TS-LDPC Codes: Turbo-Structured Codes With Large Girth

Jin Lu, *Member, IEEE*, and José M. F. Moura, *Fellow, IEEE*

***Abstract*—We consider turbo-structured low-density parity-check (TS-LDPC) codes—structured regular codes whose Tanner graph is composed of two trees connected by an interleaver. TS-LDPC codes with good girth properties are easy to construct: careful design of the interleaver component prevents short cycles of any desired length in its Tanner graph. We present algorithms to construct TS-LDPC codes with arbitrary column weight $j \geq 2$ and row weight k and arbitrary girth g. We develop a linear complexity encoding algorithm for a type of TS-LDPC codes—encoding friendly TS-LDPC (EFTS-LDPC) codes. Simulation results demonstrate that the bit-error rate (BER) performance at low signal-to-noise ratio (SNR) is competitive with the error performance of random LDPC codes of the same size, with better error floor properties at high SNR.**

***Index Terms*—Error floor, girth, interleaver, low-density parity-check (LDPC) codes, turbo-structured.**

I. Introduction

LOW-density parity-check (LDPC) codes, [1], are being considered in numerous applications including digital communication systems and magnetic recording channels. Their bit-error rate (BER) performance using iterative decoding is close to the Shannon limit, [2].

There are several ways to specify LDPC codes. In this paper, we use indistinctly the code parity-check matrix $\boldsymbol{H}$ and its associated bipartite Tanner graph, [3]. We adopt the common notation of referring to a code with block length n and column and row weights j and k, respectively, as an (n, j, k)-LDPC code. The rate of the code is (approximately) given by $r = 1 - j/k$. An important parameter affecting the performance of the code is the length of the shortest cycle in its Tanner graph. This parameter is the girth g of the code. The relation of girth to performance is not completely understood. Codes with four-cycles and projective geometry codes, which may have limited girth, can perform well, [4]. However, in general, in codes with small g, i.e., with short cycles in the Tanner graph, the decoding algorithm takes longer to converge or may fail to converge to the optimal decoding result. On the other hand, for codes with very large girth g, the first $\left\lfloor \frac{g-2}{4} \right\rfloor$ decoding iterations correspond to optimal decoding iterations. For moderate to high signal-to-noise ratio (SNR), it is with very high probability that the decoding success is reached within the "optimal" $\left\lfloor \frac{g-2}{4} \right\rfloor$ iterations when g is large. Therefore, we can actually achieve optimal decoding results in the sense that the probability of symbol error is minimized. Furthermore, [3] shows that a lower bound on the minimum distance $d_{\min}$ of LDPC codes increases exponentially with the girth g. Therefore, it is desirable to have LDPC codes with good girth properties.

An important consideration for the practical application of LDPC codes is the regularity and structure of their encoding and decoding implementation. Recently, structured regular LDPC codes have drawn much attention in the sense that they facilitate low-complexity encoder and decoder designs. We briefly review several typical designs of structured regular LDPC codes in this paper.

We discuss here very briefly recent literature on designs of LDPC codes. A more extensive review of LDPC codes is in [5], [6]. Kou, Lin, and Fossorier [7], [8] developed LDPC codes based on finite geometries and incidence structures. Finite-geometry LDPC codes can be designed over a wide variety of block lengths and code rates and achieve good minimum distances. Finite-geometry LDPC codes have girth 6. Another method to construct structured 4-cycle-free regular LDPC codes is based on balanced incomplete block designs (BIBD) [9]–[13]. A BIBD is defined as a collection B of equal size blocks, comprising elements drawn from a set V, such that each pair of distinct elements (x, y) of V occurs in exactly λ blocks of B. BIBD-based codes are well structured, free of 4-cycles, i.e., with girth $g = 6$, but exhibit a large number of 6-cycles in their Tanner graphs. Finite-geometry codes and BIBD codes are examples of cyclic and quasi-cyclic [14], [15] codes. For these codes with column weight $j \geq 3$ the girth is less than or equal to 12, see Fossorier, [16], and our own paper [17]. This prevents the girth of (n, j, k)-cyclic and quasi-cyclic LDPC codes of growing according to the prediction $g \propto \log_{(j-1)(k-1)} n$ derived by [1] for random LDPC codes; hence, such codes with very long code block lengths perform poorly.

Hu, Eleftheriou, and Arnold propose in [18] a nonalgebraic method named progressive edge-growth (PEG). They present examples of codes of girth $g = 8$ by progressively establishing edges between bit and check nodes in an edge-by-edge manner. PEG optimizes the placement of a new edge on the Tanner graph with the goal of maximizing the local girth. Reference [16] constructed LDPC codes from circulant permutation matrices. It

Manuscript received November 19, 2004; revised November 18, 2006. This work was supported by the Data Storage Systems Center (DSSC) at Carnegie Mellon University. The material in this paper was presented in part at IEEE 4th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Rome, Italy, June 2003.

J. Lu is with Sun Microsystems, Louisville, CO 80028 USA (e-mail: j.lu@sun.com).

J. M. F. Moura is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213-3890 USA (e-mail: moura@ece.cmu.edu).

Communicated by M. P. C. Fossorier, Associate Editor for Coding Techniques.

Color versions of Figures 6, 9–13, and 15–24 in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2006.890690

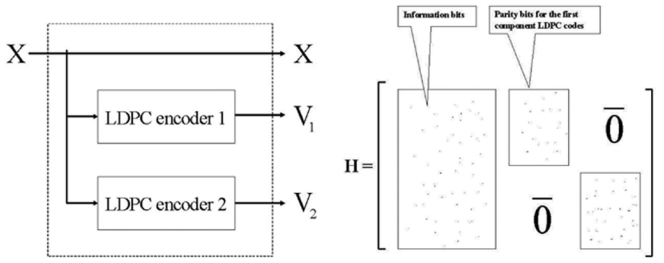

Fig. 1. Left: The encoder structure for the concatenated LDPC codes in [22]. Right: The parity-check matrix $\boldsymbol{H}$ for the concatenated LDPC codes in [21], [22].

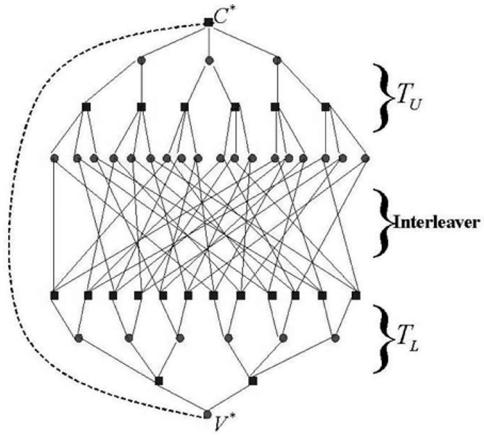

Fig. 2. The decoder Tanner graph of a TS-LDPC code: $h = 4$, $j = 3$, and $k = 4$.

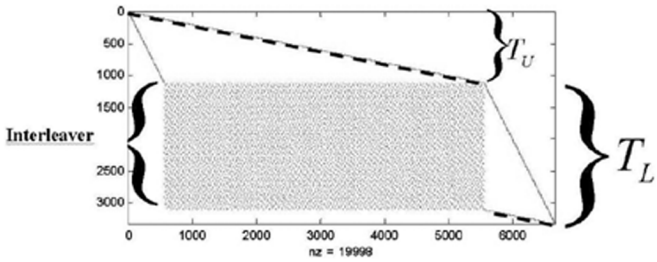

Fig. 3. Parity-check matrix $\boldsymbol{H}$ of a $(6666, 3, 6)$ TS-LDPC code with girth 10.

presents conditions for such codes to achieve girth up to $g \leq 12$. References [19], [20] also provide good constructions for structured LDPC codes with large girth.

This paper considers a class of structured regular LDPC codes, the turbo-structured LDPC (TS-LDPC) codes, that can be designed with arbitrary large girth g by appropriately choosing the code block length n. We describe an algorithm to construct TS-LDPC codes with any desired column and row weights $j \geq 2$ and k, hence, with any desired practical rate r. TS-LDPC codes are hardware friendly: they are regular and structured, and their parity-check matrix $\boldsymbol{H}$, which can be very large, is determined from a much smaller object, the shift matrix $\boldsymbol{S}$, so that their memory requirements can be negligible. We show by simulation that the BER performance of TS-LDPC codes is competitive with that of random LDPC codes, with better error floor properties. Finally, we exploit the specific structure of the Tanner graph of a particular type of TS-LDPC codes—encoding-friendly TS-LDPC (EFTS-LDPC) codes to derive an encoding algorithm with linear complexity.

The Tanner graph of TS-LDPC codes is composed of two trees, an upper tree T_U and a lower tree T_L, that are connected in a turbo-like manner by an interleaver $\mathcal{I}$. This turbo structure is exploited to facilitate the systematic design of LDPC codes with large girth g and flexible code rates r. Turbo structures have been used in [21], [22] to construct LDPC codes. The codes proposed by these authors combine two LDPC codes as component codes on the *encoder* side. Reference [21] directly borrows the structure of the turbo *encoder*, replacing the recursive convolutional codes commonly used in turbo codes with a tree code—a specific LDPC code whose associated graph for the code-generating matrix $\boldsymbol{G}$, not for the parity-check matrix $\boldsymbol{H}$, is a tree. In [22], the *encoder* is a parallel concatenation of two LDPC codes without an interleaver, as shown on the left of Fig. 1. Our TS-LDPC codes stand in sharp contrast with the codes in [21], [22]: they are turbo-like from the *decoder* point of view, see Fig. 2, i.e., from the parity-check matrix $\boldsymbol{H}$ and its associated Tanner graph; their Tanner graphs are NOT the concatenation of two trees through an interleaver as our decoders are. Fig. 1 on the right shows the parity-check matrices $\boldsymbol{H}$ for the codes in [21], [22] and Fig. 3 for a TS-LDPC code. The structure of these matrices bear no resemblance. The top and lower tree structures of the TS-LDPC codes give rise to the top and bottom diagonal lines in $\boldsymbol{H}$, while the cloud of points in $\boldsymbol{H}$ arises from the interleaver. In contradistinction to this structure, the $\boldsymbol{H}$ matrix for the code in [22] has rectangular blocks, the two right ones corresponding to the parity bits of each encoder component.

We comment briefly on the relation between TS-LDPC codes and generalized quasi-cyclic LDPC (GQC-LDPC) codes, a class of LDPC code designs that we described recently in [17], [23]–[25]. In the terminology of TS-LDPC codes, GQC-LDPC codes are simply reduced to an interleaver, no upper or lower trees are present like in TS-LDPC codes. At first sight, it seems that the TS-LDPC codes generalize GQC-LDPC codes. On the one hand, they are distinct from GQC-LDPC codes since they add to the interleaver the upper and lower trees in their code Tanner graph; on the other hand, they seem to borrow in some generic sense ideas from the design of GQC-LDPC codes including"grouping and shifting" to avoid cycles within the interleaver. In this restricted sense, Algorithm 1 for TS-LDPC codes extends Algorithm 1 for GQC-LDPC codes, see [24]. However, the addition of the upper and lower trees in the TS-LDPC codes adds additional constraints and degrees of freedom to the design process not present for the GQC-LDPC codes. The design constraints on the interleaver being different, the details of the construction and the theorems underlying the construction of TS-LDPC codes are quite different from the corresponding results for GQC-LDPC codes.

Both codes can be designed with flexible girth and code rates but their designs are very different and they represent distinct tradeoffs: usually, for codes of similar girth, GQC-LDPC codes have smaller block length n, or, in alternative, for codes with similar block length n, GQC-LDPC codes can be designed

with larger girth; for similar girth, TS-LDPC codes have shorter diameter, which may lead to faster convergence; and, for TS-LDPC codes, we can design faster decoding, [26], and faster (linear) encoding algorithms, see Section V.

II. TS-LDPC Codes

As shown in Fig. 2, the Tanner graph of TS-LDPC codes is composed of three components: two height-balanced trees, denoted as an upper-tree T_U and a lower-tree T_L, and an interleaver $\mathcal{I}$ that connects T_U and T_L. The leaf nodes of T_U are bit nodes (circles), whereas the leaf nodes of T_L are check nodes (squares). The number of layers or tiers in the trees T_U and T_L is kept the same and given by h. We require h to be an even number because we start the upper tree from a check node and require its leaf nodes to be bit nodes, and, similarly, we start the lower tree from a bit node and require its leaf nodes to be check nodes. The two trees are "coupled" in a turbo-like manner such that many edges join the leaf nodes of T_U and T_L together, see Fig. 2. The structure formed by the edges connecting the leaf nodes of T_U and the leaf nodes of T_L is named the interleaver $\mathcal{I}$.

The first tier of T_U contains only one check node C^*. To match T_U, we let the root of T_L be a bit node V^* and connect V^* to C^*. For the TS-LDPC code in Fig. 2, the height of the tree is $h = 4$, and the column and row weights are, respectively, $j = 3$ and $k = 4$. The rate of the code is $\rho = 1 - \frac{\text{rank}(\boldsymbol{H})}{n} \approx 1 - \frac{j}{k}$ where $\boldsymbol{H}$ denotes the parity-check matrix and n is the code block length. This rate can be arbitrarily adjustable as long as we can design TS-LDPC codes with appropriate values for j and k. We will present an algorithm to achieve just that.

III. Interleaver Design

This section considers the design of TS-LDPC codes, in particular the design of the interleaver $\mathcal{I}$. The interleaver is designed by specifying the rules of how to connect the leaf bit nodes in the upper tree T_U to the leaf check nodes in the lower tree T_L. Interleaver designs for turbo codes have been extensively studied [27]–[30]. For example, an S-random interleaver [27] guarantees that any two positions within distance S are mapped to two positions with distance greater than S after interleaving. The functions of the interleavers for turbo codes are to avoid low-weight codewords and to decrease the correlation between the extrinsic information and the input data sequence. They are not designed to construct Tanner graphs with large girth. Hence, we can not directly borrow the existing interleaver designs for turbo codes, say, the S-random interleaver, for designing TS-LDPC codes. We need to develop new interleavers that suit the structure of TS-LDPC codes and lead to TS-LDPC codes with large girth.

Section III-A introduces $p-q$-alternate-decimal indexing that is used in labeling the nodes in the Tanner graph and in determining how to connect the leaf nodes in T_U and T_L. We design these rules to prevent cycles of length smaller than the desired girth g. We achieve this by categorizing the cycles in two types: type I and type II cycles. Section III-B specifies the connecting rules to avoid type I cycles. Type II cycles are considered in Section III-C—they are prevented by "grouping and shifting:" we group the leaf nodes and connect leaf nodes in distinct trees whose labels are appropriately shifted. Section III-D discusses the number of groups needed. Based on the discussions in Sections III-A-D, Section III-E shows the detailed algorithm to construct TS-LDPC codes. Finally, Section III-F discusses the advantages provided by the structure of TS-LDPC codes in terms of the memory required to store them.

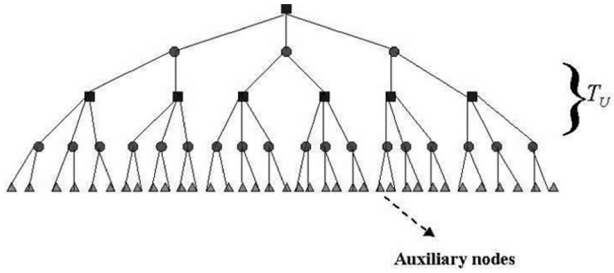

Fig. 4. Auxiliary nodes of T_U.

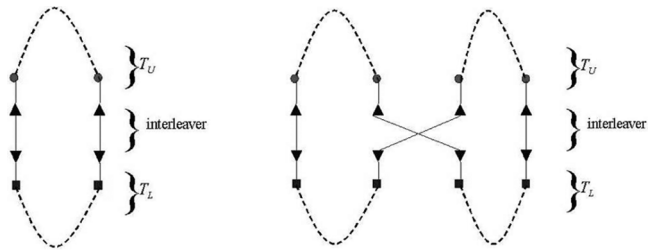

Fig. 5. Left: A type I cycle. Right: A type II cycle.

A. Auxiliary Nodes and $p-q$-Alternate-Decimal Indexing

Because of the regularity of the code, by construction, each leaf node in T_U is to be connected to $q = j - 1$ leaf nodes in T_L. This is a one-to-q mapping, while the standard interleaver is a one-to-one mapping between elements of two sets with the same size. To get a standard interleaver, we introduce "auxiliary nodes" (solid triangles) as shown in Fig. 4 to facilitate the code design. For each leaf node in T_U, we add $j-1$ auxiliary nodes as its children. Similarly, each leaf node in T_L has $k - 1$ auxiliary nodes as its descendants.

Due to the tree structure of the upper and lower components of the codes, cycles present in TS-LDPC codes must contain at least four "auxiliary nodes"—two auxiliary nodes of T_U and two auxiliary nodes of T_L. We use this observation to classify cycles into two disjoint categories: **Type I** cycles contain four and only four auxiliary nodes; **type II** cycles are all the other cycles in the code and contain six or more auxiliary nodes. We will dispose of each of these two types of cycles separately. Fig. 5 shows on the left a type I cycle and on the right a type II cycle.

For T_U with h tiers, there are $[(k-1)(j-1)]^{h/2}$ auxiliary nodes of T_U. We address the interleaver design problem algebraically by indexing all the auxiliary nodes of T_U and T_L. We start by numbering the auxiliary nodes in T_U from 0 to $[(k-1)(j-1)]^{h/2} - 1$ in the following format—the $p-q$-*alternate-decimal* format, where $p = k - 1$ and $q = j - 1$. We need h digits in the $p-q$-alternate-decimal indexing to label all the auxiliary nodes in T_U. These h digits are numbered from 1 to h, starting from the rightmost one. The odd-numbered digits take values 0 to $q - 1$ and the even-numbered digits take values

0 to $p-1$. We refer to the position of each digit as its *coordinate*. Similarly, we index all the auxiliary nodes of T_L from 0 to $[(k-1)(j-1)]^{h/2}-1$ and represent also all these indices in $q-p$-*alternate-decimal* format.

To be concrete, we provide an example. The index X_{p-q} of the auxiliary node in T_U in Fig. 4, in $p-q$-alternate-decimal form, is

$$X_{p-q} = \overline{\underbrace{x_4}_{p}\ \underbrace{x_3}_{q}\ \underbrace{x_2}_{p}\ \underbrace{x_1}_{q}}. \tag{1}$$

In (1), x_i, $i=1,\ldots,4$, represents the ith digit. The coordinate of x_i is i. The corresponding value of X_{p-q} in decimals is

$$X_{p-q} = \left(x_4 \times pq^2 + x_3 \times pq + x_2 \times q + x_1\right)_{10}.$$

From here on, when we refer to "indices" of nodes we mean their $p-q$-alternate-decimal or $q-p$-alternate-decimal representations.

Before establishing the connection rules, we first prove the following auxiliary lemma.

Lemma 1:

(a) Let the distance between two auxiliary nodes A and B within T_U be $d_U(A,B)$. X_A and X_B are indices for A and B, respectively. If i is the leftmost coordinate where the digits of X_A and X_B differ from each other, then $d_U(A,B)=2i$.

(b) Denote the distance between two auxiliary nodes A and B within T_L as $d_L(A,B)$. X_A and X_B are indices for A and B. If i is the leftmost coordinate where the digits of X_A and X_B differ from each other, then $d_L(A,B)=2i$.

Proof: We first prove part (a) of the lemma. From the definition of the $p-q$-alternate-decimal indexing, the first common ancestor R of the auxiliary nodes A and B is in the $(h+1-i)$th tier of T_U where i is the leftmost coordinate where the digits of X_A and X_B differ from each other and h denotes the number of tiers in T_U. For example, with reference to the index in Fig. 4, let $X_A=\overline{0000}$ and $X_B=\overline{1000}$ be the indices of two auxiliary nodes A and B of T_U. Since the leftmost coordinate where the digits of X_A and X_B differ from each other is the fourth, then the first common ancestor R of the auxiliary nodes A and B is the root of T_U, located in the first tier of T_U.

To find the shortest path in T_U that connects A and B, we have to go up i tiers from A to reach its first common ancestor R and then go downwards i tiers from R to B. Therefore, the distance through the tree T_U between A and B is $d_U(A,B)=2i$. Similarly, we can prove part (b) of the lemma. This completes the proof. □

We now consider the connection rule to avoid type I cycles.

B. Avoiding Short Type I Cycles—Digit-Wise Reversal

We start from a simple interleaver design—*digit-wise reversal*, [31]. For an index X_{p-q} in $p-q$-alternate-decimal form with h digits, its digit-wise reversal interchanges the ith digit and the $(h+1-i)$th digit for $i=1,\ldots,h/2$. We represent the digit-wise reversal operator by $\pi_d(\cdot)$. For the index X_{p-q} in (1), its digit-wise reversal is

$$\begin{aligned}\pi_d(X_{p-q}) &= \overline{\underbrace{x_1}_{q}\ \underbrace{x_2}_{p}\ \underbrace{x_3}_{q}\ \underbrace{x_4}_{p}} \\ &= \left(x_1 \times qp^2 + x_2 \times qp + x_3 \times p + x_4\right)_{10}. \end{aligned}\tag{2}$$

We state the advantage of the digit-wise reversal interleaver in the following theorem.

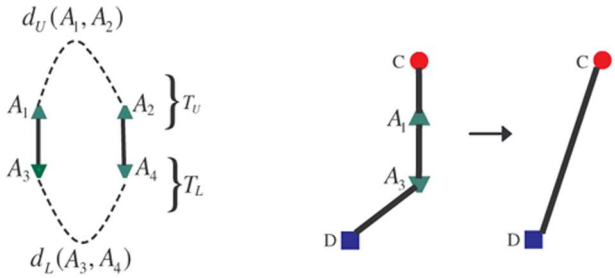

Fig. 6. Left: Type I cycle with four auxiliary nodes A_1, A_2, A_3, and A_4. Right: A path of length 3 that contains two auxiliary nodes is actually a path of length1.

Theorem 1: Connecting the auxiliary nodes indexed by X_{p-q} in T_U to the auxiliary nodes indexed by $\pi_d(X_{p-q})$ in T_L guarantees that any resulting type I cycle is at least of length $2h$, where h denotes the number of tiers in T_U.

Proof: From its definition, a type I cycle contains four auxiliary nodes A_1, A_2, A_3, and A_4 as shown on the left in Fig. 6. Their associated indices are X_1, X_2, X_3, and X_4, respectively. Denote the distance between A_1 and A_2 within the tree T_U as $d_U(A_1,A_2)$, and the distance of A_3 and A_4 within T_L as $d_L(A_3,A_4)$. According to the left plot in Fig. 6, the length of a type I cycle is:

$$L' = d_U(A_1,A_2) + d_L(A_3,A_4) + 2 \tag{3}$$

However, as auxiliary nodes A_1, A_2, A_3, and A_4 are imaginary nodes, the type I cycle does not contain them as vertices. For example, the path of length 3 with two auxiliary nodes A_1 and A_3 shown on the right in Fig. 6 is actually a path of length 1 in the Tanner graph. Therefore, the actual cycle length L is $L'-4$, i.e.,

$$L = L' - 4 = d_U(A_1,A_2) + d_L(A_3,A_4) - 2. \tag{4}$$

We relate the distance $d_U(A_1,A_2)$ to the value of their indices. By Lemma 1, $d_U(A_1,A_2)=2i$, where i is the leftmost coordinate where the digits of X_1 and X_2 differ from each other. Let h represent the height of T_U. After digit-wise reversal, the digits of X_1 and X_2 at the coordinate i become the digits of $\pi_d(X_1)$ and $\pi_d(X_2)$ at the coordinate $h+1-i$, respectively. So, the digits of $\pi_d(X_1)$ and $\pi_d(X_2)$ at the coordinate $h+1-i$ are different. According to the connecting rule stated in the theorem, $X_3=\pi_d(X_1)$ and $X_4=\pi_d(X_2)$, so the digits of X_3 and X_4 at the coordinate $h+1-i$ are different. Therefore, by Lemma 1

$$d_L(A_3,A_4) \geq 2(h+1-i).$$

By (4), the length of such a type I cycle is then

$$L = d_U(A_1, A_2) + d_L(A_3, A_4) - 2 \geq 2h.$$

From the above analysis, all type I cycles that result from following the connection rule in Theorem 1 are at least of length $2h$. This completes the proof. □

Theorem 1 states that to increase the length of type I cycles we need simply to increase the number of tiers h in the upper and lower trees.

C. Avoiding Type II Cycles—Grouping and Shifting

The connection rule in Theorem 1 prevents short type I cycles. We now consider the connection rule to avoid short type II cycles. To exclude short type II cycles, we propose grouping and shifting.

Shifting We define the *shift* S to be a constant in $q-p$-alternate-decimal format that is added to the original index $\pi_d(X_{p-q})$ to form a new index. We illustrate it with an example. Let $\pi_d(X_{p-q}) = \overline{x_1x_2x_3x_4}$, the shift $S = \overline{s_1s_2s_3s_4}$, and represent by $\dot{+}$ the digit-wise addition (with no carry). Then

$$\pi_d(X_{p-q}) \dot{+} S = \overline{y_1y_2y_3y_4}. \tag{5}$$

In (5), $y_i = x_i \dot{+} s_i = \mathrm{mod}(x_i + s_i, \mathrm{div}_i)$, where $\mathrm{div}_i = p$ if i is even and $\mathrm{div}_i = q$ if i is odd. In a similar fashion, we represent the digit-wise subtraction by $\dot{-}$.

Grouping We divide the auxiliary nodes of T_U into groups of the same size according to their indices. Those auxiliary nodes whose indices have the same t leftmost digits are placed in the same group. The auxiliary nodes of T_L can, likewise, be classified into groups based also on whether their indices have the same t leftmost digits. We will derive in Section III-D, in particular, Lemma 3, the number of groups that we need for each tree to achieve a given girth g, i.e., what is the relationship between t and g.

After clustering the auxiliary nodes into groups, we further let the shift S to be the same when we connect the auxiliary nodes of T_U in the same group to the auxiliary nodes of T_L in the same group. Denote by $S_{\alpha,\beta}$ the shift introduced when we connect the auxiliary nodes of T_U in the αth group to the auxiliary nodes of T_L in the βth group. For different α and β, the shifts $S_{\alpha,\beta}$ may be the same or different from each other. The mapping rule for the interleaver is now the following.

Connection rule to avoid type II cycles Connect the auxiliary node indexed by X_{p-q} in the αth group in T_U to the auxiliary node indexed by $\pi_d(X_{p-q}) \dot{+} S_{\alpha,\beta}$ in the βth group in T_L.

We will show that in fact this rule prevents short type II cycles. But, first, we need to make sure that using this rule does not introduce short type I cycles. This is settled in the next theorem that shows that type I cycles do not depend on the shift $S_{\alpha,\beta}$.

Theorem 2: Connecting the auxiliary node indexed by X_{p-q} in the αth group in T_U to the auxiliary node indexed by $\pi_d(X_{p-q}) \dot{+} S_{\alpha,\beta}$ in the βth group in T_L guarantees that any type I cycle formed is at least of length $2h$, where h denotes the number of tiers in T_U.

The proof of Theorem 2 is similar to that of Theorem 1 and is omitted here.

By Theorem 2, we are free to choose any shift $S_{\alpha,\beta}$ in the rule $X_{p-q} \to \pi_d(X_{p-q}) \dot{+} S_{\alpha,\beta}$ because the length of any type I cycle is guaranteed to be greater than or equal to $2h$. The freedom to choose the values of $S_{\alpha,\beta}$ is exploited to avoid short type II cycles. This is the subject of Theorem 3 and Section III-C2.

We observe that each type II cycle with $2k$ edges in the interleaver is associated with $2k$ shifts

$$S_{\alpha_1,\beta_1}, S_{\alpha_2,\beta_2}, \ldots, S_{\alpha_{2k},\beta_{2k}}.$$

We say that the type II cycle is characterized by the shift sequence $\boldsymbol{A} = \{S_{\alpha_1,\beta_1}, S_{\alpha_2,\beta_2}, \ldots, S_{\alpha_{2k},\beta_{2k}}\}$. The index labels of the shift sequence characterizing a type II cycle satisfy the following two conditions:

(i) $\alpha_{2t-1} \neq \alpha_{2t}$, $t = 1, 2, \ldots, k$ and $\beta_{2t-1} = \beta_{2t}$, $t = 1, 2, \ldots, k$;
(ii) $\alpha_{2t} = \alpha_{2t+1}$, $t = 1, 2, \ldots, k-1$ and $\alpha_{2k} = \alpha_1$ and $\beta_{2t} \neq \beta_{2t+1}$, $t = 1, 2, \ldots, k-1$ and $\beta_{2k} \neq \beta_1$.

For example, the two type II cycles of length six on the bottom left and bottom right of Fig. 7 contain four edges in the interleaver. They are characterized by the same shift sequence

$$\{S_{\alpha_1,\beta_1}, S_{\alpha_2,\beta_1}, S_{\alpha_2,\beta_2}, S_{\alpha_1,\beta_2}\}.$$

The type II cycle shown on the top of Fig. 7 contains six edges in the interleaver; it is characterized by the shift sequence

$$\{S_{\alpha_1,\beta_1}, S_{\alpha_2,\beta_1}, S_{\alpha_2,\beta_2}, S_{\alpha_3,\beta_2}, S_{\alpha_3,\beta_3}, S_{\alpha_1,\beta_3}\}.$$

Given a shift sequence $\boldsymbol{A} = \{S_{\alpha_1,\beta_1}, S_{\alpha_2,\beta_2}, \ldots, S_{\alpha_{2k},\beta_{2k}}\}$ that satisfies conditions (i) and (ii) above, we define further its accumulated alternate sum Δ_A to be

$$\begin{aligned}\Delta_A &= \sum_{i=1}^{2k} (-1)^{i+1} S_{\alpha_i,\beta_i} \\ &= S_{\alpha_1,\beta_1} \dot{-} S_{\alpha_2,\beta_2} \ldots \dot{+} S_{\alpha_{2i-1},\beta_{2i-1}} \dot{-} S_{\alpha_{2i},\beta_{2i}} \\ &\qquad \ldots \dot{+} S_{\alpha_{2k-1},\beta_{2k-1}} \dot{-} S_{\alpha_{2k},\beta_{2k}}. \end{aligned} \tag{6}$$

The following theorem helps to eliminate type II cycles with $2k$ edges in the interleaver.

Theorem 3: Let $\boldsymbol{A} = \{S_{\alpha_1,\beta_1}, S_{\alpha_2,\beta_2}, \ldots, S_{\alpha_{2k},\beta_{2k}}\}$ be a shift sequence that contains $2k$ shifts.

$$\Delta_A = \sum_{i=1}^{2k} (-1)^{i+1} S_{\alpha_i,\beta_i}$$

is the accumulated alternate sum of $\boldsymbol{A}$ and has h digits in its $q-p$-alternate-decimal expansion. If Δ_A contains at most $d = h - l$ consecutive digits "0" in its $q-p$-alternate-decimal expansion, then any type II

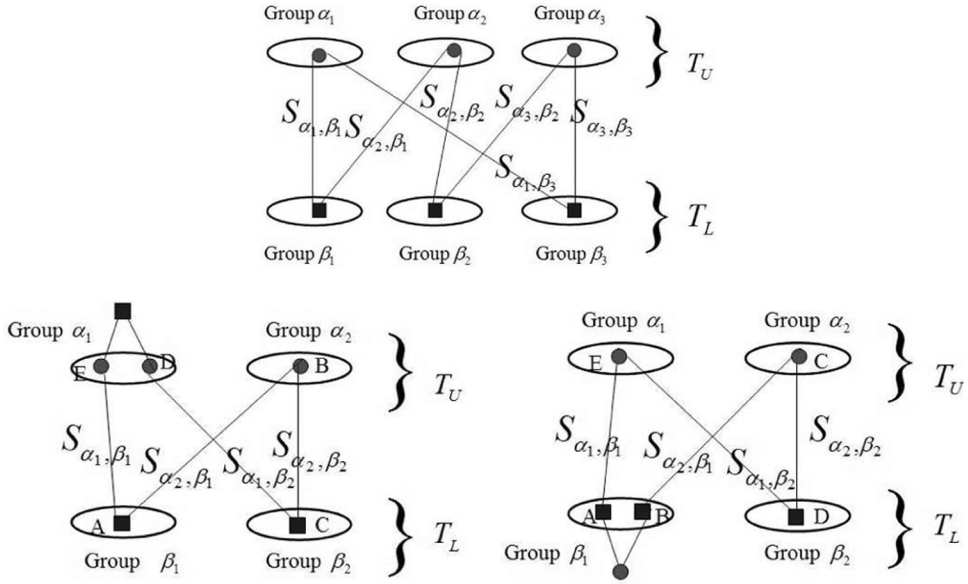

Fig. 7. Three different type II cycles of length 6.

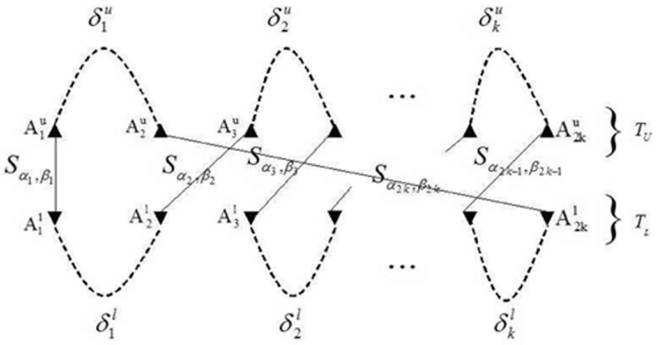

Fig. 8. Type II cycle with $2k$ edges in the interleaver.

cycle characterized by $\boldsymbol{A}$ has length NO less than $2(l+k)$.

Proof: We prove Theorem 3 by proving an equivalent proposition: If there exists a type II cycle with $2k$ edges in the interleaver and its length is less than $2(l+k)$, then the associated $\Delta_A = \sum_{i=1}^{2k}(-1)^{i+1}S_{\alpha_i,\beta_i}$ MUST contain more than $d = h - l$ consecutive digits "0." Fig. 8 shows a type II cycle with $2k$ edges in the interleaver. Let this type II cycle contain $\mathcal{N}_U$ edges in T_U and $\mathcal{N}_L$ edges in T_L.

By assumption, the length of this type II cycle is less than $2(l+k)$. Since the length of the type II cycle is $\mathcal{N}_U + \mathcal{N}_L + 2k$, then

$$\mathcal{N}_U + \mathcal{N}_L < 2l. \tag{7}$$

Since there are $2k$ edges in the interleaver, the cycle contains $4k$ auxiliary nodes, $2k$ auxiliary nodes $A_1^u, A_2^u, \ldots, A_{2k}^u$ in T_U, and $2k$ auxiliary nodes $A_1^l, A_2^l, \ldots, A_{2k}^l$ in T_L. With reference to the plot in Fig. 8, and assuming that the index for the auxiliary node A_1^u is $X_{A_1^u}$, according to the connecting rule presented in Theorem 2, the index for the auxiliary node $X_{A_1^l}$ is $X_{A_1^l} = \pi_d(X_{A_1^u}) \dot{+} S_{\alpha_1,\beta_1}$. Let δ_1^l denote the difference between $X_{A_2^l}$ and $X_{A_1^l}$, i.e., $\delta_1^l = X_{A_2^l} \dot{-} X_{A_1^l}$. The index for the auxiliary node A_3^u is $X_{A_3^u} = \pi_d(X_{A_2^l} \dot{-} S_{\alpha_2,\beta_2})$. Again, let $\delta_2^u = X_{A_4^u} \dot{-} X_{A_3^u}$. The index for the auxiliary node A_3^l is $X_{A_3^l} = \pi_d(X_{A_4^u}) \dot{+} S_{\alpha_3,\beta_3}$.

Continuing to trace the cycle and finding the relationships between the indices of the auxiliary nodes, when we reach the auxiliary node A_2^u, we find that $X_{A_2^u} = \pi_d(X_{A_{2k}^l} \dot{-} S_{\alpha_{2k},\beta_{2k}})$. The relationship between $X_{A_1^u}$ and $X_{A_2^u}$ is $X_{A_1^u} = X_{A_2^u} + \delta_1^u$. Iterating in the definition of $X_{A_2^u}$, we have

$$\sum_{i=1}^{k}(-1)^{i+1}S_{\alpha_i,\beta_i} \dot{+} \pi_d\left(\sum_{s=1}^{k}\delta_s^u\right) \dot{+} \sum_{t=1}^{k}\delta_t^l = 0. \tag{8}$$

Since the cycle has $\mathcal{N}_L$ edges in T_L, then the distance between auxiliary nodes A_{2t-1}^l and A_{2t}^l, $t = 1, 2, \ldots, k$, through T_L is less than or equal to $\mathcal{N}_L$. By Lemma 1, we know that only the rightmost $\frac{\mathcal{N}_L}{2}$ digits of δ_t^l, $t = 1, 2, \ldots, k$, can be nonzero, the other digits of δ_t^l have to be zero. Note that, for the digit-wise add $\dot{+}$, there is no carry. Therefore, only the rightmost $\frac{\mathcal{N}_L}{2}$ digits of $\sum_{t=1}^{k}\delta_t^l$ can be nonzero. Similarly, we derive that only the rightmost $\frac{\mathcal{N}_U}{2}$ digits of $\sum_{s=1}^{k}\delta_s^u$ can be nonzero. From the definition of the digit-wise reversal $\pi_d(\cdot)$, only the leftmost $\frac{\mathcal{N}_U}{2}$ digits of $\pi_d(\sum_{s=1}^{k}\delta_s^u)$ can be nonzero. According to (8), we derive that only the leftmost $\frac{\mathcal{N}_U}{2}$ digits and the rightmost $\frac{\mathcal{N}_L}{2}$ digits of Δ_A can be nonzero. That means that the intermediate $h - \frac{\mathcal{N}_L}{2} - \frac{\mathcal{N}_U}{2}$ digits of Δ_A are zero. Since by (7) $\mathcal{N}_U + \mathcal{N}_L < 2l$, then Δ_A contains more than $h - \frac{\mathcal{N}_L}{2} - \frac{\mathcal{N}_U}{2} > h - \frac{2l}{2} = h - l$ consecutive zeros. Thus, if a cycle has $2k$ edges in the interleaver and its length is less than $2(l+k)$, then its associated Δ_A contains more than $h - l$ consecutive zeros in its $q-p$-alternate-decimal representation. This completes the proof. □

D. Minimum Number of Groups in Each Tree

There is a tradeoff when deciding the number of groups to choose in each tree. On the one hand, to get compact TS-LDPC codes, we prefer a small number of groups in T_U and T_L. To reduce the number of groups, the number t of the common

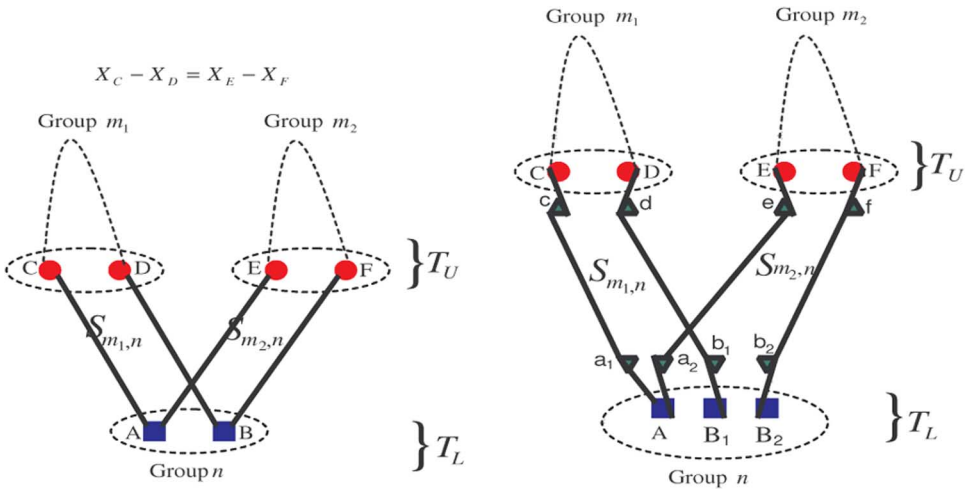

Fig. 9. Left: A type II cycle that can **NOT** be excluded by any choice of the shifts $S_{m_1,n}$ and $S_{m_2,n}$. Right: A subsidiary figure with auxiliary nodes to prove that the cycle shown on the left always exists.

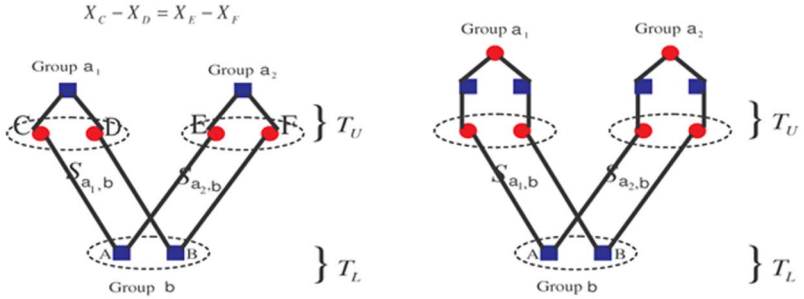

Fig. 10. Left: A length 8 cycle that can **NOT** be excluded by any choice of the shifts $S_{a_1,b}$ and $S_{a_2,b}$. Right: A length 12 cycle that can **NOT** be excluded by any choice of the shifts $S_{a_1,b}$ and $S_{a_2,b}$.

leftmost digits of the indices of the auxiliary nodes in the same group should be as small as possible. On the other hand, a smaller number of groups decreases the number of free parameters in the code design. We will see that a specific class of type II cycles constrains how small the minimum number of groups in each tree can be.

Lemma 2: The cycle shown on the left in Fig. 9 always exists for any possible values of the related shifts $S_{m_1,n}$ and $S_{m_2,n}$ when $X_C - X_D = X_E - X_F$. (X_C, X_D, X_E, and X_F are indices of the bit nodes C, D, E, and F, respectively.)

Proof: As shown on the right in Fig. 9, we add auxiliary nodes c, d, e, f, a_1, a_2, b_1, and b_2 to the cycle shown on the left in Fig. 9. Let X_c, X_d, X_e, X_f, X_{a_1}, X_{a_2}, X_{b_1}, and X_{b_2} be the indices of the auxiliary nodes c, d, e, f, a_1, a_2, b_1, and b_2, respectively. Suppose that the cycle shown on the left in Fig. 9 does not exist for certain values of the shifts $S_{m_1,n}$ and $S_{m_2,n}$. With reference to the plot on the right in Fig. 9, the two check nodes B_1 and B_2 should be different. Similar to (8), we derive that

$$(S_{m_1,n} \dot{-} S_{m_1,n}) \dot{+} (S_{m_2,n} \dot{-} S_{m_2,n}) \dot{+} \pi_d \left(\delta_1^u \dot{+} \delta_2^u\right) \dot{+} \left(\delta_1^l \dot{+} \delta_2^l\right) = 0 \quad (9)$$

where $\delta_1^u = X_c \dot{-} X_d$, $\delta_2^u = X_f \dot{-} X_e$, $\delta_1^l = X_{a_1} \dot{-} X_{a_2}$, and $\delta_2^l = X_{b_2} \dot{-} X_{b_1}$. Equation (9) can be simplified as

$$0 \dot{+} \pi_d \left(\delta_1^u \dot{+} \delta_2^u\right) \dot{+} \left(\delta_1^l \dot{+} \delta_2^l\right) = 0. \quad (10)$$

Since $X_C - X_D = X_E - X_F$ and c, d, e, f are auxiliary nodes of C, D, E, F, respectively, we derive that $X_c \dot{-} X_d$ and $X_e \dot{-} X_f$ are different only in the rightmost digit. Moreover, since c, d, e, f are connected to the same check-node group as shown on the right in Fig. 9, the rightmost digits of c, d, e, f are also the same. Hence, by the above reasoning

$$X_c \dot{-} X_d = X_e \dot{-} X_f. \quad (11)$$

By (11), we derive that

$$\delta_1^u = X_c \dot{-} X_d = -(X_f \dot{-} X_e) = -\delta_2^u. \quad (12)$$

Combining (10) and (12), we derive that

$$\delta_1^l \dot{+} \delta_2^l = 0. \quad (13)$$

Since a_1 and a_2 are connected to the same check node A, X_{a_1} and X_{a_2} are different only in the rightmost digit by definition. Therefore, all the digits of $\delta_1^l = X_{a_1} \dot{-} X_{a_2}$ are zeros except for the rightmost one. Since $\delta_1^l \dot{+} \delta_2^l = 0$ by (13), $\delta_2^l = X_{b_2} \dot{-} X_{b_1}$ should also have all zero digits except for the rightmost digit. Hence, X_{b_1} and X_{b_2} are different only in the rightmost digit, which implies that the two auxiliary nodes b_1 and b_2 are children of the same check node. However, as shown on the right in Fig. 9, b_1 is connected to B_1 and b_2 is connected to B_2. Hence, B_1 and B_2 are the same check node. This contradicts the assumption that B_1 and B_2 are different. Therefore, the cycle shown on the left in Fig. 9 always exists for any choices of the shifts $S_{m_1,n}$ and $S_{m_2,n}$ when $X_C - X_D = X_E - X_F$. This completes the proof. □

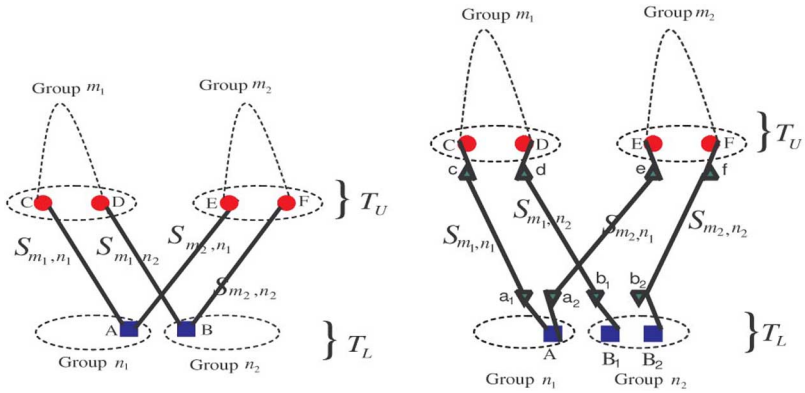

Fig. 11. Left: A type II cycle that **CAN** be excluded by choosing appropriate shifts S_{m_1,n_1}, S_{m_1,n_2}, S_{m_2,n_1}, and S_{m_2,n_2}. Right: An auxiliary figure to show that the cycle shown on the left **CAN** be excluded.

Fig. 10 shows two specific examples of the cycle described in Lemma 2 . The left plot shows a length 8 cycle that cannot be excluded by choosing appropriate shifts $S_{a_1,b}$ and $S_{a_2,b}$; the right plot shows a length 12 cycle that cannot be excluded by choosing appropriate shifts $S_{a_1,b}$ and $S_{a_2,b}$.

To eliminate the cycle shown on the left in Fig. 9, we introduce more subgroups to divide the two check nodes A and B into two different groups, as shown on the left in Fig. 11. Similar to the proof of Lemma 2 and with reference to the right plot in Fig. 11, we derive that

$$(S_{m_1,n_1} \dot{-} S_{m_2,n_1} \dot{+} S_{m_2,n_2} \dot{-} S_{m_1,n_2}) \dot{+} \pi_d\left(\delta_1^u \dot{+} \delta_2^u\right) \dot{+} \left(\delta_1^l \dot{+} \delta_2^l\right) = 0. \quad (14)$$

Since $\delta_1^u = -\delta_2^u$ by (12), (14) can be simplified as

$$X_{b_1} - X_{b_2} = (S_{m_1,n_1} \dot{-} S_{m_2,n_1} \dot{+} S_{m_2,n_2} \dot{-} S_{m_1,n_2}) \dot{+} \delta_1^l. \quad (15)$$

As analyzed before, δ_1^l has only one nonzero digit—the rightmost one. However, this time, we have the freedom to choose appropriate values of the shifts S_{m_1,n_1}, S_{m_2,n_1}, S_{m_2,n_2}, and S_{m_1,n_2} to make their summation having nonzero digits other than the rightmost one, e.g., the second to the rightmost digit. Hence, by (15), $X_{b_1} - X_{b_2}$ can be made to have nonzero digits other than the rightmost one, which means that the two check nodes B_1 and B_2 can be different. It follows that the cycle shown on the left in Fig. 11 can be avoided by carefully choosing shifts S_{m_1,n_1}, S_{m_2,n_1}, S_{m_2,n_2}, and S_{m_1,n_2}. This shows that, to achieve a desired girth g, we should require that the number of groups in which we divide the auxiliary nodes of T_U and T_L be larger than a certain minimum, which is a function of the girth g.

We now consider the question of determining the minimum number of groups to achieve the girth g and how this relates to the parameter t in Section III-C. Let G_U and G_L represent the minimum number of groups needed for T_U and T_L, respectively. Then we have the following lemma.

Lemma 3: To achieve a girth g, the minimum number of groups G_U and G_L are

$$G_U \geq ((k-1)(j-1))^{\left\lfloor \frac{(g-2)/4}{2} \right\rfloor} (k-1)^{\left\lfloor \frac{(g-2)}{4} \right\rfloor \bmod 2} \quad (16)$$

$$G_L \geq ((k-1)(j-1))^{\left\lfloor \frac{(g-2)/4}{2} \right\rfloor} (j-1)^{\left\lfloor \frac{(g-2)}{4} \right\rfloor \bmod 2}. \quad (17)$$

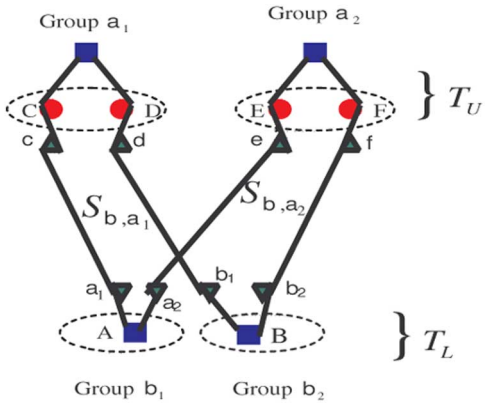

Fig. 12. Divide check nodes A and B into subgroups to avoid a length 8 cycle.

Proof: Let us first look at an example. As analyzed before, the length 8 cycle shown on the left in Fig. 10 cannot be avoided when the two check nodes A and B are in the same group. To avoid this length 8 cycle, A and B must be in two different groups, as shown in Fig. 12. Since A connects to the auxiliary node c and B connects to the auxiliary node d in Fig. 12, c and d must connect to check nodes in two different groups. Further, since c and d can be any two auxiliary nodes whose indices are different only in the two rightmost digits, we conclude that any two auxiliary nodes whose two rightmost digits are different should be connected to different groups. Since there are $(j-1)(k-1)$ categories of such auxiliary nodes, we need at least $(j-1)(k-1)$ groups in T_L. Similarly, to avoid the length 12 cycle shown on the right in Fig. 10, we need at least $(j-1)^2(k-1)$ groups in T_L.

More generally, to avoid the cycle with length $L = 4d+4$ as shown on the left in Fig. 9, we derive the following. When d is odd, at least $G_L = (j-1)^{\frac{(d+1)}{2}}(k-1)^{\frac{(d+1)}{2}}$ groups in T_L are needed; when d is even, at least $G_L = (j-1)^{\frac{d}{2}+1}(k-1)^{\frac{d}{2}}$ groups in T_L are necessary.

The above relationship can be compactly written as

$$G_L \geq ((k-1)(j-1))^{\left\lfloor \frac{(g-2)/4}{2} \right\rfloor} (j-1)^{\lfloor (g-2)/4 \rfloor \bmod 2}. \quad (18)$$

Now we study the number of groups needed for T_U. To avoid the cycle shown on the left in Fig. 13, we divide the two bit

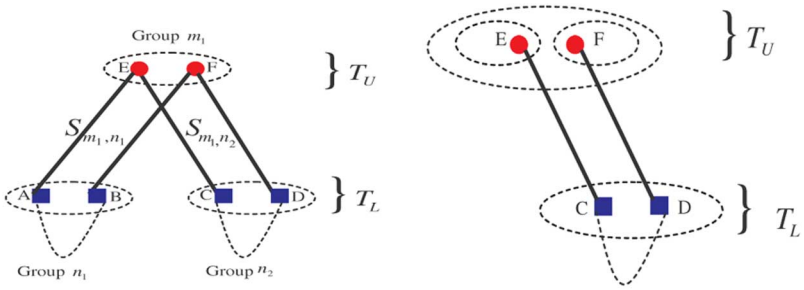

Fig. 13. Left: A type of cycle that can NOT be excluded by choices of shifts S_{m_1,n_1} and S_{m_1,n_2} Right: Divide bit nodes E and F into subgroups to avoid cycles.

nodes E and F into different subgroups, as shown on the right in Fig. 13. Symmetrically, we derive that

$$G_U \geq ((k-1)(j-1))^{\left\lfloor \frac{(g-2)/4}{2} \right\rfloor}(k-1)^{\lfloor (g-2)/4 \rfloor \bmod 2} \quad (19)$$

where g is the girth to be achieved and G_U is the minimum number of groups needed for T_U. This completes the proof. □

On the other hand, if the indices of the auxiliary nodes in the same group must have t leftmost digits in common, G_U and G_L are given by

$$G_U = ((k-1)(j-1))^{\lfloor \frac{t}{2} \rfloor}(k-1)^{t \bmod 2} \quad (20)$$

$$G_L = ((k-1)(j-1))^{\lfloor \frac{t}{2} \rfloor}(j-1)^{t \bmod 2}. \quad (21)$$

So, to achieve girth g, the parameter t that determines the number of groups has to satisfy

$$t \geq \left\lfloor \frac{g-2}{4} \right\rfloor. \quad (22)$$

Equation (22) determines the minimum value of the parameter t to achieve a girth g.

E. Construction of TS-LDPC Codes

We use Theorem 3 to reduce the construction of TS-LDPC codes with desired girth g to designing a matrix $\boldsymbol{S}$ that collects appropriate shifts $S_{\alpha,\beta}$. By choosing suitably these shifts $S_{\alpha,\beta}$ according to Theorem 3, we can avoid all short type I and type II cycles up to the desired length $g-2$. We present next an algorithm that finds $\boldsymbol{S} = [S_{\alpha,\beta}]$ for TS-LDPC codes with girth g. The matrix $\boldsymbol{S}$ is $G_U \times G_L$, which is much smaller than the TS-LDPC code parity-check matrix $\boldsymbol{H}$. The algorithm is greedy; we determine shifts $S_{\alpha,\beta}$, one at a time, its value being strongly dependent on the previously determined shifts. Different initial settings of $S_{\alpha,\beta}$ will lead to different matrices $\boldsymbol{S}$. If the algorithm fails to generate a matrix $\boldsymbol{S}$, it is restarted with different initial settings. To construct a TS-LDPC code with column weight j, row weight k, and tier h (number of tiers contained in the upper tree T_U), the number of candidate matrices $\boldsymbol{S}$ is $[(j-1)(k-1)]^{\frac{h}{2}\cdot G_U \cdot G_L}$, which is exponential in the number of groups G_U and G_L. Large girth g may require increasing the number of tiers h in the upper and lower trees.

As an illustration, we applied Algorithm 1 to construct a $(6666, 3, 6)$ regular LDPC code, with rate $r = 0.5$ and girth $g = 10$. Its structure is given by the 3333×6666 matrix $\boldsymbol{H}$ shown in Fig. 3.

We can clearly identify T_U, T_L, and the interleaver component $\mathcal{I}$ from the constructed matrix, as labeled in Fig. 3. In this matrix, along the solid lines, there is a single 1 in each row, while along the dashed thicker diagonals there are five 1's in each row, so that per row there are six 1's.

We have the following relation between the column weight j, the row weight k, the girth g, and the code block length n of TS-LDPC codes that we construct when the girth g is small:

$$n = \frac{k\left\{[(k-1)(j-1)]^{\frac{(g-2)}{2}} - 1\right\}}{[(k-1)(j-1)] - 1}. \quad (23)$$

When the girth g is large, we need to apply Algorithm 1 to find the code block length n required.

Algorithm 1 TS-LDPC codes with girth g

Initialization Set matrix $\boldsymbol{S} = \phi$, the empty matrix.
Determine the elements of the matrix $\boldsymbol{S}$ row by row.
$S_{1,1} \leftarrow \text{rand}(\cdot)$. Set $\alpha = 1$ and $\beta = 2$
step a: $S_{\alpha,\beta} \leftarrow \text{rand}(\cdot)$ and set its flag to 0.
for $t = 2$ to $\frac{g-2}{2}$ **do**
  for all closed paths of length $2t$ in the current entries of the shift matrix $\boldsymbol{S}$ that pass the entry $S_{\alpha,\beta}$ **do**
    Check if $\sum_{i=1}^{2t}(-1)^{i+1}S_{\alpha_i,\beta_i}$ contains more than $h + t - \frac{g}{2}$ consecutive zeros $\{S_{\alpha_1,\beta_1}, S_{\alpha_2,\beta_2}, \ldots, S_{\alpha_{2t},\beta_{2t}}$ are the $2t$ consecutive corners of the closed path considered$\}$
    if $\sum_{i=1}^{2t}(-1)^{i+1}S_{\alpha_i,\beta_i}$ contains more than $h + t - \frac{g}{2}$ consecutive zeros **then**
      set the flag to be 1 and stop the for loop;
    else
      keep the flag to 0.
    end if
  end for
end for
if the value of the flag is 1 **then**
  discard the current candidate for $S_{\alpha,\beta}$, go back to **step a** to select another possible value for $S_{\alpha,\beta}$.
else
  fill the entry (α, β) of $\boldsymbol{S}$ with the current value $S_{\alpha,\beta}$. Set the values of α and β to the next element of $\boldsymbol{S}$ that is to be determined.
  if all the elements of $\boldsymbol{S}$ have already been properly chosen **then**
    go to **step b**
  else
    go back to **step a**
  end if
end if
step b: End, output the shift matrix $\boldsymbol{S}$

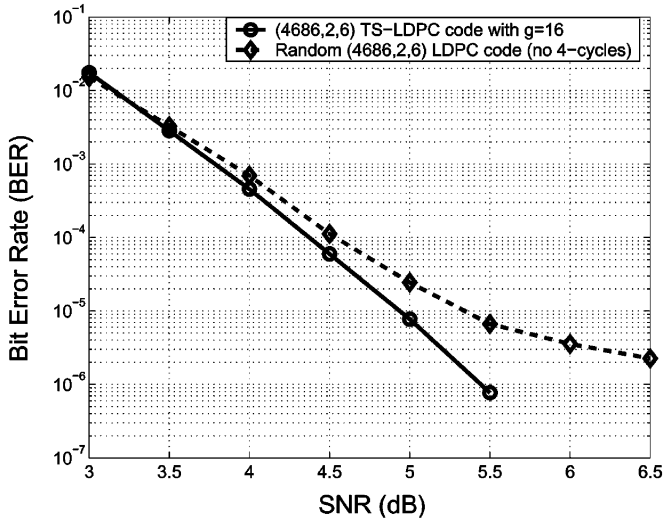

Fig. 14. Comparison of BER performance for a $(4686, 2, 6)$ TS-LDPC code with girth 16, rate $2/3$, and a randomly constructed $(4686, 2, 6)$ LDPC code (no 4-cycles).

F. Efficient Memory Utilization

In general, an (n, j, k)-LDPC code is represented by an $m \times n$ parity-check matrix $\boldsymbol{H}$. Its efficient storage records only the nonzero column indices in each row, hence, at least $n \times j$ indices needs to be stored. In contrast, for a TS-LDPC code we need only to store its small shift generating matrix $\boldsymbol{S}$. For example, for the 3333×6666 matrix with girth $g = 10$ shown in Fig. 3, according to (16) and (17), $G_U = G_L = 10$, and so the shift matrix $\boldsymbol{S}$ is 10×10. Hence, instead of storing the $6666 \times 3 = 19998$ column indices required for generic LDPC codes, TS-LDPC require only storing $10 \times 10 = 100$ shifts, reducing the memory by a factor of 200.

IV. Performance Evaluation

We compare by simulation the BER of the TS-LDPC codes with the BER of randomly constructed LDPC codes that are free of 4-cycles [32] in additive white Gauss noise (AWGN) channels. The codes are decoded with the sum–product algorithm [33]. We adopt the rate normalized SNR defined in [32]: $\text{SNR} = 10 \log_{10} \left[E_b / \left(2r\sigma^2 \right) \right]$, where r denotes the code rate.

We first compare the performance of TS-LDPC codes and random codes free of 4-cycles with column weight $j = 2$. We consider TS-LDPC codes with girth $g = 16$. Fig. 14 compares the BER performance for two $(n = 4686, 2, 6)$ LDPC codes with rate $r = 2/3$: a random (4-cycle free) and a TS-LDPC code of girth $g = 16$. In the high-SNR region, the TS-LDPC code outperforms the random code: at BER $= 2 \times 10^{-6}$ this gain is 1.2 dB. In the low-SNR region, the TS-LDPC code has performance comparable to that of the random code. The slope of the BER curve for the random LDPC code decreases with the SNR in the high-SNR region, implying that for this code the error floor occurs when the BER reaches 10^{-6}, which is not the case for the girth 16 TS-LDPC code.

For LDPC codes with column weight $j = 2$, we can derive that the minimum distance $d_{\min} = g/2$, where g is the girth. For the TS-LDPC code with girth $g = 16$, $d_{\min} = 8$; for the random LDPC code that is free of 4-cycles, since the girth g is only 6, $d_{\min} = 3$. In the high-SNR region, $d_{\min}$ is a dominant factor in determining the code BER performance. Therefore, TS-LDPC codes with girth 16 outperform random codes when the SNR is high. This is in agreement with our simulation studies.

We now study column weight $j = 3$ codes. Fig. 15 shows the BER performance for a column weight $j = 3$ TS-LDPC code with girth 8. For comparison, we also show the BER performance of a randomly constructed LDPC code (no 4-cycles) with column weight $j = 3$ (dashed line). Both codes have the same block length 6084 and the same code rate $3/4$. Since there exist many explicitly constructed LDPC codes with high girth, we also incorporate a $(6108, 3, 12)$ LDPC code constructed from rectangular integer lattices [20] in our simulations. In particular, the constructed LDPC code is based on the lattice construction of 2-$(v, k, 1)$ 1-configurations [20] and has girth 8.

Fig. 15 shows that the BER performance of the TS-LDPC code outperforms that of the random LDPC code at BER $< 10^{-5}$ while at low SNR, both codes have identical error-correcting performance. The BER performance of the TS-LDPC code is also slightly better than that of the LDPC code constructed from rectangular integer lattices at high SNR.

The construction of TS-LDPC codes is based on interleaving edges between the leaf nodes of upper and lower trees. Cycles are avoided by controlling the interleaver. The variables in the upper and lower trees are strongly connected only through leaf nodes. During the decoding process, the computation of the likelihood for non-leaf nodes of the upper and lower trees depends on the information from the leaf nodes of the trees. It is interesting to check the error events of the iterative decoder to see if specific error concentrations in the non-leaf nodes occur. This is not supported by the experimental evidence shown in Fig. 16 that plots these BER for the $(6084, 3, 12)$ TS-LDPC code constructed above. This figure shows that the BER for non-leaf nodes is slightly better than that for leaf nodes. The figure also shows that the BER for undetected codeword errors is much lower than the BER for non-leaf and leaf nodes, as expected.

We study now TS-LDPC codes with girth 10. The plot in Fig. 17 shows the BER performance for a column weight $j = 3$ TS-LDPC code with girth 10. For comparison, we show the BER performance of a randomly constructed LDPC code (no 4-cycles) with column weight $j = 3$ (dashed line) as well. Both codes have the same block length 6666 and the same code rate $1/2$. We also study by simulation three other structured regular LDPC codes: finite-geometry LDPC code [7]; LU$(3, q)$ LDPC code [19]; and GQC-LDPC code [17], [23]–[25]. The finite-geometry LDPC code we use is an extended code constructed from the TYPE-I 2-D EG-LDPC code. The code constructed has code block length 8190 and code rate 0.5. LU$(3, q)$ LDPC code constructed has code block length 6859, girth 8, and code rate 0.4514. The GQC-LDPC code constructed has code block length 6666, girth 10, and code rate 0.5.

Fig. 17 shows that the BER performance of the TS-LDPC code is 0.12 dB better than that of the random LDPC code at BER $= 10^{-5}$, while at low SNR, both codes have similar error-correcting performance. Using results from [3], we can compute that the $(6666, 3, 6)$ TS-LDPC code has minimum distance $d_{\min} \geq 10$. Since this lower bound on $d_{\min}$ derived in [3] is not tight, the actual $d_{\min}$ of the $(6666, 3, 6)$ TS-LDPC

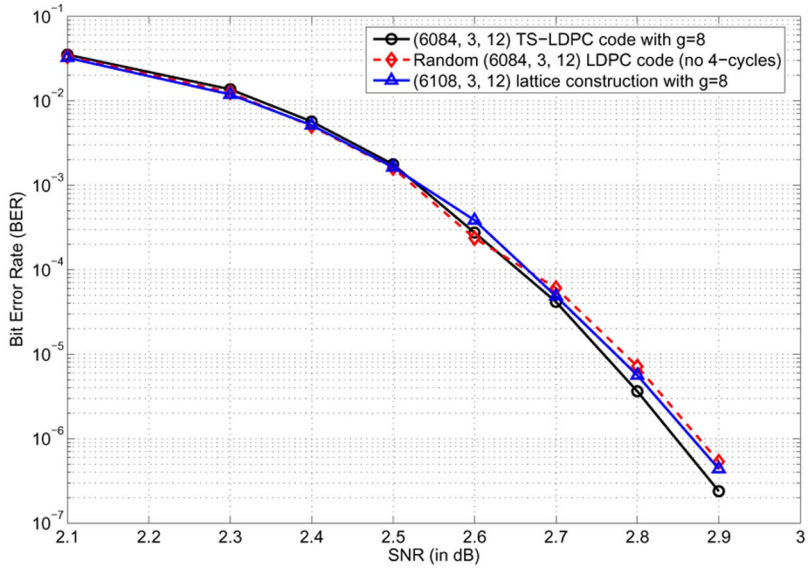

Fig. 15. BER performance comparison between a $(6084, 3, 12)$ TS-LDPC code with girth 8, rate $3/4$, a randomly constructed $(6084, 3, 12)$ LDPC code (no 4-cycles), and a $(6108, 3, 12)$ LDPC code on rectangular integer lattices.

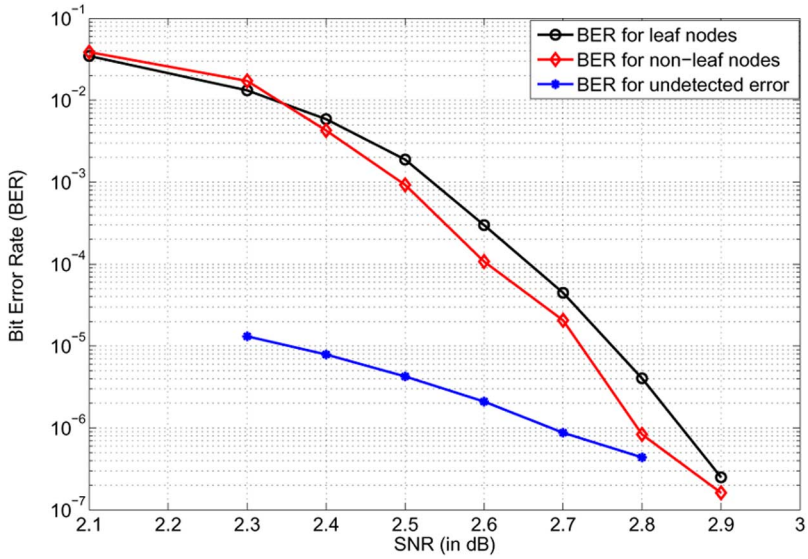

Fig. 16. BER performance of leaf nodes, non-leaf nodes, and undetected error events for a $(6084, 3, 12)$ TS-LDPC code.

code may be much larger than 10. Again, in the high-SNR region, $d_{\min}$ is a dominant factor in determining the code BER performance. This explains why the TS-LDPC code with girth 10 has good BER performance in the high-SNR region. We also notice that at moderate-to-high SNR the TS-LDPC code outperforms the GQC-LDPC code, which suggests that TS-LDPC codes have better BER performances than GQC-LDPC codes with the same code parameters. Further, the TS-LDPC code, finite-geometry LDPC code, and LU$(3, q)$ LDPC code have similar BER performances. The finite-geometry LDPC code and the LU$(3, q)$ LDPC code have good BER performances since they have large minimum distance $d_{\min}$[7], [19].

Again, we compare the BER performance for non-leaf and leaf nodes for the same $(6666, 3, 6)$ TS-LDPC code just discussed. This is shown in Fig. 18 that also plots the undetected error rate. These simulation results confirm the previous observation, shown in Fig. 16 for the $(6084, 3, 12)$ TS-LDPC code, that the BER performance of non-leaf nodes is either comparable or slightly better than the BER performance of leaf nodes.

V. Efficient Encoding for TS-LDPC Codes

TS-LDPC codes, or a variant of TS-LDPC codes, can be encoded with *linear* complexity. We first describe this variant—encoding friendly TS-LDPC (EFTS-LDPC) codes, then we show how EFTS-LDPC codes can be encoded in linear complexity.

A. Encoding Friendly TS-LDPC (EFTS-LDPC) Codes

The Tanner graph of an EFTS-LDPC code still contains an upper tree T_U and a lower tree T_L that are interconnected by

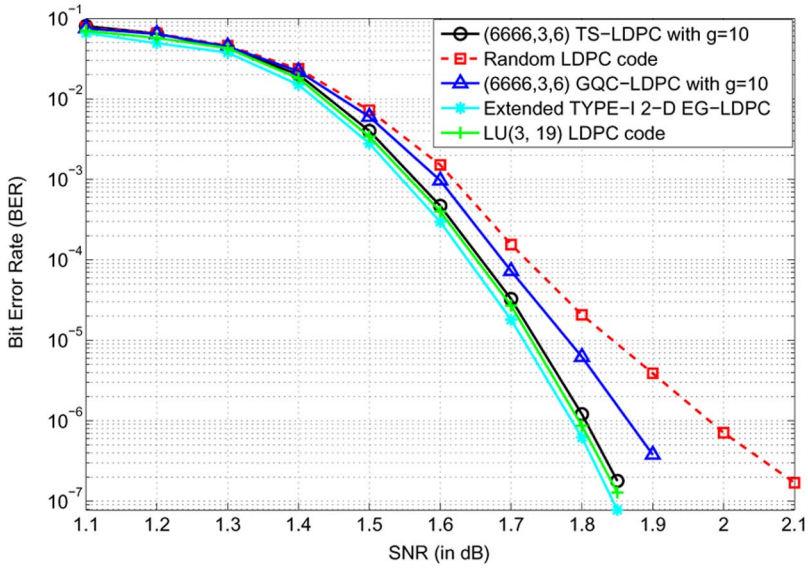

Fig. 17. BER performance comparison between a $(6666, 3, 6)$ TS-LDPC code with girth 10, rate $1/2$, a randomly constructed $(6666, 3, 6)$ LDPC code (no 4-cycles), a $(6666, 3, 6)$ GQC-LDPC code with girth 10, rate $1/2$, an extended type-I 2-D EG-LDPC code with code block length 8190 and code rate 0.5 (LDPC code based on finite geometries), and LU$(3, 19)$ LDPC code with code block length 6859 and code rate 0.4514.

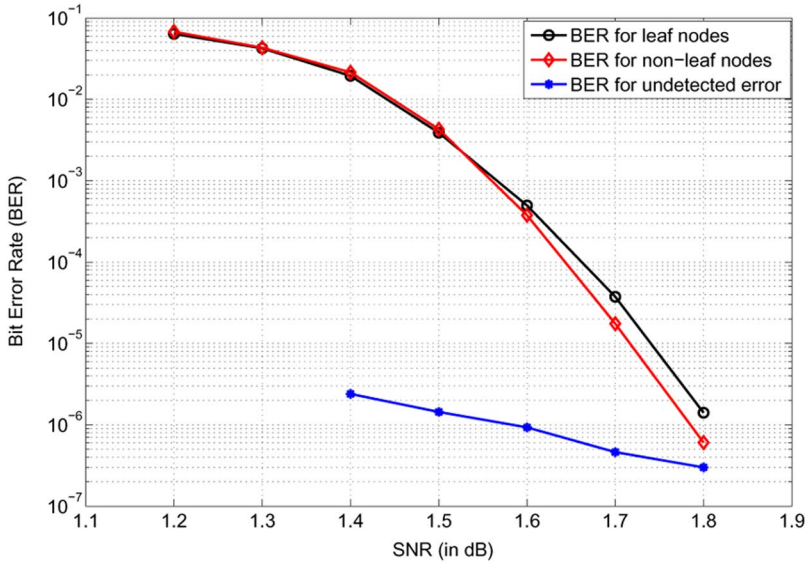

Fig. 18. BER performance of leaf-nodes, nonleaf-nodes, and undetected error events for a (6666,3,6) TS-LDPC code.

an interleaver $\mathcal{I}$. There are no restrictions on the upper tree T_U of the EFTS-LDPC code, which can be exactly the same as the T_U part of the standard TS-LDPC codes. The T_L of the EFTS-LDPC code is restricted so that the degree of its bit nodes is two. In addition, the root of the T_L is changed from a bit node to a check node, as shown in Fig. 19. The Tanner graph for an EFTS-LDPC code is shown in Fig. 19. EFTS-LDPC codes are slightly irregular. These modifications enable EFTS-LDPC codes to be encoded with linear complexity.

B. Linear-Complexity Encoding of EFTS-LDPC Codes

Since an LDPC code is equivalently represented by its Tanner graph $\mathcal{G}$, we explain how to encode EFTS-LDPC codes using their Tanner graph.

To achieve linear-complexity encoding, we need to first remove the root (a check node) of the lower tree T_L from the Tanner graph of the EFTS-LDPC codes. We will show in the

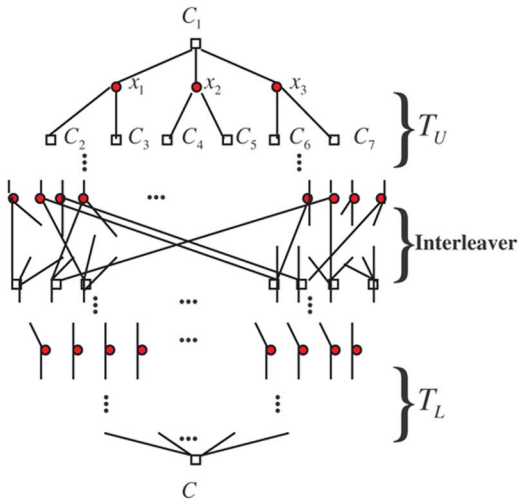

Fig. 19. EFTS-LDPC codes: A variant of TS-LDPC codes (EFTS-LDPC codes).

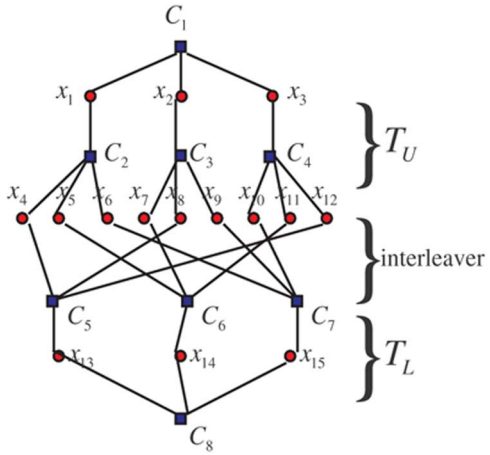

Fig. 20. Tanner graph for an EFTS-LDPC code (The bit node degree of T_U is even).

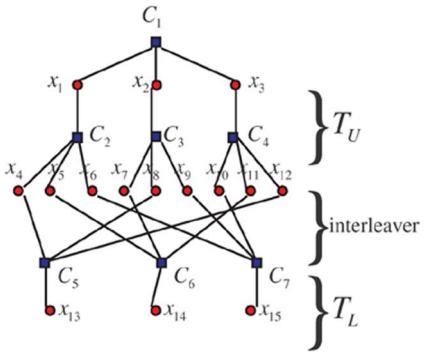

Fig. 21. An equivalent representation of the Tanner graph shown in Fig. 20. (The root of T_L is removed.)

following lemma that removing the root of T_L will not alter the underlying code structure.

Lemma 4: The parity-check equation denoted by the root of T_L is redundant and can be removed from the parity-check matrix $\boldsymbol{H}$ without changing the underlying code structure.

Proof: We discuss two different cases: the bit node degree of T_U is even; the bit node degree of T_U is odd.

1) The Bit Node Degree of T_U Is Even: Since all the bit nodes in T_L have uniform degree two by definition, then the degree of all the bit nodes is even, which means that each column of $\boldsymbol{H}$ contains an even number of 1's. Hence, the sum of all the rows of $\boldsymbol{H}$ in the binary field is a vector of 0's. Therefore, one row of $\boldsymbol{H}$ is linearly dependent on the remaining rows and can be removed without affecting the code. We choose to remove the row that corresponds to the root of T_L. For example, Fig. 20 shows an EFTS-LDPC code. The bit node degree of its T_U is two, an even number. The root of its T_L can be removed to generate an equivalent Tanner graph, as shown in Fig. 21.

2) The Bit Node Degree j of T_U is Odd: By construction, check nodes in T_L connect to either leaf nodes of T_U or bit nodes of T_L. Since each leaf node of T_U is connected to $j-1$ check nodes in T_L and j is an odd number, then each leaf node of T_U is connected to an even number of check nodes in T_L. Further, every bit node in T_L is connected to exactly two check nodes in T_L by construction. Hence, every bit node is connected to an even number of check nodes in T_L. If we sum up those parity-check equations denoted by the check nodes in T_L, the summation in the binary field is a vector of 0's. Therefore, we can remove one of those parity-check equations in T_L without changing the underlying code structure. We again choose to remove the parity-check equation denoted by the root of T_L. For example, Fig. 22 shows an EFTS-LDPC code. The bit node degree of T_U is 3, an odd number. The root of T_L can be removed from its Tanner graph without changing the code, as shown in Fig. 23. This completes the proof. □

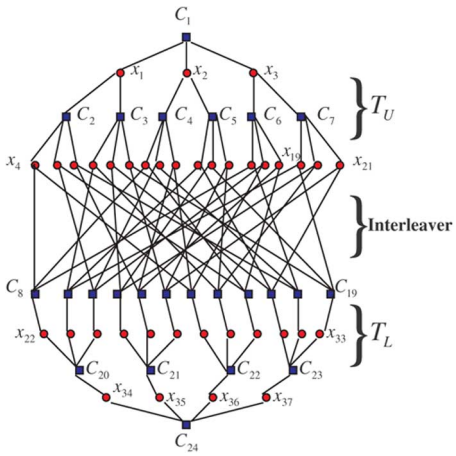

Fig. 22. Tanner graph for an EFTS-LDPC code. (The bit node degree of T_U is odd.)

After removing the root of T_L, we use Algorithm 2 to encode an EFTS-LDPC code.

Algorithm 2 Encoding algorithms for EFTS-LDPC codes (T_L contains h tiers)

Initialization
Encode $\boldsymbol{T_U}$, get the values of all the bit nodes in T_U, including the leaf nodes;
Compute values of the bit nodes in tier $h-1$ of T_L based on the values of the leaf-nodes of T_U;
for $i = h-3$ to 2 step–2 **do**
  Compute values of the bit nodes in tier i of T_L based on the values of bit nodes in tier $i+2$ of T_L;
end for
Output the encoding result.

Next, we show that the encoding complexity of Algorithm 2 is linear in the block length n. We first study the encoding process for the upper tree T_U. The upper tree T_U can be easily encoded in linear time, as shown in the following lemma.

Lemma 5: The upper tree T_U can be encoded in linear complexity.

The proof is straightforward. We omit it here. We look at an example. Fig. 24 shows an upper tree T_U. We encode T_U as follows:

a. Acquire the values of the information bits x_2, x_3, x_5, x_6, x_8, x_9, x_{11}, x_{12}, x_{14}, x_{15}, x_{17}, x_{18}, x_{20}, and x_{21}.
b. Compute the parity bit x_1 from the parity-check equation $C_1 : x_1 = x_2 \oplus x_3$.

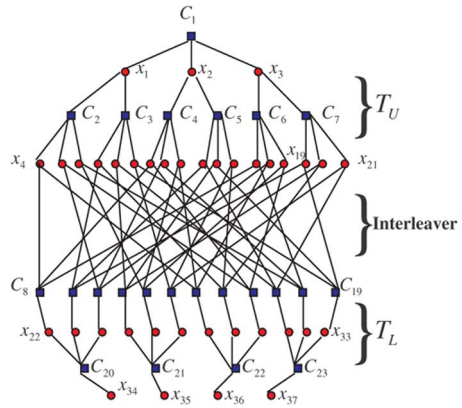

Fig. 23. An equivalent representation of the Tanner graph shown in Fig. 22. (The root of T_L is removed.)

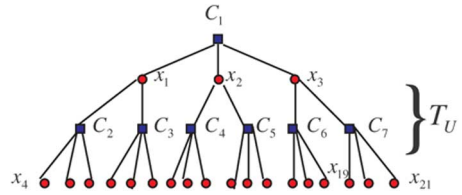

Fig. 24. Upper tree T_U of an EFTS-LDPC code.

c. Compute the parity bits x_4, x_7, x_{10}, x_{13}, x_{16}, and x_{19} from the parity-check equations C_2 to C_7: $x_4 = x_1 \oplus x_5 \oplus x_6$; $x_7 = x_1 \oplus x_8 \oplus x_9$; $x_{10} = x_2 \oplus x_{11} \oplus x_{12}$; $x_{13} = x_2 \oplus x_{14} \oplus x_{15}$; $x_{16} = x_3 \oplus x_{17} \oplus x_{18}$; $x_{19} = x_3 \oplus x_{20} \oplus x_{21}$.

The complexity of the above encoding process is only $7\times2-1 = 13$ XOR operations.

After encoding T_U, we notice that all the bit nodes in T_L represent parity bits. Let h represent the number of tiers in T_L. Since the degree of every bit node x in T_L is two, the value of x depends only on the values of the bit nodes in the lower tier. Particularly, the values of the bit nodes in tier $h-1$ of T_L are based only on the values of the leaf nodes of T_U. Hence, we can compute the values of the bits in T_L tier by tier, starting from the bottom tier. Each time we succeed in obtaining the values of all the bits in a given tier, say, the $(2i)$th tier, we can then compute the values of all the bits in the $(2i-2)$th tier. This encoding process keeps going on until the values of all the bits in the second tier are known (the first tier has been removed by Lemma 4). In this way, we encode all the bits in T_L.

We, again, look at an example. Fig. 23 shows the Tanner graph for an EFTS-LDPC code. Following Algorithm 2, we encode it as follows.

a. Encode the upper tree T_U as shown earlier.
b. Compute the parity bits x_{22} to x_{33} from the parity-check equations C_8 to C_{19}, respectively.
c. Compute the parity bits x_{34}, x_{35}, x_{36}, and x_{37}: $x_{34} = x_{22} \oplus x_{23} \oplus x_{24}$; $x_{35} = x_{25} \oplus x_{26} \oplus x_{27}$; $x_{36} = x_{28} \oplus x_{29} \oplus x_{30}$; $x_{37} = x_{31} \oplus x_{32} \oplus x_{33}$.

We evaluate the computational complexity of Algorithm 2. Let k_l, $l = 1, 2, \ldots, m$, denote the number of bits contained in the lth parity-check equation. Each of the m parity-check equations is used to obtain the value of a parity bit. When employing the lth parity-check equation to determine the value of a parity bit, (k_l-2) XOR operations are needed. So, $\sum_{l=1}^{m}(k_l-2)$ XOR operations are required to obtain all the m parity bits. Let $\overline{k} = \frac{1}{m}\sum_{l=1}^{m} k_l$ denote the average number of bits in the m parity-check equations, then the encoding complexity can be expressed as $\mathcal{O}(m(\overline{k}-2))$. For LDPC codes with uniform row weight k, the encoding complexity is $\mathcal{O}(m(k-2))$. From the above analysis, the encoding process proposed can be accomplished in linear time.

VI. Conclusion

This paper proposes a class of well-structured regular LDPC codes—the turbo-structured (TS-LDPC) codes. We showed through a series of theorems that we can design TS-LDPC codes with arbitrary desired column and row weights j and k, hence, with any practical rate r and arbitrary girth g. TS-LDPC codes can be designed by specifying a shift matrix $\boldsymbol{S}$, a much smaller object than the parity-check matrix $\boldsymbol{H}$, hence, requiring much less memory to store them. TS-LDPC codes with girth $g \geq 8$ have good BER performance, with lower error floor at high SNR than equivalent size and rate 4-cycle-free random codes. We further showed that a variant of TS-LDPC codes—EFTS-LDPC codes—can be encoded efficiently in linear complexity. These characteristics of flexible code rates, arbitrary large girth, good error floor performance, efficient storage, and efficient encoding make TS-LDPC codes attractive for applications such as digital communication systems and data storage systems.

Acknowledgment

The authors would like to thank the anonymous reviewers for their suggestions that significantly improved the presentation of the results in this paper and to Dr. Marc P. C. Fossorier for his leadership with this paper.

References

[1] R. G. Gallager, *Low-Density Parity Check Codes*. Cambridge, MA: MIT Press, 1963.
[2] D. J. C. MacKay and R. M. Neal, "Good codes based on very sparse matrices," in *Cryptography and Coding, 5th IMA Conf. (Lecture Notes in Computer Science)*, C. Boyd, Ed. Berlin, Germany: Springer-Verlag, 1995, vol. 1025, pp. 110–111.
[3] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.
[4] J. Xu, H. Tang, Y. Kou, S. Lin, and K. A. S. Abdel-Ghaffar, "A general class of LDPC finite geometry codes and their performance," in *Proc. IEEE Int. Symp. Information Theory*, Lausanne, Switzerland, Jun./Jul. 2002, p. 309.
[5] J. M. F. Moura, J. Lu, and H. Zhang, "Structured LDPC codes with large girth," *IEEE Signal Process. Mag.*, vol. 21, no. 1, pp. 42–55, Jan. 2004.
[6] B. Vasić and O. Milenkovic, "Combinatorial constructions of low-density parity-check codes for iterative decoding," *IEEE Trans. Inf. Theory*, vol. 50, no. 6, pp. 1156–1176, Jun. 2004.
[7] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: A rediscovery and new results," *IEEE Trans. Inf. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.
[8] ——, "Low density parity check codes: Construction based on finite geometries," in *Proc. IEEE Globecom 2000*, San Francisco, CA, Nov. 2000, vol. 2, pp. 825–829.

[9] J. H. Dinitz and D. R. Stinson, "A brief introduction to design theory," in *Contemporary Design Theory: A Collection of Surveys*, J. H. Dinitz and D. R. Stinson, Eds. New York: Wiley, 1992, ch. 1, pp. 1–12.

[10] D. J. C. MacKay and M. C. Davey, "Evaluation of Gallager codes for short block length and high rate applications, in codes," in *Systems and Graphical Models; IMA Volumes in Mathematics and its Applications*, B. Marcus and J. Rosenthal, Eds. New York: Springer-Verlag, 2000, vol. 123, pp. 113–130.

[11] S. J. Johnson and S. R. Weller, "Construction of low-density parity-check codes from kirkman triple systems," in *Proc. IEEE Globecom 2001*, San Antonio, TX, Nov. 2001, vol. 2, pp. 970–974.

[12] ——, "Regular low-density parity-check codes from combinatorial design," in *Proc. IEEE Information Technology Workshop*, Cairns, Australia, Sep. 2001, pp. 90–92.

[13] B. Vasic, "Structured iteratively decodable codes based on steiner systems and their application in magnetic recording," in *Proc. IEEE Globecom 2001*, San Antonio, TX, Nov. 2001, vol. 5, pp. 2954–2960.

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

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

[16] 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.

[17] J. Lu, J. M. F. Moura, and U. Niesen, "Grouping-and-shifting designs for structured LDPC codes with large girth," in *Proc. IEEE Int. Symp. Information Theory*, Chicago, IL, Jun./Jul. 2004, p. 236.

[18] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, "Progressive edge-growth tanner graphs," in *Proc. IEEE Globecom 2001*, San Antonio, TX, Nov. 2001, pp. 995–1001.

[19] J. L. Kim, U. N. Peled, I. Perepelitsa, V. Pless, and S. Friedland, "Explicit construction of families of LDPC codes with-cycles," *IEEE Trans. Inf. Theory*, vol. 50, no. 10, pp. 2378–2388, Oct. 2004.

[20] B. Vasic, K. Pedagani, and M. Ivkovic, "High-rate girth-eight low-density parity-check codes on rectangular integer lattices," *IEEE Trans. Inf. Theory*, vol. 52, no. 8, pp. 1248–1252, Aug. 2004.

[21] L. Ping and K. Y. Wu, "Concatenated tree codes: A low-complexity, high-performance approach," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 791–799, Feb. 2001.

[22] H. Behairy and S.-C. Chang, "Parallel concatenated Gallager codes for CDMA applications," in *Proc. IEEE Globecom 2001*, San Antonio, TX, Nov. 2006, vol. 2, pp. 1002–1006.

[23] J. Lu and J. M. F. Moura, "Partition and shift LDPC codes," *IEEE Trans. Magn.*, vol. 41, no. 10, pp. 2977–2979, Oct. 2005.

[24] ——, "Structured LDPC codes for high-density recording: Large girth and low error floor," *IEEE Trans. Magn.*, vol. 42, no. 2, pp. 208–213, Feb. 2006.

[25] J. Lu, J. M. F. Moura, and U. Niesen, Generalized Quasi-Cyclic LDPC Codes: Regular Structure and Arbitrary Girth, Oct. 2004, submitted for publication.

[26] J. Lu and J. M. F. Moura, "Turbo like decoding of LDPC codes," in *Proc. IEEE INTERMAG 2003*, Boston, MA, Mar./Apr. 2003, pp. D1–11.

[27] S. Dolinar and D. Divsalar, Weight Distribution for Turbo Codes Using Random and Nonrandom Permutations, JPL, Pasadena, CA, Progr. Rep. 42-122, Aug. 1995, pp. 56–65.

[28] H. R. Sadjadpour, M. Salehi, N. J. A. Sloane, and G. Nebe, "Interleaver design for short block length turbo codes," in *Proc. Int. Conf. Communications 2000*, New Orleans, LA, Jun. 2000, vol. 2, pp. 628–632.

[29] M. Ferrari, F. Scalise, and S. Bellini, "Prunable s-random interleavers," in *Int. Conf. Communications 2002*, New York, Apr. 2002, vol. 3, pp. 1711–1715.

[30] P. Popovski, L. Kocarev, and A. Risteski, "Design of flexible-length s-random interleaver for turbo codes," *IEEE Commun. Lett.*, vol. 8, no. 7, pp. 461–463, Jul. 2004.

[31] J. Lu and J. M. F. Moura, "Turbo design for LDPC codes with large girth," in *IEEE Int. Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, Rome, Italy, Jun. 2003, pp. 90–94.

[32] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf.. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

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