\mathrm{D})\big)_{[c,a-1]}$ is diagonal, we have just proved that

$$\mathfrak{G}(\mathrm{D}) = [\big(\mathfrak{G}(\mathrm{D})\big)_{[0,c-1]} |\, \mathbf{I}_{a-c}\, \mathfrak{g}_{0,c}(\mathrm{D})].$$

If we suppose that $\mathfrak{g}_{0,c}(\mathrm{D})$ is a common factor for every entry $\mathfrak{g}_{i,j}(\mathrm{D})$ in the first c columns, i.e.,

$$\mathfrak{g}_{i,j}(\mathrm{D}) = \mathfrak{g}_{0,c}(\mathrm{D})\hat{\mathfrak{g}}_{i,j}(\mathrm{D}),\ \forall\, 0 \leq i \leq a-c-1,\ 0 \leq j \leq c-1,$$

then also

$$[\hat{\mathfrak{G}}(\mathrm{D}) \,|\, \mathbf{I}_{a-c}]$$

is a valid generator matrix in standard form, where $\hat{\mathfrak{G}}(\mathrm{D})$ consists of the polynomials $\hat{\mathfrak{g}}_{i,j}(\mathrm{D})$. ■

Remark 3 The proof of Corollary 4 further reveals that all entries in the $(a-c)\times(a-c)$ rightmost diagonal submatrix of $\mathfrak{G}(\mathrm{D})$ are identical. When this condition holds, as anticipated in Section II, we refer to $\mathfrak{G}(\mathrm{D})$ as being in *quasi-standard* form.

A polynomial generator matrix in standard form, if attainable, would greatly simplify implementation, making it an important asset for practical applications. However, the hypotheses of Corollary 4 are often not met. For general polynomial parity-check matrices, in order to obtain systematic encoding, one needs to consider a rational function generator matrix.

Corollary 5 Given $\mathfrak{G}(\mathrm{D})$, obtained as described in Theorem 1 (and therefore in quasi-standard form), a rational function generator matrix in standard form for a time-invariant SC-LDPC code can always be obtained as

$$\mathfrak{G}^*(\mathrm{D}) = \mathfrak{G}(\mathrm{D})/\mathfrak{g}_{0,c}(\mathrm{D}) \in \mathbb{F}_2(\mathrm{D})^{(a-c)\times a}. \tag{9}$$

Proof: It easily follows from Corollary 1 and from the proof of Corollary 4. We obviously need to take into account

that, unlike $\mathbb{F}_2[\mathrm{D}]$, $\mathbb{F}_2(\mathrm{D})$ is a field, hence endowed with division. ∎

Corollary 5 is very important, since a rational function generator matrix as that in (9) always enables systematic (and thus non-catastrophic) encoding, whereas it is not guaranteed that Theorem 1 provides a generator matrix that enables non-catastrophic encoding.

Let us now generalize Theorem 1 to the case of periodically time-varying SC-LDPC codes.

Corollary 6 Given $\mathcal{C}$, a periodically time-varying SC-LDPC code of period T, described by

$$\{\boldsymbol{\mathcal{H}}^{(0)},\ldots,\boldsymbol{\mathcal{H}}^{(T-1)}\}\in\mathbb{F}_2^{a\times(m_s+1)c},$$

then we can find a valid $\boldsymbol{\mathfrak{H}}(\mathrm{D})\in\mathbb{F}_2[\mathrm{D}]^{Tc\times Ta}$, from which

$$\boldsymbol{\mathfrak{G}}(\mathrm{D})=\begin{bmatrix}\mathfrak{m}^{([0,Tc-1]\cup\{Tc\})}(\mathrm{D})\\ \mathfrak{m}^{([0,Tc-1]\cup\{Tc+1\})}(\mathrm{D})\\ \vdots\\ \mathfrak{m}^{([0,Tc-1]\cup\{Ta-1\})}(\mathrm{D})\end{bmatrix}\in\mathbb{F}_2[\mathrm{D}]^{T(a-c)\times Ta} \tag{10}$$

is a (polynomial) generator matrix for $\mathcal{C}$.

Proof: To prove the corollary, let us assume without loss of generality that $\mathbf{H}_{m_s}^{\top}(t)$ is not an all-zero matrix, $\forall t\in[0,T-1]$. We need to consider that a periodically time-varying code represented by T matrices $\boldsymbol{\mathcal{H}}^{(t)}$, $t\in[0,T-1]$, of size $a\times(m_s+1)c$, can be seen as a time-invariant code represented by the single $Ta\times(T+m_s)c$ matrix

$$\boldsymbol{\mathcal{H}}\triangleq\begin{bmatrix}\mathbf{H}_0(0) & \mathbf{0} & \mathbf{0}\\ \vdots & \ddots & \vdots\\ \mathbf{H}_{m_s}(m_s) & \ddots & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{H}_0(T-1)\\ \vdots & \ddots & \vdots\\ \mathbf{0} & \mathbf{0} & \mathbf{H}_{m_s}(T+m_s-1)\end{bmatrix}. \tag{11}$$

Then, the proof is the same as that of Theorem 1. ∎

The different points of view from which a periodically time-varying code can be seen were discussed, for example, in [31, Section II-A]. The properties of $\boldsymbol{\mathfrak{H}}(\mathrm{D})\in\mathbb{F}_2[\mathrm{D}]^{Tc\times Ta}$ are described in [8, Lemma 1]. Corollary 5 also naturally extends to the periodically time-varying case.

For better clarity, next we provide two toy examples.

Example 1 Let us consider the time-invariant SC-LDPC code with $c=2$ and $a=4$, having girth $g=6$, represented by

$$\boldsymbol{\mathfrak{H}}(\mathrm{D})=\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & \mathrm{D} & \mathrm{D}^8 & \mathrm{D}^{25}\end{bmatrix};$$

then, according to Theorem 1, we get

$$\begin{aligned}\mathfrak{m}^{([0,1]\cup\{2\})}(\mathrm{D})&=\begin{bmatrix}\mathrm{D}+\mathrm{D}^8 & 1+\mathrm{D}^8 & 1+\mathrm{D} & 0\end{bmatrix},\\ \mathfrak{m}^{([0,1]\cup\{3\})}(\mathrm{D})&=\begin{bmatrix}\mathrm{D}+\mathrm{D}^{25} & 1+\mathrm{D}^{25} & 0 & 1+\mathrm{D}\end{bmatrix},\end{aligned}$$

and thus the following polynomial generator matrix in quasi-standard form

$$\boldsymbol{\mathfrak{G}}(\mathrm{D})=\begin{bmatrix}\mathrm{D}+\mathrm{D}^8 & 1+\mathrm{D}^8 & 1+\mathrm{D} & 0\\ \mathrm{D}+\mathrm{D}^{25} & 1+\mathrm{D}^{25} & 0 & 1+\mathrm{D}\end{bmatrix}.$$

In this case, given the permanents Δ_i of all the 2×2 submatrices of $\boldsymbol{\mathfrak{G}}(\mathrm{D})$, we have $\gcd\{\Delta_i\}(\mathrm{D})=\mathrm{D}^2+1$, and therefore encoding would be catastrophic. However, by removing the common factors in each row, we get the standard form

$$\boldsymbol{\mathfrak{G}}^*(\mathrm{D})=\begin{bmatrix}\mathfrak{m}_0^{[0,2]}(\mathrm{D}) & (\mathrm{D}+1)^7 & 1 & 0\\ \mathfrak{m}_0^{[0,1]\cup\{3\}}(\mathrm{D}) & \mathfrak{m}_1^{[0,1]\cup\{3\}}(\mathrm{D}) & 0 & 1\end{bmatrix}$$

where

$$\begin{aligned}\mathfrak{m}_0^{[0,2]}(\mathrm{D})&=\mathrm{D}(\mathrm{D}^3+\mathrm{D}+1)\\ \mathfrak{m}_0^{[0,1]\cup\{3\}}(\mathrm{D})&=\mathrm{D}(\mathrm{D}+1)^7(\mathrm{D}^2+\mathrm{D}+1)^8\\ \mathfrak{m}_1^{[0,1]\cup\{3\}}(\mathrm{D})&=(\mathrm{D}^4+\mathrm{D}^3+\mathrm{D}^2+\mathrm{D}+1)\\ &\quad(\mathrm{D}^{20}+\mathrm{D}^{15}+\mathrm{D}^{10}+\mathrm{D}^5+1).\end{aligned}$$

It is important to emphasize that, in this simple example, $\boldsymbol{\mathfrak{G}}^*(\mathrm{D})$ is defined over $\mathbb{F}_2[\mathrm{D}]^{(a-c)\times a}$.

Example 2 Let us consider the periodically time-varying SC-LDPC code with $a=2$, $c=1$, period $T=2$ and girth $g=6$, represented by

$$\boldsymbol{\mathcal{H}}^{(0)}=\begin{bmatrix}1 & 0 & 1\\ 1 & 1 & 0\end{bmatrix},\boldsymbol{\mathcal{H}}^{(1)}=\begin{bmatrix}1 & 1 & 0\\ 1 & 0 & 1\end{bmatrix}.$$

We readily obtain, according to (11),

$$\boldsymbol{\mathcal{H}}^{\top}=\begin{bmatrix}1 & 1 & 0 & 0\\ 1 & 0 & 1 & 1\\ 0 & 1 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix},$$

which allows us to treat this periodically time-varying code as a time-invariant code with $a'=Ta=4$ and $c'=Tc=2$. We thus get

$$\boldsymbol{\mathfrak{H}}(\mathrm{D})=\begin{bmatrix}1 & 1+\mathrm{D} & 0 & \mathrm{D}\\ 1 & 0 & 1+\mathrm{D} & 1\end{bmatrix},$$

and then, from (10),

$$\boldsymbol{\mathfrak{G}}(\mathrm{D})=\begin{bmatrix}1+\mathrm{D}^2 & 1+\mathrm{D} & 1+\mathrm{D} & 0\\ 1+\mathrm{D} & 1+\mathrm{D} & 0 & 1+\mathrm{D}\end{bmatrix}.$$

Also in this case, $\gcd\{\Delta_i\}(\mathrm{D})=\mathrm{D}^2+1$. Applying Corollary 4, we get

$$\boldsymbol{\mathfrak{G}}^*(\mathrm{D})=\begin{bmatrix}1+\mathrm{D} & 1 & 1 & 0\\ 1 & 1 & 0 & 1\end{bmatrix}.$$

B. Tail-biting SC-LDPC codes

The generator matrix of a tail-biting SC-LDPC code with block length aL can be obtained as described in Theorem 1, but with an important difference: the entries of $\mathfrak{G}(\mathrm{D})$ should now be reduced mod $\left(\mathrm{D}^L+1\right)$. This is because, as recalled in Remark 2, a tail-biting SC-LDPC code can be seen as a quasi-cyclic code with length aL.

This way, the conditions in Corollary 4 relax as follows.

Corollary 7 Given a tail-biting time-invariant SC-LDPC code represented by $\mathfrak{G}(\mathrm{D})$ $\in \mathbb{F}_2[\mathrm{D}]^{(a-c)\times a}/\langle \mathrm{D}^L+1\rangle$, if

$$\gcd\left(\mathfrak{g}_{0,c}(\mathrm{D}), \mathrm{D}^L+1\right)=1, \tag{12}$$

then $\mathfrak{G}(\mathrm{D})$ has an equivalent standard form in $\mathbb{F}_2[\mathrm{D}]^{(a-c)\times a}/\langle \mathrm{D}^L+1\rangle$, which is $\mathfrak{G}^*(\mathrm{D})=\mathfrak{G}(\mathrm{D})\,\mathfrak{g}_{0,c}^{-1}(\mathrm{D})$.

Proof: Equation (12) provides the condition for the invertibility of $\mathfrak{g}_{0,c}(\mathrm{D}) \mod \left(\mathrm{D}^L+1\right)$. Indeed, if this polynomial is invertible in $\mathbb{F}_2[\mathrm{D}]^{(a-c)\times a}/\langle \mathrm{D}^L+1\rangle$, then we can obtain the standard form of $\mathfrak{G}(\mathrm{D})$ by multiplying it by the inverse of $\mathfrak{g}_{0,c}(\mathrm{D})$. ∎

All the theoretical and numerical results presented above are fully consistent with the scenario in which unterminated SC-LDPC codes as well as tail-biting SC-LDPC codes are considered. We dedicate the remainder of this section to the binary representation of the generator matrix, which is particularly useful in the case of terminated SC-LDPC codes.

C. Binary representation of the generator matrix of SC-LDPC codes

Given $\mathfrak{m}^{(S)}(\mathrm{D})$, a codeword obtained by exploiting Lemma 1, let us define

$$\beta^{(S)}=\max_{i\in[0,a-1]}\left\{\deg\left(\mathfrak{m}_i^{(S)}(\mathrm{D})\right)\right\}.$$

The binary representation of $\mathfrak{m}^{(S)}(\mathrm{D})$, simply denoted as $\mathbf{m}^{(S)}$, can be easily obtained as described in [30, Section 4.4.1], by multiplexing in a round-robin fashion. An example is shown next.

Example 3 Consider a polynomial codeword from Example 2, that is

$$\mathfrak{m}(\mathrm{D})=\begin{bmatrix}1+\mathrm{D}^2 & 1+\mathrm{D} & 1+\mathrm{D} & 0\end{bmatrix}.$$

We can extract the binary pattern by listing the coefficients of $\mathrm{D}^0=1$, $\mathrm{D}^1=\mathrm{D}$, and D^2 for each entry, ordered by increasing powers of D. Specifically:

- the coefficients of D^0 (constant terms) are $[1,\,1,\,1,\,0]$,
- the coefficients of D^1 are $[0,\,1,\,1,\,0]$,
- the coefficients of D^2 are $[1,\,0,\,0,\,0]$.

By concatenating these rows, we obtain the binary semi-infinite codeword

$$\mathbf{m}=[1,\,1,\,1,\,0\,|\,0,\,1,\,1,\,0\,|\,1,\,0,\,0,\,0\,|\,\ldots],$$

where all subsequent entries are zero.

Definition 4 Given a binary semi-infinite codeword $\mathbf{m}^{(S)}$, its right cyclic shift by ja positions is denoted by $\mathbf{m}^{(S)}_{\to j}$, with zeros inserted in the first $(j-1)a$ positions.

Corollary 8 The binary generator matrix corresponding to (7) can thus be constructed as

$$\mathbf{G}=\begin{bmatrix}\mathbf{m}_{\to 0}^{([0,c-1])\cup\{c\}}\\ \mathbf{m}_{\to 0}^{([0,c-1])\cup\{c+1\}}\\ \vdots\\ \mathbf{m}_{\to 0}^{([0,c-1])\cup\{a-1\}}\\ \mathbf{m}_{\to 1}^{([0,c-1])\cup\{c\}}\\ \mathbf{m}_{\to 1}^{([0,c-1])\cup\{c+1\}}\\ \vdots\\ \mathbf{m}_{\to 1}^{([0,c-1])\cup\{a-1\}}\\ \vdots\end{bmatrix} \tag{13}$$

$$=\begin{bmatrix}\mathbf{G}_0 & \mathbf{G}_1 & \ldots & \mathbf{G}_\eta & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \ldots & \mathbf{G}_\eta & \\ & & & \ddots & & \end{bmatrix}. \tag{14}$$

where $\mathbf{G}_i\in\mathbb{F}_2^{(a-c)\times a}$, $\forall i$, and $\eta=\max_S\{\beta^{(S)}\}$.

Proof: Easily follows from the polynomial to binary conversion for codewords. ∎

We define

$$\boldsymbol{\mathcal{G}}=\begin{bmatrix}\mathbf{G}_0 & \mathbf{G}_1 & \ldots & \mathbf{G}_\eta\end{bmatrix}. \tag{15}$$

Example 4 Let us consider the generator matrix in systematic form obtained in Example 2:

$$\mathfrak{G}^*(\mathrm{D})=\begin{bmatrix}1+\mathrm{D} & 1 & 1 & 0\\ 1 & & 1 & 0 & 1\end{bmatrix}.$$

So, we have

$$\begin{aligned}\mathbf{m}_{\to 0}^{([0,1])\cup\{2\}}&=[1\,1\,1\,0\,|\,1\,0\,0\,0\,|\,\ldots]\\ \mathbf{m}_{\to 1}^{([0,1])\cup\{2\}}&=[0\,0\,0\,0\,|\,1\,1\,1\,0\,|\,1\,0\,0\,0\,|\,\ldots]\\ &\ldots\\ \mathbf{m}_{\to 0}^{([0,1])\cup\{3\}}&=[1\,1\,0\,1\,|\,0\,0\,0\,0\,|\,\ldots]\\ \mathbf{m}_{\to 1}^{([0,1])\cup\{3\}}&=[0\,0\,0\,0\,|\,1\,1\,0\,1\,|\,0\,0\,0\,0\,|\,\ldots]\\ &\ldots,\end{aligned}$$

from which

$$\mathbf{G}^*=\begin{bmatrix}1&1&1&0&1&0&0&0&&&&&&\\ 1&1&0&1&0&0&0&0&&&&&&\\ &&&&1&1&1&0&1&0&0&0&&\\ &&&&1&1&0&1&0&0&0&0&&\\ &&&&&&&&1&1&1&0&1&0&0&0\\ &&&&&&&&1&1&0&1&0&0&0&0\\ &&&&&&&&&&&&&&&\ddots\end{bmatrix}.$$

Let us now consider a terminated time-invariant SC-LDPC code with length aL. We recall that its design rate is $R=1-\frac{\mathrm{rank}(\mathbf{H})}{aL}\geq R_\infty-\frac{m_s c}{aL}$. Therefore, the generator matrix should

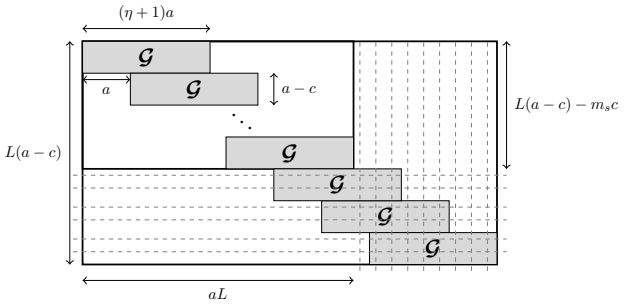

Fig. 1. Generator matrix of a terminated SC-LDPC code with a full rank parity-check matrix.

have size $[aL - \mathrm{rank}(\mathbf{H})] \times aL$. For the sake of simplicity, we first assume that the parity-check matrix has full rank. Then, $\mathbf{G} \in \mathbb{F}_2^{[L(a-c)-m_sc]\times La}$. Given the polynomial generator matrix $\mathfrak{G}(\mathrm{D})$, we can proceed as follows to easily compute the binary generator matrix:

1) Obtain the binary representation of $\mathfrak{G}(\mathrm{D})$, which is $\boldsymbol{\mathcal{G}}$;
2) Construct an $[L(a-c)] \times [a(L+\eta)]$ matrix, as in (14), but considering only L replicas of $\boldsymbol{\mathcal{G}}$;
3) Remove the last $a\eta$ columns and m_sc rows of the matrix constructed in Step 1.

A general scheme of this procedure is shown in Fig. 1. The resulting final binary generator matrix is the one included in the top-left rectangle.

If instead the rank of $\mathbf{H}$ is not full, we need to include

$$c(L+m_s) - \rho - \mathrm{rank}(\mathbf{H}) \tag{16}$$

additional linearly independent rows in $\mathbf{G}$, where in (16) we also take into account the fact that, by construction, the parity-check matrix may have ρ null rows. The additional linearly independent rows can be found through the transform theory approach proposed by Tanner in [32].

All the above reasoning is easily extended to the case of time-varying SC-LDPC codes of period T, by considering a $Tc \times Ta$ polynomial parity-check matrix, as discussed earlier.

The following example helps clarifying how the generator matrix of an SC-LDPC code with small constraint length is constructed.

Example 5 We apply the procedure described in this section to the code proposed in [33, Eq. (6)], which has particularly favorable properties in terms of leafless elementary trapping sets. These objects are known to negatively influence the error rate performance of SC-LDPC codes in the error-floor region. The polynomial parity-check matrix of this time invariant SC-LDPC code with $a = 5$, $c = 3$, and $m_s = 33$ is

$$\mathfrak{H}(\mathrm{D}) = \begin{bmatrix} 1 & \mathrm{D}^{30} & 1 & \mathrm{D}^{33} & 1 \\ \mathrm{D}^{26} & 1 & \mathrm{D}^{14} & 1 & \mathrm{D}^{5} \\ \mathrm{D}^{13} & \mathrm{D}^{3} & \mathrm{D}^{6} & 1 & \mathrm{D}^{28} \end{bmatrix}. \tag{17}$$

The entries of the quasi-standard polynomial generator matrix are as follows

$$\begin{aligned}
\mathfrak{g}_{0,0}(\mathrm{D}) &= 1 + \mathrm{D}^3 + \mathrm{D}^{36} + \mathrm{D}^{39} + \mathrm{D}^{44} + \mathrm{D}^{50}, \\
\mathfrak{g}_{0,1}(\mathrm{D}) &= \mathrm{D}^6 + \mathrm{D}^{13} + \mathrm{D}^{14} + \mathrm{D}^{26} + \mathrm{D}^{60} + \mathrm{D}^{65}, \\
\mathfrak{g}_{0,2}(\mathrm{D}) &= 1 + \mathrm{D}^3 + \mathrm{D}^{43} + \mathrm{D}^{46} + \mathrm{D}^{56} + \mathrm{D}^{62}, \\
\mathfrak{g}_{0,3}(\mathrm{D}) &= \mathrm{D}^6 + \mathrm{D}^{13} + \mathrm{D}^{17} + \mathrm{D}^{29} + \mathrm{D}^{57} + \mathrm{D}^{62}, \\
\mathfrak{g}_{0,4}(\mathrm{D}) &= 0, \\
\mathfrak{g}_{1,0}(\mathrm{D}) &= \mathrm{D}^6 + \mathrm{D}^8 + \mathrm{D}^{17} + \mathrm{D}^{28} + \mathrm{D}^{41} + \mathrm{D}^{72}, \\
\mathfrak{g}_{1,1}(\mathrm{D}) &= \mathrm{D}^{11} + \mathrm{D}^{18} + \mathrm{D}^{27} + \mathrm{D}^{32} + \mathrm{D}^{42} + \mathrm{D}^{54}, \\
\mathfrak{g}_{1,2}(\mathrm{D}) &= \mathrm{D}^8 + \mathrm{D}^{13} + \mathrm{D}^{28} + \mathrm{D}^{29} + \mathrm{D}^{48} + \mathrm{D}^{84}, \\
\mathfrak{g}_{1,3}(\mathrm{D}) &= 0, \\
\mathfrak{g}_{1,4}(\mathrm{D}) &= \mathrm{D}^6 + \mathrm{D}^{13} + \mathrm{D}^{17} + \mathrm{D}^{29} + \mathrm{D}^{57} + \mathrm{D}^{62}.
\end{aligned} \tag{18}$$

However, in this case we have that

$$\begin{aligned}
\gcd\{\Delta_i(\mathrm{D})\} = \mathrm{D}^{64} + \mathrm{D}^{62} + \mathrm{D}^{59} + \mathrm{D}^{57} + \mathrm{D}^{31} + \mathrm{D}^{29} \\
+ \mathrm{D}^{19} + \mathrm{D}^{17} + \mathrm{D}^{15} + \mathrm{D}^{13} + \mathrm{D}^{8} + \mathrm{D}^{6}.
\end{aligned}$$

yielding catastrophic encoding. Therefore, the entries of the corresponding rational function generator matrix in standard form are as follows

$$\begin{aligned}
\mathfrak{g}^*_{0,0}(\mathrm{D}) &= \frac{\mathfrak{U}_{38,33}(\mathrm{D}) + \mathfrak{U}_{24,22}(\mathrm{D}) + \mathfrak{U}_{13,11}(\mathrm{D}) + \mathrm{D}^2 + D + 1}{\mathfrak{U}_{50,46}(\mathrm{D}) + \mathfrak{U}_{39,35}(\mathrm{D}) + \mathfrak{U}_{28,24}(\mathrm{D}) + \mathfrak{U}_{12,6}(\mathrm{D})}, \\
\mathfrak{g}^*_{0,1}(\mathrm{D}) &= \frac{\mathrm{D}^{57} + \mathrm{D}^{55} + \mathfrak{U}_{52,19}(\mathrm{D}) + \mathrm{D}^{17} + \mathrm{D}^{15} + \mathrm{D}^{13} + \mathrm{D}^{11} + \mathrm{D}^9 + \mathrm{D}^7 + \mathrm{D}^6 + \mathrm{D}^4 + \mathrm{D}^2 + 1}{\mathrm{D}^{54} + \mathrm{D}^{52} + \mathfrak{U}_{50,22}(\mathrm{D}) + \mathrm{D}^{20} + \mathrm{D}^{18} + \mathrm{D}^{16} + \mathrm{D}^{14} + \mathrm{D}^{12} + \mathfrak{U}_{10,6}(\mathrm{D}) + \mathrm{D}^4 + \mathrm{D}^2 + 1}, \\
\mathfrak{g}^*_{0,2}(\mathrm{D}) &= \frac{\mathfrak{U}_{53,48}(\mathrm{D}) + \mathfrak{U}_{45,40}(\mathrm{D}) + \mathfrak{U}_{34,32}(\mathrm{D}) + \mathfrak{U}_{26,24}(\mathrm{D}) + \mathfrak{U}_{18,16}(\mathrm{D}) + \mathfrak{U}_{10,8}(\mathrm{D}) + \mathfrak{U}_{2,0}(\mathrm{D})}{\mathfrak{U}_{53,49}(\mathrm{D}) + \mathfrak{U}_{45,41}(\mathrm{D}) + \mathfrak{U}_{37,33}(\mathrm{D}) + \mathfrak{U}_{29,25}(\mathrm{D}) + \mathrm{D}^{21} + \mathfrak{U}_{16,14}(\mathrm{D}) + \mathfrak{U}_{12,6}(\mathrm{D})}, \\
\mathfrak{g}^*_{0,3}(\mathrm{D}) &= 1, \quad \mathfrak{g}^*_{0,4}(\mathrm{D}) = 0, \\
\mathfrak{g}^*_{1,0}(\mathrm{D}) &= \frac{\mathfrak{U}_{54,44}(\mathrm{D}) + \mathfrak{U}_{32,24}(\mathrm{D}) + \mathfrak{U}_{21,13}(\mathrm{D}) + \mathrm{D} + 1}{\mathfrak{U}_{44,40}(\mathrm{D}) + \mathfrak{U}_{33,29}(\mathrm{D}) + \mathfrak{U}_{22,18}(\mathrm{D}) + \mathfrak{U}_{6,0}(\mathrm{D})}, \\
\mathfrak{g}^*_{1,1}(\mathrm{D}) &= \frac{\mathrm{D}^{46} + \mathrm{D}^{44} + \mathrm{D}^{42} + \mathrm{D}^{40} + \mathrm{D}^{38} + \mathrm{D}^{36} + \mathrm{D}^{24} + \mathrm{D}^{22} + \mathfrak{U}_{20,11}(\mathrm{D}) + \mathrm{D}^9 + \mathrm{D}^7 + \mathrm{D}^5}{\mathrm{D}^{54} + \mathrm{D}^{52} + \mathfrak{U}_{50,22}(\mathrm{D}) + \mathrm{D}^{20} + \mathrm{D}^{18} + \mathrm{D}^{16} + \mathrm{D}^{14} + \mathrm{D}^{12} + \mathfrak{U}_{10,6}(\mathrm{D}) + \mathrm{D}^4 + \mathrm{D}^2 + 1}, \\
\mathfrak{g}^*_{1,2}(\mathrm{D}) &= \frac{\mathfrak{U}_{69,62}(\mathrm{D}) + \mathfrak{U}_{53,46}(\mathrm{D}) + \mathfrak{U}_{37,34}(\mathrm{D}) + \mathfrak{U}_{29,26}(\mathrm{D}) + \mathfrak{U}_{21,18}(\mathrm{D}) + \mathfrak{U}_{14,10}(\mathrm{D}) + \mathfrak{U}_{6,2}(\mathrm{D})}{\mathfrak{U}_{47,43}(\mathrm{D}) + \mathfrak{U}_{39,35}(\mathrm{D}) + \mathfrak{U}_{31,27}(\mathrm{D}) + \mathfrak{U}_{23,19}(\mathrm{D}) + \mathrm{D}^{15} + \mathrm{D}^{10} + \mathrm{D}^9 + \mathrm{D}^8 + \mathfrak{U}_{6,0}(\mathrm{D})}, \\
\mathfrak{g}^*_{1,3}(\mathrm{D}) &= 0, \quad \mathfrak{g}^*_{1,4}(\mathrm{D}) = 1,
\end{aligned} \tag{19}$$

where $\mathfrak{U}_{j,i}(\mathrm{D}) = \sum_{z=i}^{j} \mathrm{D}^z$, as defined in Section II.

Regarding termination, we consider $L = 300$ as in [33]. The resulting 999×1500 binary parity-check matrix actually contains 12 null rows, and its rank is 985. Therefore, according to (16), we need to insert 2 additional rows to the binary generator matrix to compensate the rank deficiency.

IV. Remarks on asymptotic computational complexity

Generally speaking, while finding a general generator or parity-check matrix for a code is a relatively simple task from a computational standpoint, finding a specific generator or parity-check matrix, possibly with some particular property, can be a computationally intensive task, to the point of becoming prohibitively expensive in some cases. For such reason, in this section we study the computational complexity of the methods we have introduced in the previous sections. For this purpose, let us consider separately the construction phase of the generator matrix and the encoding phase.

A. Construction of the polynomial generator matrix

The polynomial generator matrix incurs a one-time overhead cost during its construction. Precisely, it requires computing

$a-c$ codewords following Lemma 1. The cost of computing a single codeword corresponds to the cost of computing c permanents on a polynomial matrix. Let $M(m_s)$ denote the cost of multiplying two polynomials of degree m_s. For example, considering classical multiplication and fast Fourier transform (FFT)-based multiplication, we get [34]

$$M(m_s) = \begin{cases} O(m_s^2) & \text{(classical)}, \\ O(m_s \log m_s) & \text{(FFT-based)}. \end{cases}$$

Using Ryser's algorithm [35], the permanent of a $c \times c$ polynomial matrix (with entries of maximum degree m_s) can be computed in[3]

$$O\Big(2^c\, c\, M(cm_s)\Big)$$

operations. Since computing a single codeword requires the computation of c such permanents, and we need to compute $a-c$ many codewords, the total cost for obtaining the polynomial generator matrix is given by

$$C_{\text{total}} = (a-c)\, C_{\text{codeword}},$$

where

$$C_{\text{codeword}} = \begin{cases} O\Big(2^c\, c^3\, m_s^2\Big) & \text{(classical)}, \\ O\Big(2^c\, c^2\, m_s \log(cm_s)\Big) & \text{(FFT-based)}. \end{cases}$$

Thus, we have

$$C_{\text{total}} = \begin{cases} O\Big(2^c\, c^3\, m_s^2\, (a-c)\Big) & \text{(classical)}, \\ O\Big(2^c\, c^2\, m_s \log(cm_s)\, (a-c)\Big) & \text{(FFT-based)}. \end{cases}$$

We remark that the cost of computing the polynomial generator matrix $\mathfrak{G}(D)$ is independent of L, which represents a fundamental difference with methods that rely on the generator or parity-check matrix with scalar entries.

These cost estimates are conservative, as we assume the worst case in which every square submatrix of $\mathfrak{H}(\mathrm{D})$ contains at least one polynomial of degree m_s. Furthermore, optimizations can be applied by leveraging, for example, the fact that the computation of multiple permanents involves square submatrices $\mathfrak{H}(\mathrm{D})$ that differ by only a single column from a previously computed one. It is possible to exploit this structure to avoid redundant calculations: instead of recomputing each permanent from scratch, efficient update techniques can be applied to reuse previously computed values, leading to significant computational savings, which, however, are outside the scope of this paper and are not discussed further.

Remark 4 Note that applying FFT-based approaches to polynomial multiplication over $\mathbb{F}_2[\mathrm{D}]$ may not be the most convenient method in practice. Although FFT-based algorithms offer an asymptotic complexity of $O(m_s \log m_s)$, which is theoretically superior to the classical $O(m_s^2)$ methods, several practical issues can make them less attractive, especially for polynomials of moderate degree. In fact, standard FFT algorithms require the existence of sufficiently many roots of unity in the working field. However, $\mathbb{F}_2$ does not contain non-trivial roots of unity. This forces one to either work in an extension field of $\mathbb{F}_2$ (which introduces additional complexity) [36] or to employ FFT variants, such as additive FFT methods [37]. FFT-based methods come with significant overhead due to the need for forward and inverse transforms, as well as other setup costs. When the polynomial degrees are not extremely large, these constant factors can outweigh the asymptotic benefits, making simpler algorithms more efficient in practice. Moreover, since their coefficients are binary, polynomials over $\mathbb{F}_2$ can be represented as bit strings. This enables the use of highly optimized bit-level operations and hardware-accelerated instructions.

[3]Unlike the determinant, which is computable in polynomial time, evaluating the permanent is substantially more complex and currently requires exponential-time algorithms.

B. Encoding complexity

Let us study the complexity of different encoding methods.

1) Encoding with a rational function generator matrix in standard form: In general, encoding is performed as

$$\mathbf{c}(\mathrm{D}) = \mathbf{u}(\mathrm{D})\, \mathfrak{G}(\mathrm{D}),$$

where $\mathbf{c}(\mathrm{D}) \in \mathbb{F}_2\langle\mathrm{D}\rangle^a$, $\mathbf{u}(\mathrm{D}) \in \mathbb{F}_2\langle\mathrm{D}\rangle^{a-c}$, and

$$\mathfrak{G}(\mathrm{D}) = \Big[\, \hat{\mathfrak{G}}(\mathrm{D}) \;\Big|\; \mathbf{I}_{a-c} \,\Big] \in \mathbb{F}_2(\mathrm{D})^{(a-c)\times a}.$$

Encoding is systematic and recursive. Since $\mathfrak{G}(\mathrm{D})$ contains rational function entries, the encoding process is naturally separated in the two following parts. Let

$$\deg_{\max} = \max_j \deg\big(\mathfrak{u}_j(\mathrm{D})\big),$$

and assume that the numerator polynomials satisfy $\deg\big(\mathfrak{P}_{ij}(\mathrm{D})\big) \le d_P$, while the denominator polynomials have degree at most d_Q.

- Feedforward convolution: each parity polynomial $\mathfrak{p}_i(\mathrm{D})$ is given by

$$\mathfrak{p}_i(\mathrm{D}) = \sum_{j=1}^{a-c} \mathfrak{P}_{ij}(\mathrm{D})\, \mathfrak{u}_j(\mathrm{D}),$$

  where $\mathfrak{P}_{ij}(\mathrm{D})$ are the numerator polynomials of the rational functions in $\hat{\mathfrak{G}}(\mathrm{D})$. Using naive polynomial multiplication, each product $\mathfrak{P}_{ij}(\mathrm{D})\, \mathfrak{u}_j(\mathrm{D})$ involves polynomials of degrees at most d_P and $\deg_{\max}$, and therefore requires $O\big((d_P+1)\,(\deg_{\max}+1)\big)$ operations. Since there are $c(a-c)$ such products, the feedforward contribution to the encoding complexity is

$$O\Big(c(a-c)\,(d_P+1)\,(\deg_{\max}+1)\Big).$$

- Feedback (recurrence) solving: each parity stream must satisfy a recurrence induced by a denominator polynomial of degree at most d_Q. Applying such a recurrence to a polynomial of degree $\deg_{\max}$ requires $O\big((d_Q+1)\,(\deg_{\max}+1)\big)$ operations per parity stream. For all c parity polynomials, the feedback contribution is therefore

$$O\Big(c\,(d_Q+1)\,(\deg_{\max}+1)\Big).$$

Summing the two components, the overall encoding complexity is

$$O\Big(c\,(\deg_{\max}+1)\,\big[(a-c)\,(d_P+1)+(d_Q+1)\big]\Big).$$

Disregarding lower-order terms, this can be written more compactly as

$$O\Big(c\,\deg_{\max}\big[(a-c)\,d_P+d_Q\big]\Big),$$

which shows that the encoding complexity grows linearly with the degree of the information polynomials and depends on the degrees d_P and d_Q of the numerator and denominator polynomials in $\mathfrak{G}(\mathrm{D})$. In practice, additional structure or sparsity in $\mathfrak{G}(\mathrm{D})$ may further reduce this cost.

2) Encoding with a quasi-standard polynomial generator matrix: Let

$$\mathfrak{G}(\mathrm{D})=\Big[\,\hat{\mathfrak{G}}(\mathrm{D})\;\Big|\;\delta(\mathrm{D})\,\mathbf{I}_{a-c}\Big],$$

such that:

- $\hat{\mathfrak{G}}(\mathrm{D})\in\mathbb{F}_2[\mathrm{D}]^{(a-c)\times c}$, where each entry $\mathfrak{P}_{ij}(\mathrm{D})$ has degree at most d_P;
- $\delta(\mathrm{D})\in\mathbb{F}_2[\mathrm{D}]$ is a polynomial of degree d_δ.

The encoding process consists of the following steps:

- Parity computation (feedforward convolution): as above, the total contribution to the encoding complexity due to feedforward convolution is

$$O\Big(c(a-c)\,(d_P+1)(\deg_{\max}+1)\Big).$$

- Diagonal multiplication (systematic part): each systematic component is given by

$$\delta(\mathrm{D})\,\mathfrak{u}_j(\mathrm{D}).$$

This requires $O\big((d_\delta+1)(\deg_{\max}+1)\big)$ operations per symbol, yielding

$$O\Big((a-c)\,(d_\delta+1)(\deg_{\max}+1)\Big)$$

total operations.

Summing the two contributions, the overall encoding complexity is

$$O\Big((a-c)\,(\deg_{\max}+1)\big[\,c\,(d_P+1)+(d_\delta+1)\big]\Big).$$

Up to lower-order terms, this can be expressed as

$$O\Big((a-c)\;\deg_{\max}\big[\,c\,d_P+d_\delta\,\big]\Big),$$

which highlights that the encoding complexity grows linearly with the degree of the information polynomials and with d_P and d_δ. If the generator matrix is polynomial and in standard form (i.e., $\delta(\mathrm{D})=1$), this expression simplifies to

$$O\Big((a-c)\,c\;\deg_{\max}\;d_P\Big).$$

C. Comparison with Gaussian-elimination–based encoding

We now compare the proposed encoding approach with the conventional method based on Gaussian elimination. In traditional encoding, where the parity-check matrix of a terminated SC-LDPC code is treated as that of a block code, the generator matrix can be obtained via Gaussian elimination from the binary parity-check matrix, and the encoding is then carried out entirely in the binary domain, rather than in the polynomial domain.

In the Gaussian-elimination-based approach, a generator matrix $\mathbf{G}$ in standard form can be derived from (1) for finite L, after which the encoded codeword is obtained through the matrix-vector product $\mathbf{c}=\mathbf{u}\mathbf{G}$. Let $\mathbf{H}\in\mathbb{F}_2^{c(L+m_s)\times La}$ and assume that $\mathbf{H}$ has full row rank $r=c(L+m_s)$. The resulting generator matrix $\mathbf{G}\in\mathbb{F}_2^{k\times n}$ thus has

$$k=n-r=La-c(L+m_s)=L(a-c)-cm_s,\qquad n=La.$$

(i) Cost to obtain $\mathbf{G}$ *from* $\mathbf{H}$. Row-reduction of a rectangular $r\times n$ matrix over $\mathbb{F}_2$ with $r\le n$ requires

$$O\big(nr^2\big)\;=\;O\Big(La\big[c(L+m_s)\big]^2\Big)$$

operations in the worst case.

(ii) Cost to encode with $\mathbf{G}$. Given an information vector $\mathbf{u}\in\mathbb{F}_2^k$, with $\mathbf{G}=[\,\mathbf{I}_k\;\;\mathbf{P}\,]$ in standard form, one has $\mathbf{c}=[\,\mathbf{u}\;\;\mathbf{u}\mathbf{P}\,]$. Hence, the per-codeword cost is

$$O\big(k(n-k)\big)\;=\;O\Big(\big[L(a-c)-cm_s\big]c(L+m_s)\Big),$$

while storing $\mathbf{P}$ requires $O\big(k(n-k)\big)$ bits.

Therefore, overall, the total computational burden grows cubically with L during matrix formation[4] and quadratically with L during encoding.

In contrast, the proposed polynomial and rational-function encoders operate locally in time, maintaining only a small set of convolutional coefficients and achieving an overall complexity that scales linearly with L. This makes them far more efficient and memory-light, while the Gaussian-elimination approach remains practical only for very short code lengths.

The encoding complexity analysis presented in this section is deliberately formulated in a general way, rather than being tailored to specific constructions, like the one we propose in Section V. It applies to any SC-LDPC code family that can be represented through a relatively small polynomial parity-check matrix. The aim is to highlight how the encoder complexity scales with the structural parameters of the generator matrix, independently of the particular code design.

V. A FAMILY OF CODES WITH LOW ENCODING COMPLEXITY

Let us introduce a family of time-invariant codes characterized by a polynomial generator matrix in standard form, thus allowing non-recursive systematic encoding. Due to their features, we call them polynomial generator sequentially multiplied columns (PG-SMC) SC-LDPC codes (or, briefly,

[4]The banded structure of the binary parity-check matrix slightly lowers the exact operation count, as shown in Appendix A.

polynomial generator spatially coupled (PG-SC) codes). Their polynomial parity-check matrix is as follows

$$\mathfrak{H}(\mathrm{D}) = \begin{bmatrix} \mathfrak{H}_{\text{left}}(\mathrm{D}) & \mathfrak{H}_{\text{right}}(\mathrm{D}) \end{bmatrix}, \tag{20}$$

with

$$\mathfrak{H}_{\text{left}}(\mathrm{D}) = \mathbf{I}_c + \mathbf{L}_c + \mathrm{D}\,\mathbf{F}_c,$$

where $\mathbf{F}_c$ is a $c \times c$ matrix, with $c \geq 2$, such that $f_{0,c-1} = 1$ and all its other entries are 0. We remind that the $\mathbf{L}_c$ is the $c \times c$ lower shift matrix, which is a binary matrix with ones only on the subdiagonal. The matrix $\mathfrak{H}_{\text{right}}(\mathrm{D})$, instead, contains only monomials in $\mathbb{F}_2[\mathrm{D}]$, and is obtained by tweaking the design of QC-LDPC codes based on sequentially multiplied column (SMC) proposed in [38]. In particular, to design a time-invariant PG-SC code represented by a $c \times a$ polynomial parity-check matrix, given $N > a - c + 1$, let us start from the following $c \times (a-c+1)$ matrix (SMC-based construction) with entries in $\mathbb{Z}/N\mathbb{Z}$

$$\mathbf{P}^{\text{SMC}} = \left[\ \vec{0}\ \middle|\ \vec{P}_1\ \middle|\ \gamma_2 \otimes_N \vec{P}_1\ \middle|\ \ldots\ \middle|\ \gamma_{a-c} \otimes_N \vec{P}_1\ \right], \tag{21}$$

where $\vec{0}$ and $\vec{P}_1$ are column vectors of size c, with $\vec{0}$ being an all zero vector, in particular. More precisely, $\vec{P}_1$ is a vector with first (i.e., top most) entry equal to 0, second entry equal to 1, while its remaining entries are selected from $\{2, \ldots, N-1\}$ in an increasing order. The vectors $\gamma_j \otimes \vec{P}_1$ (with $j \in [2, a-c]$, $\gamma_j \in [2, N-1]$, $\gamma_j < \gamma_{j+1}$) are obtained from the base vector $\vec{P}_1$ through sequential multiplications. The value of N should be sufficiently large to guarantee that at least one matrix with the desired girth g exists. Too large values of N yield codes with very large constraint length. A very small number of attempts are usually needed to find a good trade-off between search speed and size of the code memory.

To obtain $\mathfrak{H}_{\text{right}}(\mathrm{D})$, we remove the first column of $\mathbf{P}^{\text{SMC}}$, which is an all-zero column, thus deriving $\hat{\mathbf{P}}^{\text{SMC}}$. Then, we substitute each entry $\hat{p}^{\text{SMC}}_{i,j}$ with the monomial $\mathrm{D}^{\hat{p}^{\text{SMC}}_{i,j}}$. The corresponding matrix results in $\mathfrak{H}_{\text{right}}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]^{c \times (a-c)}$.

The procedure for constructing $\mathfrak{H}(\mathrm{D})$ is summarized in Algorithm 1, where $p_{i,j}$ and $\hat{p}_{i,j}$ denote the entries of matrices $\mathbf{P}$ and $\hat{\mathbf{P}}$, respectively. Therefore, in summary, PG-SC codes are characterized by a polynomial parity-check matrix such that the right submatrix $\mathfrak{H}_{\text{right}}(\mathrm{D})$ is derived (with necessary modifications) from the SMC-based construction in [38], whereas the left submatrix $\mathfrak{H}_{\text{left}}(\mathrm{D})$ is designed independently to guarantee the existence of a polynomial generator matrix in standard form. It is interesting to observe that the PG-SC ensemble can be seen as the fully monomial ensemble, to which a special kind of masking [39] has been applied.

Algorithm 1 Construction of polynomial parity-check matrix $\mathfrak{H}(\mathrm{D})$ for efficient encoding

Input: c, a, N : modulo value, $g \leq 2c$: target girth for $\mathfrak{H}_{\text{right}}(\mathrm{D})$
Output: $\mathfrak{H}(\mathrm{D})$

$\gamma_1 \leftarrow 1$
$\mathfrak{H}_{\text{left}}(\mathrm{D}) \leftarrow \mathbf{I}_c + \mathbf{L}_c + \mathrm{D}\mathbf{F}_c$ ▷ Left part construction
Initialize zero matrix $\mathbf{P} = [\vec{0} \mid \vec{P}_1 \mid \vec{P}_2 \mid \cdots \mid \vec{P}_{a-c}]$
Set $p_{10} \leftarrow 0$, $p_{11} \leftarrow 1$
Initialize candidate set $\mathcal{S}_2 \leftarrow (c-2)$-combinations of $\{2, \ldots, N-1\}$
repeat
    Pick $(p_{21}, \ldots, p_{(c-1)1})$ from $\mathcal{S}_2$ such that $p_{21} < p_{31} < \cdots < p_{(c-1)1}$
    Remove chosen combination from $\mathcal{S}_2$
until no cycles of length $\leq g$ appear or $\mathcal{S}_2$ is empty
if $\mathcal{S}_2 = \emptyset$ **then**
    return failure
end if
for $j = 2$ to $a - c$ **do**
    Initialize candidate set $\mathcal{N}_j \leftarrow \{\gamma_{j-1} + 1, \ldots, N-1\}$
    repeat
        Select $\gamma_j \in \mathcal{N}_j$
        $\mathcal{N}_j \leftarrow \mathcal{N}_j \setminus \{\gamma_j\}$
        $\vec{P}_j \leftarrow \gamma_j \otimes_N \vec{P}_1$
    until no cycles of length $\leq g$ appear or $\mathcal{N}_j$ is empty
    if $\mathcal{N}_j = \emptyset$ **then**
        return failure
    end if
end for
Remove first column from $\mathbf{P}$ to get $\hat{\mathbf{P}}$
for each entry (i, j) in $\hat{\mathbf{P}}$ **do**
    Replace $\hat{p}_{i,j}$ with $\mathrm{D}^{\hat{p}_{i,j}}$
end for
$\mathfrak{H}_{\text{right}}(\mathrm{D}) \leftarrow \hat{\mathbf{P}}$
$\mathfrak{H}(\mathrm{D}) \leftarrow [\mathfrak{H}_{\text{left}}(\mathrm{D}) \mid \mathfrak{H}_{\text{right}}(\mathrm{D})]$
return $\mathfrak{H}(\mathrm{D})$

Let us provide the following result on the girth of the code represented by $[\mathfrak{H}_{\text{left}}(\mathrm{D}) \mid \mathfrak{H}_{\text{right}}(\mathrm{D})]$.

Theorem 2 Let

$$\mathfrak{H}(\mathrm{D}) = [\mathfrak{H}_{\text{left}}(\mathrm{D}) \mid \mathfrak{H}_{\text{right}}(\mathrm{D})]$$

be the parity-check matrix of the time-invariant SC-LDPC code constructed using Algorithm 1. If the girth of $\mathfrak{H}_{\text{right}}(\mathrm{D})$ is $g > 4$, then the girth g_{full} of $\mathfrak{H}(\mathrm{D})$ satisfies

$$6 \leq g_{\text{full}} \leq g.$$

Proof: By Algorithm 1, $\mathfrak{H}_{\text{right}}(\mathrm{D})$ has girth $g > 4$, whereas $\mathfrak{H}_{\text{left}}(\mathrm{D})$ corresponds to a binary matrix without cycles. Therefore, if a cycle of length 4 existed in $\mathfrak{H}(\mathrm{D})$, it would necessarily involve one column from $\mathfrak{H}_{\text{right}}(\mathrm{D})$ and one column from $\mathfrak{H}_{\text{left}}(\mathrm{D})$.

Let us distinguish two cases:

- if the index of the column in $\mathfrak{H}_{\text{left}}(\mathrm{D})$ is in $[0, c-2]$, then the existence of a cycle of length 4 would imply that

$p_{i,j} = 0$ for some $i > 0$ and some j, or that $p_{i,j} = p_{i+1,j}$, which are not possible by construction (see Lines 14–19 in Algorithm 1);
- if the index of the column in $\mathfrak{H}_{\text{left}}(\mathrm{D})$ is $c-1$, then the existence of a cycle of length 4 would imply that $p_{0,j} = p_{i,j} + 1$, for some $i > 0$ and some j, which is not possible since the entries of the exponent matrix $\mathbf{P}$ are nonnegative and $p_{0,j} = 0$ for all j.

Hence, $\mathfrak{H}(\mathrm{D})$ contains no cycle of length 4, and its girth satisfies $g_{\text{full}} \geq 6$. Moreover, since $\mathfrak{H}_{\text{right}}(\mathrm{D})$ is a submatrix of $\mathfrak{H}(\mathrm{D})$, every cycle of $\mathfrak{H}_{\text{right}}(\mathrm{D})$ is also a cycle of $\mathfrak{H}(\mathrm{D})$. Therefore, $g_{\text{full}} \leq g$, which completes the proof. ∎

Theorem 3 Any PG-SC code admits a polynomial generator matrix in standard form, if c is even.

Proof: We show next that, given any PG-SC code, all the entries in $\mathfrak{G}(\mathrm{D})$ (obtained by Theorem 1) can be written in the form

$$\mathfrak{g}_{i,j}(\mathrm{D}) = (\mathrm{D}+1)\,\hat{\mathfrak{g}}_{i,j}(\mathrm{D}),$$

where $\hat{\mathfrak{g}}_{i,j}(\mathrm{D}) \in \mathbb{F}_2[\mathrm{D}]$. Given this, also the matrix formed by the polynomials $\hat{\mathfrak{g}}_{i,j}(\mathrm{D})$ is a polynomial generator matrix for the code.

To prove the above, let us consider any $c \times c$ submatrix of $\mathfrak{H}(\mathrm{D})$, and let us denote it by $\mathfrak{M}(\mathrm{D})$. We observe that, since c is even by hypothesis, each column of $\mathfrak{M}(\mathrm{D})$ has an even number of nonzero entries, which are monomials. This implies that, if we substitute each entry of $\mathfrak{M}(\mathrm{D})$ with its coefficients vector, we obtain a matrix $\mathbf{M} \in \mathbb{F}_2$. Every column of $\mathbf{M}$ has a column-sum equal to 0 in $\mathbb{F}_2$.

Since each column of $\mathbf{M}$ has an even number of 1's, we have that every column $\mathbf{M}_j$ is orthogonal (over $\mathbb{F}_2$) to the "all-ones" vector $\mathbf{1}$. Hence each column $\mathbf{M}_j$ lies in the subspace

$$S \;=\; \{\,\mathbf{x} \in \mathbb{F}_2^c \mid \mathbf{x} \cdot \mathbf{1} = 0\}.$$

This subspace S has dimension $c-1$. Since there are c columns of $\mathbf{M}$ (all in an at-most-$(c-1)$-dimensional subspace), they are necessarily linearly dependent over $\mathbb{F}_2$. Consequently, $\mathbf{M}$ is singular over $\mathbb{F}_2$, which means $\det(\mathbf{M}) = 0$.

On the other hand, recall that the determinant in characteristic 2 coincides with the permanent. Hence, $\mathrm{perm}(\mathbf{M}) \equiv \det(\mathbf{M}) \equiv 0 \pmod 2$. Mapping back to polynomials, this implies that $\mathrm{perm}(\mathfrak{M}(\mathrm{D}))$ is either 0 or is a polynomial formed by an even number of monomials. Therefore, since $\mathrm{D}+1$ is a factor of any polynomial of even weight in $\mathbb{F}_2[\mathrm{D}]$, considering that each entry of $\mathfrak{G}(\mathrm{D})$ is the permanent of a $c \times c$ submatrix of the polynomial parity-check matrix, we can write

$$\mathfrak{g}_{i,j}(\mathrm{D}) = (\mathrm{D}+1)\,\hat{\mathfrak{g}}_{i,j}(\mathrm{D}).$$

Then, the generator matrix formed by entries $\hat{\mathfrak{g}}_{i,j}(\mathrm{D})$ is in standard form.

∎

Example 6 Let us consider the case with $a = 8$, $c = 4$, $N = 139$, and girth $g = 10$ for $\mathfrak{H}_{\text{right}}(\mathrm{D})$. By running Algorithm 1 we get

$$\mathfrak{H}(\mathrm{D}) = \begin{bmatrix} 1 & 0 & 0 & \mathrm{D} & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & \mathrm{D} & \mathrm{D}^3 & \mathrm{D}^{30} & \mathrm{D}^{105} \\ 0 & 1 & 1 & 0 & \mathrm{D}^{43} & \mathrm{D}^{129} & \mathrm{D}^{39} & \mathrm{D}^{67} \\ 0 & 0 & 1 & 1 & \mathrm{D}^{61} & \mathrm{D}^{44} & \mathrm{D}^{23} & \mathrm{D}^{11} \end{bmatrix}, \quad (22)$$

which indeed represents a time-invariant SC-LDPC code with asymptotic rate $1/2$ and overall girth 8. Using Theorem 1, its generator matrix can be written as

$$\mathfrak{G}(\mathrm{D}) = \left[\begin{array}{cccc|c} 1+\mathrm{D}^2+\mathrm{D}^{44}+\mathrm{D}^{62} & 1+\mathrm{D}+\mathrm{D}^{44}+\mathrm{D}^{62} & 1+\mathrm{D}+\mathrm{D}^{43}+\mathrm{D}^{62} & 1+\mathrm{D}+\mathrm{D}^{43}+\mathrm{D}^{61} & \\ 1+\mathrm{D}^4+\mathrm{D}^{45}+\mathrm{D}^{130} & 1+\mathrm{D}^3+\mathrm{D}^{45}+\mathrm{D}^{130} & 1+\mathrm{D}^3+\mathrm{D}^{45}+\mathrm{D}^{129} & 1+\mathrm{D}^3+\mathrm{D}^{44}+\mathrm{D}^{129} & \\ 1+\mathrm{D}^{24}+\mathrm{D}^{31}+\mathrm{D}^{40} & 1+\mathrm{D}^{24}+\mathrm{D}^{30}+\mathrm{D}^{40} & 1+\mathrm{D}^{24}+\mathrm{D}^{30}+\mathrm{D}^{39} & 1+\mathrm{D}^{23}+\mathrm{D}^{30}+\mathrm{D}^{39} & (\mathrm{D}+1)\,\mathbf{I}_4 \\ 1+\mathrm{D}^{12}+\mathrm{D}^{68}+\mathrm{D}^{106} & 1+\mathrm{D}^{12}+\mathrm{D}^{68}+\mathrm{D}^{105} & 1+\mathrm{D}^{12}+\mathrm{D}^{67}+\mathrm{D}^{105} & 1+\mathrm{D}^{11}+\mathrm{D}^{67}+\mathrm{D}^{105} & \end{array}\right]. \quad (23)$$

To obtain non-catastrophic encoding, it is possible to remove the common factor $\mathrm{D}+1$ from each entry of (23), thus renouncing the sparsity and obtaining

$$\mathfrak{G}^*(\mathrm{D}) = \left[\begin{array}{cccc|c} \mathfrak{U}_{61,44}(\mathrm{D})+\mathrm{D}+1 & \mathfrak{U}_{61,44}(\mathrm{D})+1 & \mathfrak{U}_{61,43}(\mathrm{D})+1 & \mathfrak{U}_{60,43}(\mathrm{D})+1 & \\ \mathfrak{U}_{129,45}(\mathrm{D})+\mathfrak{U}_{3,0}(\mathrm{D}) & \mathfrak{U}_{129,45}(\mathrm{D})+\mathfrak{U}_{2,0}(\mathrm{D}) & \mathfrak{U}_{128,45}(\mathrm{D})+\mathfrak{U}_{2,0}(\mathrm{D}) & \mathfrak{U}_{128,44}(\mathrm{D})+\mathfrak{U}_{2,0}(\mathrm{D}) & \\ \mathfrak{U}_{39,31}(\mathrm{D})+\mathfrak{U}_{23,0}(\mathrm{D}) & \mathfrak{U}_{39,30}(\mathrm{D})+\mathfrak{U}_{23,0}(\mathrm{D}) & \mathfrak{U}_{38,30}(\mathrm{D})+\mathfrak{U}_{23,0}(\mathrm{D}) & \mathfrak{U}_{38,30}(\mathrm{D})+\mathfrak{U}_{22,0}(\mathrm{D}) & \mathbf{I}_4 \\ \mathfrak{U}_{105,68}(\mathrm{D})+\mathfrak{U}_{11,0}(\mathrm{D}) & \mathfrak{U}_{104,68}(\mathrm{D})+\mathfrak{U}_{11,0}(\mathrm{D}) & \mathfrak{U}_{104,67}(\mathrm{D})+\mathfrak{U}_{11,0}(\mathrm{D}) & \mathfrak{U}_{104,67}(\mathrm{D})+\mathfrak{U}_{10,0}(\mathrm{D}) & \end{array}\right],$$

which is a standard form polynomial generator matrix enabling efficient, systematic, non-recursive encoding. We observe that the SC-LDPC code in this example has girth larger than 6.

In order to demonstrate the advantage in terms of complexity that can be obtained by performing the encoding procedure using the polynomial matrix described above, compared to the classic approach based on (generator or parity-check) matrices with scalar entries that describe the code, in Appendix A we continue to consider the code described in Example 6 and estimate and compare the corresponding encoding complexity.

When c is odd, unfortunately the entries of the standard form of $\mathfrak{G}(\mathrm{D})$ may be polynomials but also rational functions. Nevertheless, as shown next, they still have good asymptotic thresholds.

VI. Error rate performance

In this section, we first evaluate the asymptotic threshold of PG-SC codes and compare it with that of fully monomial codes. We then present finite-length error-rate performance curves, comparing them with the LDPC codes proposed in [40], [41].

A. Iterative decoding threshold analysis

Let us compute the PEXIT threshold [16] of some PG-SC codes on the additive white Gaussian noise (AWGN) channel, for many different values of c and a, and compare the PEXIT threshold of the fully monomial ensemble to that of the PG-SC ensemble. Before proceeding, we briefly recall the fundamental concepts underlying PEXIT analysis. A detailed and rigorous formulation can be found in [16] and is omitted here for brevity.

Let $J(\cdot)$ denote the function mapping the mean of Gaussian messages to extrinsic mutual information, i.e.,

$$J(\sigma) = 1 - \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(y-\sigma^2/2)^2}{2\sigma^2}\right)} \log_2\!\left(1+e^{-y}\right) dy.$$

We compute this function and its inverse using the approximations proposed in [42]. For such a purpose, let us denote with

- I_{ch} the channel mutual information,
- I_{Ec} the mutual information of the messages exchanged between check nodes and variable nodes,
- I_{Ev} the mutual information of the messages exchanged between variable nodes and check nodes,
- I_{APP} the a-posteriori mutual information.

Given a $c \times a$ base matrix $\mathbf{B}$, we can iteratively compute the evolution of the mutual information under belief propagation (BP) decoding, over the AWGN channel, as follows.

a) Initialization: For each $j = 0, \ldots, a-1$, set

$$I_{ch}(j) = J\big(\sigma_{ch,j}\big),$$

where $\sigma^2_{ch,j} = 8R\frac{E_b}{N_0}$.

b) Check-to-variable update: For $j = 0, \ldots, a-1$ and $i = 0, \ldots, c-1$, if $b_{i,j} \neq 0$ we compute

$$I_{Ec}(i,j) = 1 - J\left(\sqrt{\sum_{s \neq j} b_{i,s}\left[J^{-1}\big(1 - I_{Ev}(i,s)\big)\right]^2}\right),$$

and we set $I_{Ec}(i,j) = 0$ if $b_{i,j} = 0$.

c) Variable-to-check update: For $j = 0, \ldots, a-1$ and $i = 0, \ldots, c-1$, if $b_{i,j} \neq 0$ we compute

$$I_{Ev}(i,j) = J\left(\sqrt{\sum_{s \neq i} b_{s,j}\left[J^{-1}\big(I_{Ec}(s,j)\big)\right]^2 + \left[J^{-1}\big(I_{ch}(j)\big)\right]^2}\right).$$

and we set $I_{Ev}(i,j) = 0$ if $b_{i,j} = 0$.

d) APP-LLR mutual information: For each $j = 0, \ldots, a-1$, we compute

$$I^{(j)}_{\text{APP}} = J\left(\sqrt{\left[J^{-1}\big(I_{ch}(j)\big)\right]^2 + \sum_{s} b_{s,j}\left[J^{-1}\big(I_{Ec}(s,j)\big)\right]^2}\right).$$

The above steps are iterated until $I^{(j)}_{\text{APP}} \approx 1$ for all j, or until a maximum number of iterations is reached.

The PEXIT threshold on the AWGN channel, denoted as $(E_b/N_0)^*$, is defined as the smallest value of the signal-to-noise ratio (SNR) per information bit for which $I^{(j)}_{\text{APP}}$ converges to 1 for all values of j, for some prefixed number of iterations, i.e.,

$$(E_b/N_0)^* = \min\left\{\frac{E_b}{N_0} : I^{(j)}_{\text{APP}} \to 1,\ \forall j \in [0, a-1]\right\}. \quad (24)$$

Since the PEXIT analysis operates solely on the code base matrix, the parameters c and a are sufficient to determine the PEXIT threshold of the fully monomial and PG-SC ensembles. The base matrices defining these ensembles are given in (4) and (5), respectively. As an example, for $c = 6$ and $a = 8$, the base matrix of the fully monomial ensemble $\mathbf{B}^{\text{mon}}$ is simply the 6×8 all-ones matrix while the base matrix of the PG-SC ensemble, $\mathbf{B}^{\text{PG}}$, is

$$\mathbf{B}^{\text{PG}} = \left[\begin{array}{cccccc|cc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right].$$

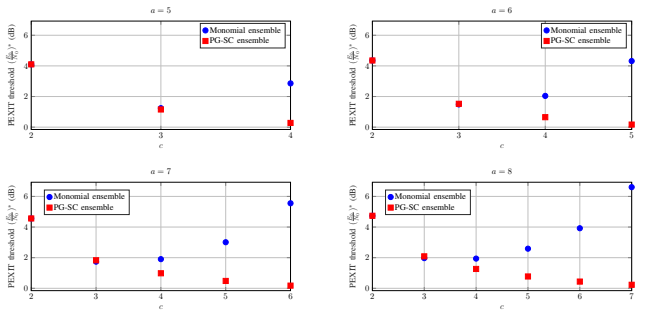

Fig. 2. Comparison of PEXIT thresholds for two code ensembles as a function of c for different values of $a > c$.

From the PEXIT thresholds comparison, reported in Fig. 2, we notice that the newly proposed ensemble has better asymptotic threshold in almost all the cases considered. This is due to the slight irregularity introduced in the polynomial parity-check matrix which, broadly speaking, usually yields a beneficial effect on the PEXIT threshold.

B. Finite-length error rate performance

To provide a deeper analysis of the performance of PG-SC codes in terms of error correction, we have also carried out Monte Carlo simulations of coded and quadrature phase-shift keying (QPSK)-modulated transmissions over the AWGN channel. The codes considered in these simulations were obtained by designing two PG-SC codes with asymptotic rate $R_\infty = \frac{2}{3}$, characterized by $c = 3$ and $c = 4$, and $m_s = 31$ and $m_s = 60$, respectively. These codes have been obtained using Algorithm 1, and applying the integer programming optimization model proposed in [6, Appendix A] to slightly reduce their memory order m_s. Then, these codes have been lifted with random permutation matrices of size 31 (for $c = 3$) and 23 (for $c = 4$); the resulting SC-LDPC codes are denoted as $\mathcal{C}_{\text{PG-SC-3}}$ and $\mathcal{C}_{\text{PG-SC-4}}$.[5] We compare their performance with that of two Advanced Television Systems Committee (ATSC) 3.0 LDPC block codes of the same rate (with block lengths 64 800 and 16 200, respectively). These two codes are denoted as $\mathcal{C}_{\text{ATSC-L}}$ and $\mathcal{C}_{\text{ATSC-S}}$. In addition, we benchmark them against a quasi-cyclic SC-LDPC code for broadcasting proposed in [41], which shares the same asymptotic rate, syndrome former constraint length, and frame length as our constructions.[6] This code is denoted as $\mathcal{C}_{\text{QC-B}}$. A comparison between the code parameters of $\mathcal{C}_{\text{QC-B}}$ and those

[5]The corresponding parity-check matrices in `alist` format are available at https://github.com/secomms/PG-SMC_SC-LDPC_codes.

[6]Minor deviations naturally arise due to the different design methods.

This article has been accepted for publication in IEEE Transactions on Information Theory. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIT.2026.3695390

TABLE I
PARAMETER SUMMARY FOR THE THREE SC-LDPC CODES ANALYZED NUMERICALLY

Code	Frame length n	Constraint length ν_s	Asymptotic rate R_∞	Actual rate R
$\mathcal{C}_{\text{PG-SC-3}}$	92 232	16 128	$\frac{2}{3}$	0.6108
$\mathcal{C}_{\text{PG-SC-4}}$	92 184	16 836	$\frac{2}{3}$	0.6180
$\mathcal{C}_{\text{QC-B}}$	92 160	16 128	$\frac{2}{3}$	0.65

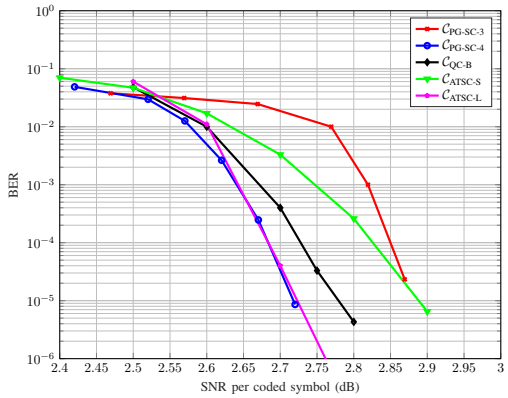

Fig. 3. BER vs SNR per coded symbol of different block and SC-LDPC codes.

of $\mathcal{C}_{\text{PG-SC-3}}$ and $\mathcal{C}_{\text{PG-SC-4}}$ is shown in Table I. We note that the considered PG-SC codes have a slightly larger rate loss than $\mathcal{C}_{\text{QC-B}}$.

The results of the Monte Carlo simulations, for all the considered codes, are shown in Fig. 3. We have employed a conventional flooding-schedule BP decoder for all the considered codes, performing a maximum of 100 decoding iterations for the block codes and a maximum number of iterations, equal to 250, for the SC-LDPC codes, as in [41]. Simulation results are reported in terms of bit error rate (BER) versus SNR per coded symbol, and each point in the curves has been obtained after observing 100 decoding errors over the entire frame. We note that, despite our codes have not undergone any error performance-driven optimization, unlike the other designs (which rely on extensive structural refinement), their behavior remains consistent with what one would expect from their structural properties. In particular, the code with $c = 3$ exhibits a relatively low average column weight, which naturally leads to a weaker finite-length performance. Conversely, the code with $c = 4$ shows a markedly improved behavior: it outperforms the short ATSC 3.0 block code and achieves performance comparable to that of the long ATSC 3.0 block code and of the quasi-cyclic SC-LDPC code, if we consider the slightly larger rate loss of the proposed code.

VII. CONCLUSION AND FUTURE WORKS

We have proposed an efficient method to encode SC-LDPC codes based on their generator matrix. Thanks to the compact representation of time-invariant and periodically time-varying codes with a small period, the proposed method is particularly effective for encoding. Notably, the cost of computing the polynomial generator matrix $\mathfrak{G}(D)$ with the proposed method is independent of the whole frame length L, in contrast with approaches that rely on matrices with scalar entries. Additionally, we have introduced a novel family of SC-LDPC codes, called PG-SMC SC-LDPC codes, which exhibit two important properties: their generator matrix is both polynomial and in standard form. The standard form ensures systematic encoding, while the polynomial structure enables efficient encoding, making these codes interesting for practical applications. As we have shown through some practical examples, these codes also show encouraging asymptotic behavior and finite-length performance.

A natural follow-up to this work is to exploit the knowledge of the polynomial generator matrix to estimate or bound the free distance of time-invariant and periodically time-varying SC-LDPC codes (as done in [17], [18] for QC-LDPC codes, for example).

REFERENCES

[1] A. Jiménez Felström and K. S. 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.
[2] M. Lentmaier, G. Fettweis, K. S. Zigangirov, and D. J. Costello, "Approaching capacity with asymptotically regular LDPC codes," in *Proc. IEEE Inf. Theory Applications Workshop (ITA 2009)*, San Diego, USA, 8-13 Feb. 2009, pp. 173–177.
[3] K. Kudekar, T. Richardson, and R. L. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 7761–7813, Dec. 2013.
[4] S. Kumar, A. J. Young, N. Macris, and H. D. Pfister, "Threshold saturation for spatially coupled LDPC and LDGM codes on BMS channels," *IEEE Trans. Inf. Theory*, vol. 60, no. 12, pp. 7389–7415, Dec. 2014.
[5] D. Achlioptas, H. Hassani, W. Liu, and R. Urbanke, "Time-invariant LDPC convolutional codes," in *Proc. IEEE International Symposium on Information Theory (ISIT 2017)*, Aachen, Germany, 25-30 Jun. 2017, pp. 366–370.
[6] M. Battaglioni, A. Tasdighi, G. Cancellieri, F. Chiaraluce, and M. Baldi, "Design and analysis of time-invariant SC-LDPC convolutional codes with small constraint length," *IEEE Trans. Commun.*, vol. 66, no. 3, pp. 918–931, Mar. 2018.
[7] M. Battaglioni, F. Chiaraluce, M. Baldi, and M. Lentmaier, "Girth analysis and design of periodically time-varying SC-LDPC codes," *IEEE Trans. Inf. Theory*, vol. 67, no. 4, pp. 2217–2235, Apr. 2021.
[8] M. Battaglioni, M. Baldi, and F. Chiaraluce, "Bounds on the free distance of periodically time-varying SC-LDPC codes," *IEEE Trans. Inf. Theory*, vol. 70, no. 4, pp. 2419–2429, Apr. 2024.
[9] M. Herrmann, N. Wehn, M. Thalmaier, M. Fehrenz, T. Lehnigk-Emden, and M. Alles, "A 336 Gbit/s full-parallel window decoder for spatially coupled LDPC codes," in *Proc. 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, Porto, Portugal, 8-11 Jun. 2021, pp. 508–513.
[10] L. Schmalen, D. Suikat, D. Rösener, V. Aref, A. Leven, and S. Ten Brink, "Spatially coupled codes and optical fiber communications: An ideal match?" in *Proc. 2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Stockholm, Sweden, 28 Jun. - 1 Jul. 2015, pp. 460–464.
[11] H. Zhang and W. Tong, "Channel coding for 6G extreme connectivity—requirements, capabilities, and fundamental tradeoffs," *IEEE BITS the Information Theory Magazine*, vol. 3, no. 1, pp. 54–66, Mar. 2023.

[12] M. Rowshan, M. Qiu, Y. Xie, X. Gu, and J. Yuan, "Channel coding toward 6G: Technical overview and outlook," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 2585–2685, Apr. 2024.

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

[14] S. Miao, C. Kestel, L. Johannsen, M. Geiselhart, L. Schmalen, A. Balatsoukas-Stimming, G. Liva, N. Wehn, and S. Ten Brink, "Trends in channel coding for 6G," *Proceedings of the IEEE*, vol. 112, no. 7, pp. 653–675, Jul. 2024.

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

[16] G. Liva and M. Chiani, "Protograph LDPC codes design based on EXIT analysis," in *Proc. IEEE Global Telecommunications Conference (Globecom 2007)*, Washington, DC, USA, 26-30 Nov. 2007, pp. 3250–3254.

[17] M. Battaglioni, P. Santini, M. Baldi, and G. Cancellieri, "Obtaining structured generator matrices for QC-LDPC codes," in *Proc. AEIT International Annual Conference (AEIT 2019)*, Florence, Italy, 18-20 Sep. 2019.

[18] R. Smarandache, A. Gómez-Fonseca, and D. G. M. Mitchell, "Using minors to construct generator matrices for quasi-cyclic LDPC codes," in *Proc. IEEE International Symposium on Information Theory (ISIT 2022)*, Espoo, Finland, 26 Jun. - 01 Jul. 2022, pp. 548–553.

[19] R. Smarandache, A. G. Fonseca, and D. G. M. Mitchell, "Generalized quasi-cyclic LDPC codes: Design and efficient encoding," in *Proc. IEEE International Symposium on Information Theory (ISIT 2024)*, Athens, Greece, 7-12 Jul. 2024, pp. 434–439.

[20] 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*, vol. 58, no. 2, pp. 585–607, Feb. 2012.

[21] V. Aref, N. Macris, and M. Vuffray, "Approaching the rate-distortion limit with spatial coupling, belief propagation, and decimation," *IEEE Trans. Inf. Theory*, vol. 61, no. 7, pp. 3954–3979, Jul. 2015.

[22] A. Golmohammadi, D. G. M. Mitchell, J. Kliewer, and D. J. Costello, "Encoding of spatially coupled LDGM codes for lossy source compression," *IEEE Trans. Commun.*, vol. 66, no. 11, pp. 5691–5703, Nov. 2018.

[23] S. Cai, W. Lin, X. Yao, B. Wei, and X. Ma, "Systematic convolutional low density generator matrix code," *IEEE Trans. Inf. Theory*, vol. 67, no. 6, pp. 3752–3764, Jun. 2021.

[24] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[25] R. Townsend and E. J. Weldon, "Self-orthogonal quasi-cyclic codes," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 2, pp. 183–195, Apr. 1967.

[26] M. Karlin, "New binary coding results by circulants," *IEEE Trans. Inf. Theory*, vol. IT-15, no. 1, pp. 81–92, Jan. 1969.

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

[28] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*. Wiley-IEEE Press, 1999.

[29] S. Lin and D. J. Costello, *Error Control Coding*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, Inc., 2004.

[30] W. E. Ryan and S. Lin, *Channel Codes - Classical and Modern*. Cambridge University, 2009.

[31] A. E. Pusane, R. Smarandache, P. O. Vontobel, and D. J. Costello, "Deriving good LDPC convolutional codes from LDPC block codes," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 835–857, Feb. 2011.

[32] R. Tanner, "A transform theory for a class of group-invariant codes," *IEEE Trans. Inf. Theory*, vol. 34, no. 4, pp. 725–775, Jul. 1988.

[33] S. Naseri and A. H. Banihashemi, "Construction of time invariant spatially coupled LDPC codes free of small trapping sets," *IEEE Trans. Commun.*, vol. 69, no. 6, pp. 3485–3501, Jun. 2021.

[34] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.

[35] H. J. Ryser, *Combinatorial Mathematics*, ser. Carus Mathematical Monographs. Washington, DC: The Mathematical Association of America, 1963, no. 14.

[36] J. Von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomials over $\mathbb{F}_2$," in *Proc. 1997 International Symposium on Symbolic and Algebraic Computation (ISSAC '96)*, Y. N. Lakshman, Ed. Zurich, Switzerland: ACM Press, 1996, technical report TR-RSFB96-018, University of Paderborn, Germany, 43 pages. Available at: http://wwwmath.uni-paderborn.de/aggathen/Publications/polyfactTR.ps.

[37] S. Gao, "Additive fast Fourier transforms for finite fields and their applications," *Journal of Symbolic Computation*, vol. 45, no. 8, pp. 933–947, Dec. 2010.

[38] M. H. Tadayon, A. Tasdighi, M. Battaglioni, M. Baldi, and F. Chiaraluce, "Efficient search of compact QC-LDPC and SC-LDPC convolutional codes with large girth," *IEEE Commun. Lett.*, vol. 22, no. 6, pp. 1156–1159, Jun. 2018.

[39] J. Xu, L. Chen, I. Djurdjevic, S. Lin, and K. Abdel-Ghaffar, "Construction of regular and irregular LDPC codes: Geometry decomposition and masking," *IEEE Trans. Inf. Theory*, vol. 53, no. 1, pp. 121–134, Jan. 2007.

[40] K.-J. Kim, S. Myung, S.-I. Park, J.-Y. Lee, M. Kan, Y. Shinohara, J.-W. Shin, and J. Kim, "Low-density parity-check codes for ATSC 3.0," *IEEE Trans. Broadcast.*, vol. 62, no. 1, pp. 189–196, Mar. 2016.

[41] Y. Zhang, K. Peng, J. Song, and Y. Wu, "Quasi-cyclic spatially coupled LDPC code for broadcasting," *IEEE Trans. Broadcast.*, vol. 66, no. 1, pp. 187–194, Mar. 2020.

[42] S. Ten Brink, G. Kramer, and A. Ashikhmin, "Design of low-density parity-check codes for modulation and detection," *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 670–678, Apr. 2004.

APPENDIX A
FINITE-LENGTH ENCODING COMPLEXITY

In this appendix, we demonstrate the advantage of encoding an SC-LDPC code using the proposed permanent-based method over conventional Gaussian elimination.

A. Proposed method

Let us compute the cost, in terms of binary operations, of computing the polynomial generator matrix (i.e., (23)) corresponding to (22). Since the rightmost 4 columns of $\mathfrak{G}(\mathrm{D})$ are defined by theory, we only need to evaluate the cost of computing the 16 polynomial permanents of the following submatrices:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & \mathrm{D} \\ 0 & 1 & 1 & \mathrm{D}^{43} \\ 0 & 0 & 1 & \mathrm{D}^{61} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & \mathrm{D}^{3} \\ 0 & 1 & 1 & \mathrm{D}^{129} \\ 0 & 0 & 1 & \mathrm{D}^{44} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & \mathrm{D}^{30} \\ 0 & 1 & 1 & \mathrm{D}^{39} \\ 0 & 0 & 1 & \mathrm{D}^{23} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & \mathrm{D}^{105} \\ 0 & 1 & 1 & \mathrm{D}^{67} \\ 0 & 0 & 1 & \mathrm{D}^{11} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 1 & 0 & \mathrm{D} \\ 0 & 1 & 0 & \mathrm{D}^{43} \\ 0 & 0 & 1 & \mathrm{D}^{61} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 1 & 0 & \mathrm{D}^{3} \\ 0 & 1 & 0 & \mathrm{D}^{129} \\ 0 & 0 & 1 & \mathrm{D}^{44} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 1 & 0 & \mathrm{D}^{30} \\ 0 & 1 & 0 & \mathrm{D}^{39} \\ 0 & 0 & 1 & \mathrm{D}^{23} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 1 & 0 & \mathrm{D}^{105} \\ 0 & 1 & 0 & \mathrm{D}^{67} \\ 0 & 0 & 1 & \mathrm{D}^{11} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D} \\ 0 & 1 & 0 & \mathrm{D}^{43} \\ 0 & 1 & 1 & \mathrm{D}^{61} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D}^{3} \\ 0 & 1 & 0 & \mathrm{D}^{129} \\ 0 & 1 & 1 & \mathrm{D}^{44} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D}^{30} \\ 0 & 1 & 0 & \mathrm{D}^{39} \\ 0 & 1 & 1 & \mathrm{D}^{23} \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D}^{105} \\ 0 & 1 & 0 & \mathrm{D}^{67} \\ 0 & 1 & 1 & \mathrm{D}^{11} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D} \\ 1 & 1 & 0 & \mathrm{D}^{43} \\ 0 & 1 & 1 & \mathrm{D}^{61} \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D}^{3} \\ 1 & 1 & 0 & \mathrm{D}^{129} \\ 0 & 1 & 1 & \mathrm{D}^{44} \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D}^{30} \\ 1 & 1 & 0 & \mathrm{D}^{39} \\ 0 & 1 & 1 & \mathrm{D}^{23} \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & \mathrm{D} & 1 \\ 1 & 0 & 0 & \mathrm{D}^{105} \\ 1 & 1 & 0 & \mathrm{D}^{67} \\ 0 & 1 & 1 & \mathrm{D}^{11} \end{bmatrix}$$

To evaluate each permanent, we employ Ryser's algorithm which, for a $c\times c$ matrix, requires summing over all nonempty column subsets. In an efficient implementation, the 2^c-1 subsets are visited in an order where only one column index changes at a time, allowing each row-sum to be updated incrementally rather than recomputed from scratch. For each subset, the product of all c row-sums is computed. When applied to a polynomial matrix over $\mathbb{F}_2[\mathrm{D}]$, this process performs polynomial additions and multiplications. If the maximum monomial degree appearing in the submatrix is denoted by ℓ, then the cost of one 4×4 permanent, in terms of logic operations, is

$$\text{ANDs} = 15\,(\ell{+}1)\,(6\ell{+}3), \qquad \text{XORs} = 90\ell^2+180\ell+180, \tag{25}$$

where each polynomial addition costs $(\ell{+}1)$ XORs and each polynomial multiplication of degrees at most p, q costs $(p{+}1)(q{+}1)$ ANDs and $(p{+}1)(q{+}1)-(p{+}q{+}1)$ XORs.

The rationale beyond these counts is that Ryser's algorithm for a 4×4 matrix involves $2^4{-}1 = 15$ nonempty subsets of columns. For each subset, the four row-sums are first computed and then multiplied together. Each row-sum requires three polynomial additions, resulting in $4\times3 = 12$ additions per subset, and hence $12(\ell{+}1)$ XORs. The four row-sums are then multiplied to form a single polynomial. This can be done by three successive polynomial multiplications. If the row-sums have degree at most ℓ, then the first multiplication yields a polynomial of degree at most 2ℓ, the second of degree at most 3ℓ, and the third of degree at most 4ℓ. The corresponding costs are:

- first multiplication (degrees $\le\ell$ and $\le\ell$): $(\ell{+}1)^2$ ANDs and ℓ^2 XORs;
- second multiplication (degrees $\le 2\ell$ and $\le\ell$): $(2\ell{+}1)(\ell{+}1)$ ANDs and $2\ell^2$ XORs;
- third multiplication (degrees $\le 3\ell$ and $\le\ell$): $(3\ell{+}1)(\ell{+}1)$ ANDs and $3\ell^2$ XORs.

Summing these contributions, each subset requires

$$3(\ell{+}1)^2+(2\ell{+}1)(\ell{+}1)+(3\ell{+}1)(\ell{+}1) = (\ell{+}1)(6\ell{+}3)$$

ANDs for the multiplications, and

$$\ell^2+2\ell^2+3\ell^2 = 6\ell^2$$

XORs for the multiplications. Including the $12(\ell{+}1)$ XORs for the additions, the per-subset totals are

$$\begin{aligned}\text{ANDs}_{\text{sub}} &= (\ell+1)(6\ell+3),\\ \text{XORs}_{\text{sub}} &= 6\ell^2+12(\ell+1) = 6\ell^2+12\ell+12.\end{aligned}$$

Finally, multiplying by the 15 subsets yields (25).

In all the submatrices above, the largest monomial exponents are $\ell = 61, 129, 39, 105$ for the four groups, respectively. The total number of XOR and AND operations are reported in Table II.

Encoding a polynomial information vector through the polynomial generator matrix obtained as described above needs to account for two steps:

- division of all the entries of (23) by $\mathrm{D}+1$, to avoid catastrophic encoding and enable systematic encoding;

TABLE II
BINARY OPERATIONS REQUIRED TO COMPUTE EACH POLYNOMIAL PERMANENT USING RYSER'S ALGORITHM.

Submatrices (columns)	ℓ	**ANDs**	**XORs**
(1,2,3,5), (1,2,4,5), (1,3,4,5), (2,3,4,5)	61	343 170	346 050
(1,2,3,6), (1,2,4,6), (1,3,4,6), (2,3,4,6)	129	1 515 150	1 521 090
(1,2,3,7), (1,2,4,7), (1,3,4,7), (2,3,4,7)	39	142 200	144 090
(1,2,3,8), (1,2,4,8), (1,3,4,8), (2,3,4,8)	105	1 006 470	1 011 330
Total		**12 027 960**	**12 090 240**

- multiplication of $u(\mathrm{D})$ by the non-systematic portion of $\mathfrak{G}^*(\mathrm{D})$.

For $p(\mathrm{D}) = \sum_{i=0}^{\ell} p_i\mathrm{D}^i$, the quotient $q(\mathrm{D})$ defined by $p(\mathrm{D}) = (\mathrm{D}+1)q(\mathrm{D})$ can be computed via the recurrence

$$q_0 = p_0, \qquad q_i = p_i \oplus q_{i-1} \quad (i\in[0,\ell-1]),$$

which requires exactly ℓ XORs and no ANDs. The XOR counts per entry of (23) (in the non-standard portion of $\mathfrak{G}(\mathrm{D})$) are thus

$$\begin{Bmatrix} 62 & 62 & 62 & 61\\ 130 & 130 & 129 & 129\\ 40 & 40 & 39 & 39\\ 106 & 105 & 105 & 105 \end{Bmatrix}.$$

Summing across all 16 entries gives a total number of 1344 XORs and no ANDs required.

Let $\mathfrak{u}(\mathrm{D}) = [\mathfrak{u}_1(\mathrm{D})\ \ \mathfrak{u}_2(\mathrm{D})\ \ \mathfrak{u}_3(\mathrm{D})\ \ \mathfrak{u}_4(\mathrm{D})]$, with $\deg(\mathfrak{u}_k(\mathrm{D})) = d_k$. The multiplication $\mathfrak{u}_k(\mathrm{D})\cdot\mathfrak{U}_{j,i}(\mathrm{D}) = \sum_{t=i}^{j}\mathfrak{u}_k(\mathrm{D})\mathrm{D}^t$ coincides with the XOR of $V = j{-}i{+}1$ shifted copies of $\mathfrak{u}_k(\mathrm{D})$ and requires exactly

$$\begin{aligned}C_{\text{step}}(d_k;i,j) &= \sum_{t=1}^{V-1}\big(d_k+i+t+1\big)\\ &= (V-1)(d_k+i+1)+\frac{(V-1)V}{2}.\end{aligned} \tag{26}$$

XORs, with no ANDs. Adding one extra monomial term into the result costs one polynomial addition with degree equal to the current maximum, that is,

$$C_{\text{mono}}(d_k;j) = d_k+j+1. \tag{27}$$

If an entry is a sum of two step polynomials $\mathfrak{U}_{i_0,i_1}(\mathrm{D})$ and $\mathfrak{U}_{i'_0,i'_1}(\mathrm{D})$, compute them separately and XOR them once, which costs

$$\begin{aligned}C_{\text{2step}}(d_k;i_1,j_1,i_2,j_2) &= C_{\text{step}}(d_k;i_1,j_1)\\ &\quad + C_{\text{step}}(d_k;i_2,j_2)+\big(d_k+\max\{j_1,j_2\}+1\big).\end{aligned}$$

In our specific case, we thus have

$$\begin{aligned}
C_{1,1} &= C_{\text{step}}(d_1; 44, 61) + 2\,C_{\text{mono}}(d_1; 61),\\
C_{1,2} &= C_{\text{step}}(d_1; 44, 61) + C_{\text{mono}}(d_1; 61),\\
C_{1,3} &= C_{\text{step}}(d_1; 43, 61) + C_{\text{mono}}(d_1; 61),\\
C_{1,4} &= C_{\text{step}}(d_1; 43, 60) + C_{\text{mono}}(d_1; 60),\\
C_{2,1} &= C_{\text{2step}}(d_2; 45, 129, 0, 3),\\
C_{2,2} &= C_{\text{2step}}(d_2; 45, 129, 0, 2),\\
C_{2,3} &= C_{\text{2step}}(d_2; 45, 128, 0, 2),\\
C_{2,4} &= C_{\text{2step}}(d_2; 44, 128, 0, 2),\\
C_{3,1} &= C_{\text{2step}}(d_3; 31, 39, 0, 23),\\
C_{3,2} &= C_{\text{2step}}(d_3; 30, 39, 0, 23),\\
C_{3,3} &= C_{\text{2step}}(d_3; 30, 38, 0, 23),\\
C_{3,4} &= C_{\text{2step}}(d_3; 30, 38, 0, 22),\\
C_{4,1} &= C_{\text{2step}}(d_4; 68, 105, 0, 11),\\
C_{4,2} &= C_{\text{2step}}(d_4; 68, 104, 0, 11),\\
C_{4,3} &= C_{\text{2step}}(d_4; 67, 104, 0, 11),\\
C_{4,4} &= C_{\text{2step}}(d_4; 67, 104, 0, 10).
\end{aligned} \tag{28}$$

Each output polynomial can be expressed as

$$\mathfrak{c}_r(\mathrm{D}) = \sum_{k=1}^{4} \mathfrak{u}_k(\mathrm{D})\, \mathfrak{g}^*_{k,r}(\mathrm{D}).$$

After computing the four partial products (whose costs are denoted by $C_{k,r}$ above), these must be combined through XOR operations. Let

$$\begin{aligned}
M_r &\triangleq \max_{k\in[1,4]} \deg\big(\mathfrak{u}_k(\mathrm{D})\, \mathfrak{g}^*_{k,r}(\mathrm{D})\big)\\
&= \max_{k\in[1,4]} \big(\deg\, \mathfrak{u}_k(\mathrm{D}) + \deg\, \mathfrak{g}^*_{k,r}(\mathrm{D})\big)\,.
\end{aligned}$$

denote the largest degree among the four partial products involved in forming the r-th column of $\mathfrak{u}(\mathrm{D})\,\mathfrak{G}^*(\mathrm{D})$, that is, the maximum degree of any term that can appear in the r-th encoded polynomial. A tight upper bound on the number of XOR operations required to accumulate the four partial products sequentially is therefore

$$C_{\text{acc}}(r) = 3\,(M_r + 1), \tag{29}$$

since three polynomial additions of degree at most M_r are needed.

In our case, M_r can be read off directly from the largest exponents appearing in the entries of $\mathfrak{G}^*(\mathrm{D})$. Specifically, for the four rows of the left block, the maximum shift values are $j \in \{61,\, 129,\, 39,\, 105\}$, corresponding respectively to $\mathfrak{U}_{61,44}$, $\mathfrak{U}_{129,45}$, $\mathfrak{U}_{39,31}$, and $\mathfrak{U}_{105,68}$. Hence,

$$\deg\big(\mathfrak{u}_{\mathrm{k}}(\mathrm{D})\mathfrak{g}^*_{\mathrm{k,r}}(\mathrm{D})\big) = \begin{cases} d_1 + 61, & k = 1,\\ d_2 + 129, & k = 2,\\ d_3 + 39, & k = 3,\\ d_4 + 105, & k = 4, \end{cases}$$

that is, the degree of each partial product equals the degree d_k of the corresponding information polynomial plus the largest shift present in that matrix row. So, the exact XOR cost to encode $\mathfrak{u}(\mathrm{D})$ is

$$C_{\text{total}} = \sum_{r=1}^{4}\sum_{k=1}^{4} C_{k,r} \;+\; \sum_{r=1}^{4} C_{\text{acc}}(r), \tag{30}$$

with $C_{k,r}$ from (28) and $C_{\text{acc}}(r)$ from (29). No ANDs are required, since we realize each product using shifts and XORs (monomial multiplications are shifts; additions are XORs).

TABLE III
XOR COUNTS PER COLUMN FOR DEGREE OF $\mathfrak{u}_k$ BEING 1 000 FOR EACH k.

	$r=1$	$r=2$	$r=3$	$r=4$
$\mathfrak{u}_1(\mathrm{D})\,\mathfrak{g}^*_{1,r}(\mathrm{D})$	20 042	18 980	20 025	18 962
$\mathfrak{u}_2(\mathrm{D})\,\mathfrak{g}^*_{2,r}(\mathrm{D})$	95 573	94 569	93 438	94 484
$\mathfrak{u}_3(\mathrm{D})\,\mathfrak{g}^*_{3,r}(\mathrm{D})$	32 631	33 663	32 622	31 598
$\mathfrak{u}_4(\mathrm{D})\,\mathfrak{g}^*_{4,r}(\mathrm{D})$	52 439	51 332	52 401	51 389
Sum of four products	200 685	198 544	198 486	196 433
Accumulation (three adds)	3 390	3 390	3 387	3 387
Total per column	**204 075**	**201 934**	**201 873**	**199 820**

Example 7 Let

$$\mathfrak{u}(\mathrm{D}) = \big[\mathfrak{u}_1(\mathrm{D})\ \mathfrak{u}_2(\mathrm{D})\ \mathfrak{u}_3(\mathrm{D})\ \mathfrak{u}_4(\mathrm{D})\big],$$

with the entries of $\mathfrak{u}(\mathrm{D})$ having degree 1 000. The XOR counts to form the four products $\mathfrak{u}_k(\mathrm{D})\,\mathfrak{g}^*_{k,r}(\mathrm{D})$ and then accumulate them into the r-th output are summarized in Table III. Here the accumulation cost uses $M_r = \max\{1\,000 + Y_{1r},\, 1\,000 + Y_{2r},\, 1\,000 + Y_{3r},\, 1\,000 + Y_{4r}\}$ with

$$(Y_{kr}) = \begin{bmatrix} 61 & 61 & 61 & 60\\ 129 & 129 & 128 & 128\\ 39 & 39 & 38 & 38\\ 105 & 104 & 104 & 104 \end{bmatrix},$$

$$M_1 = M_2 = 1\,129, \qquad M_3 = M_4 = 1\,128.$$

hence $C_{\text{acc}}(r) = 3\,(M_r + 1)$ gives 3 390, 3 390, 3 387, 3 387 XORs, respectively.

As expected, and as illustrated in Example 7, the cost of multiplying the polynomial information vector by the polynomial generator matrix (see Table III) is relatively small and essentially negligible compared to the cost of computing the polynomial generator matrix itself (see Table II). This matrix can be used to encode for any value of L. By contrast, as shown in the next subsection, obtaining a binary generator matrix via Gaussian elimination is tied to the chosen value of L, and its computation is already far more expensive than that of the polynomial generator matrix, even for very small values of L.

B. Gaussian elimination

We first expand the polynomial matrix (22) into its binary form to obtain the syndrome former matrix. The banded parity-check matrix $\mathbf{H}^{\top}_{[0,L-1]}$ in (1) is then built by coupling L of these syndrome former matrices. Gaussian elimination (over $\mathbb{F}_2$) is performed on this binary matrix using a fixed, time-ordered pivot pattern (no polynomial operations). In our

This article has been accepted for publication in IEEE Transactions on Information Theory. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIT.2026.3695390

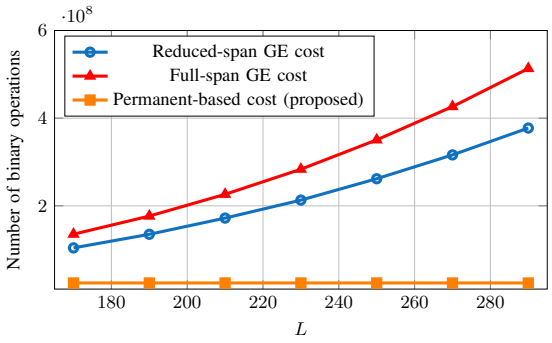

Fig. 4. Number of binary operations required by Gaussian elimination (GE in the legend) and the proposed permanent-based method to compute the generator matrix versus termination length L.

implementation, each row addition increases the XOR counter by the number of bit positions between the leftmost and rightmost nonzero entries of the two rows involved. This reflects the portion of the rows that is actually processed during Gaussian elimination, rather than the full row length.

Remark 5 This estimate is highly optimistic, since we are ignoring the cost of finding the positions of the leftmost and rightmost nonzero entries in the two rows, which are not known a priori. Moreover, we do *not* account for comparisons, row swaps, or memory-access overhead.

Figure 4 shows the number of XOR operations required by Gaussian elimination as a function of the termination length L, for some very small values of L (recall that $L \geq m_s = 129$), using the above implementation (blue curve). As expected, the number of binary operations needed to obtain the generator matrix grows rapidly and becomes prohibitively large as L increases. For comparison, we also include the full-span count (red curve), obtained by conservatively incrementing the XOR counter by La for each row operation, i.e., assuming that the entire row segment of length La is processed during Gaussian elimination. For reference, we also report the cost of obtaining a polynomial generator matrix through our proposed permanent-based method (orange curve) from Table II. The comparison highlights that the latter approach requires far fewer operations to obtain the polynomial generator matrix, and relies on a procedure that does not depend on L. The cost of the encoding step itself, both in the polynomial and in the binary domain, is negligible compared to the cost of constructing $\mathbf{G}$, and is therefore omitted.

Massimo Battaglioni (Member, IEEE) received the Laurea and Laurea Magistrale (summa cum laude) degrees in electronic engineering and the Ph.D. degree in information engineering from Università Politecnica delle Marche (UNIVPM), Ancona, Italy, in 2013, 2015, and 2019, respectively. In 2017, he was a Visiting Student with the Electrical and Information Technology Department, LTH, Lund University, Sweden. In 2018, he was a Visiting Student with the Klipsch School of Electrical and Computer Engineering, New Mexico State University (NMSU), Las Cruces, NM, USA; and the School of Electrical and Electronic Engineering, University College Dublin, Ireland. Since 2024, he has been a Tenure-Track Researcher with the Department of Information Engineering, UNIVPM. He has coauthored more than 50 scientific articles. His research interests include coding techniques for communications reliability and security. He served as an Editor for IEEE COMMUNICATIONS LETTERS from 2020 to 2026.

Marco Baldi (Senior Member, IEEE) received the Laurea degree (Hons.) in electronics engineering and the Ph.D. degree in electronics, computer, and telecommunications engineering from Università Politecnica delle Marche (UNIVPM), Ancona, Italy. Since 2019, he has been an Associate Professor with the Department of Information Engineering, UNIVPM, where he also coordinates the local node of the CINI Cybersecurity National Laboratory and takes part in the Research and Service Center for Privacy and Cybersecurity (CRiSPY). He has coauthored more than 200 scientific articles, one book and four patents. His research interests include coding and cryptography for information reliability and security. He serves as an Area Editor in coding for IEEE Communications Letters, as a Senior Area Editor for IEEE Transactions on Information Forensics and Security and as an Associate Editor for IEEE Transactions on Communications.

Franco Chiaraluce (Senior Member, IEEE) received the Laurea degree (summa cum laude) in electronic engineering from the University of Ancona in 1985. Since 1987, he has been with the Department of Electronics and Automatics, University of Ancona. He is currently a Full Professor in telecommunications with Università Politecnica delle Marche (UNIVPM), Ancona, Italy, where he is also the Director of the Department of Information Engineering (DII). He has coauthored more than 350 scientific articles and three books and holds three patents. On his research topics, he collaborates with national and international companies. His main research interests include various aspects of communication systems theory and design, with a special emphasis on error-correcting codes, cryptography, and physical layer security.