# Linear Time Encoding of LDPC Codes

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

***Abstract*—In this paper, we propose a linear complexity encoding method for arbitrary LDPC codes. We start from a simple graph-based encoding method "label-and-decide." We prove that the "label-and-decide" method is applicable to Tanner graphs with a hierarchical structure—pseudo-trees—and that the resulting encoding complexity is linear with the code block length. Next, we define a second type of Tanner graphs—the encoding stopping set. The encoding stopping set is encoded in linear complexity by a revised label-and-decide algorithm—the "label-decide-recompute." Finally, we prove that any Tanner graph can be partitioned into encoding stopping sets and pseudo-trees. By encoding each encoding stopping set or pseudo-tree sequentially, we develop a linear complexity encoding method for general low-density parity-check (LDPC) codes where the encoding complexity is proved to be less than $4 \cdot M \cdot (\bar{k} - 1)$, where $M$ is the number of independent rows in the parity-check matrix and $\bar{k}$ represents the mean row weight of the parity-check matrix.**

***Index Terms*—Encoding stopping set, low-density parity-check (LDPC) codes, linear complexity encoding, pseudo-tree, Tanner graphs.**

## I. Introduction

LOW-density parity check (LDPC) codes [1] are excellent error correcting codes with performance close to the Shannon capacity [2]. The key weakness of LDPC codes is their apparently high encoding complexity. The conventional way to encode LDPC codes is to multiply the data words $\overrightarrow{s}$ by the code generator matrix $\boldsymbol{G}$, i.e., the codewords are $\overrightarrow{x} = \boldsymbol{G} \cdot \overrightarrow{s}$. Though the parity-check matrix $\boldsymbol{H}$ for LDPC codes is sparse, the associated generator matrix $\boldsymbol{G}$ is not. The encoding complexity of LDPC codes is $\mathcal{O}(n^2)$ where $n$ is the block length of the LDPC code. For moderate-to-high code block length $n$, this quadratic behavior is very significant and it severely affects the application of LDPC codes. For example, LDPC codes have advantages over turbo codes [3] in almost every aspect except that LDPC codes have $\mathcal{O}(n^2)$ encoding complexity, while turbo codes have $\mathcal{O}(n)$ encoding complexity. It is highly desirable to reduce the $\mathcal{O}(n^2)$ encoding complexity of LDPC codes.

Several authors have addressed the issue of speeding encoding of LDPC codes and, generally speaking, they follow three different paths. The first path designs efficient encoding methods for particular types of LDPC codes. We list a few typical representations. Reference [4] proposes a linear complexity encoding method for cycle codes—LDPC codes with column weight 2. Reference [5] presents an efficient encoder for quasi-cyclic LDPC codes. In [6], an efficient encoding approach is proposed for Reed–Solomon-type array codes. Reference [7] shows that there exists a linear time encoder for turbo-structured LDPC codes. Reference [8] constructs LDPC codes based on finite geometries and proves that this type of structured LDPC codes can be encoded in linear time. In [9], [11], two families of irregular LDPC codes with cyclic structure and low encoding complexity are designed. In addition, an approximately lower triangular ensemble of LDPC codes [10] was proposed to facilitate almost linear complexity encoding. The above low-complexity encoders are only applicable to a small subset of LDPC codes, and some of the LDPC codes discussed above have performance loss when compared to randomly constructed LDPC codes. The second path borrows the decoder architecture and encodes LDPC codes iteratively on their Tanner graphs [12], [13]. The iterative LDPC encoding algorithm is easy to implement. However, there is no guarantee that iterative encoding will successfully get the codeword. In particular, the iterative encoding method will get trapped at the stopping set. The third path utilizes the sparseness of the parity-check matrix to design a low-complexity encoder. In [14], the authors present an algorithm named "greedy search" that reduces the coefficient of the quadratic term. This encoding method is relatively efficient. Its computation complexity and matrix storage need to be further reduced for most practical applications.

In this paper, we develop an *exact* linear complexity encoding method for arbitrary LDPC codes. We start from two particular Tanner graph structures—"pseudo-tree" and "encoding stopping set"—and prove that both the pseudo-tree and the encoding stopping set LDPC codes can be encoded in linear time. Next, we prove that any LDPC code with maximum column weight three can be decomposed into pseudo-trees and encoding stopping sets. Therefore, LDPC codes with maximum column weight three can be encoded in linear time and the encoding complexity is no more than $2 \cdot M \cdot (\bar{k} - 1)$ where $M$ denotes the number of independent rows of the parity-check matrix and $\bar{k}$ represents the average row weight. Finally, we extend the $\mathcal{O}(n)$ complexity encoder to LDPC codes with arbitrary row weight distributions and column weight distributions. For arbitrary LDPC codes, we achieve $\mathcal{O}(n)$ encoding complexity, not exceeding $4 \cdot M \cdot (\bar{k} - 1)$.

The remainder of the paper is organized as follows. In Section II, we introduce relevant definitions and notation. Section III proposes a simple encoding algorithm "label-and-decide" that directly encodes an LDPC code on its Tanner graph. Section IV presents a particular type of Tanner graph with multiple layers—"pseudo-tree" and proves that any pseudo-tree can be encoded successfully in linear time by the label-and-decide algorithm. Section V studies the complement of the

Manuscript received April 23, 2008; revised March 26, 2009. Current version published December 23, 2009.

J. Lu is with Sun Microsystems, Bloomfield, CO 80021 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 USA (e-mail: moura@ece.cmu.edu).

Communicated by I. Sason, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2009.2034823

pseudo-tree—"encoding stopping set." Section VI proves that the encoding stopping set can also be encoded in linear time by an encoding method named "label-decide-recompute." Section VII demonstrates that any LDPC code with column weight at most three can be decomposed into pseudo-trees and encoding stopping sets. By encoding each pseudo-tree or encoding stopping set sequentially using the label-and-decide or the label–decide–recompute algorithms, we achieve linear complexity encoding for LDPC codes with maximum column weight three. Finally, we extend in this section this linear time encoding method to LDPC codes with arbitrary column weight distributions and row weight distributions. Section VIII concludes the paper.

## II. Notation

**LDPC codes**. LDPC codes can be described by their parity-check matrix or their associated Tanner graph [15]. In the Tanner graph, each bit becomes a bit node and each parity-check constraint becomes a check node. If a bit is involved in a parity-check constraint, there is an edge connecting the bit node and the corresponding check node. The degree of a check node in a Tanner graph is equivalent to the number of 1's in the corresponding row of the parity-check matrix, or, in another words, the row weight of the corresponding row. We will use the term "degree of a check node" and "row weight" interchangeably in this paper. Similarly, the degree of a bit node in a Tanner graph is equivalent to the column weight of the corresponding column of the parity-check matrix, and we will interchangeably use the term "degree of a bit node" and "column weight" in this paper. The LDPC codes discussed in this paper may be irregular, i.e., different columns of the parity-check matrix have different column weights and different rows of the parity-check matrix have different row weights. The parity-check matrix of an LDPC code may not be of full rank. If a row in the parity-check matrix can be written as the binary sums of some other rows in the parity-check matrix, this row is said to be *dependent* on the other rows. Otherwise, it is an *independent* row.

**Arithmetic over the binary field**. We represent by "$\oplus$" the summation over the binary field, i.e., an XOR operation. For example, $0 \oplus 1 = \mathrm{mod}\,(0+1, 2) = 1$. Similarly, we have $0 \oplus 0 = 0$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. In addition, we have the following $-x = x$ in the binary field. Further, we use the symbol "$\sim x$" to represent the inverse of the value $x$ in the binary field. As an illustration, $\sim 0 = 1$ and $\sim 1 = 0$.

**Generalized parity-check equation**. A conventional parity-check equation is shown in (1). The right-hand side of the parity-check equation is always 0.

$$x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0. \tag{1}$$

In this paper, we define the *generalized parity-check equation*, as shown in (2)

$$x_1 \oplus x_2 \oplus \cdots \oplus x_k = b. \tag{2}$$

On the right-hand side of (2), $b$ is a constant that can be either 0 or 1.

Let $C$ be a standard parity-check equation. If the values of some of the bits in the left-hand side of $C$ are already known, then $C$ can be equivalently rewritten as a generalized parity-check equation. For example, if the values of the bits $x_{p+1}, x_{p+2}, \ldots, x_k$ are known, we move these bits from the left-hand side of (1) to its right-hand side and rewrite it as follows:

$$x_1 \oplus x_2 \oplus \cdots \oplus x_p = x_{p+1} \oplus x_{p+2} \oplus \cdots \oplus x_k = b. \tag{3}$$

Let $C_1, C_2, \ldots, C_p$ be $p$ generalized parity-check equations, as shown in (4)

$$\begin{array}{lcccccccc} C_1: & x_{1,1} & \oplus & x_{1,2} & \oplus & \ldots & x_{1,a_1} & = & b_1 \\ C_2: & x_{2,1} & \oplus & x_{2,2} & \oplus & \ldots & x_{2,a_2} & = & b_2 \\ \vdots & \vdots & & & & & \vdots & & \vdots \\ C_p: & x_{p,1} & \oplus & x_{p,2} & \oplus & \ldots & x_{p,a_p} & = & b_p. \end{array} \tag{4}$$

We say $C_1, C_2, \ldots, C_p$ are dependent on each other if the corresponding homogeneous equations in (5) are dependent on each other

$$\begin{array}{ccccccccc} x_{1,1} & \oplus & x_{1,2} & \oplus & \ldots & x_{1,a_1} & = & 0 \\ x_{2,1} & \oplus & x_{2,2} & \oplus & \ldots & x_{2,a_2} & = & 0 \\ \vdots & & \vdots & & & \vdots & & \vdots \\ x_{p,1} & \oplus & x_{p,2} & \oplus & \ldots & x_{p,a_p} & = & 0. \end{array} \tag{5}$$

From (4) and (5), we derive that

$$b_1 \oplus b_2 \oplus \cdots \oplus b_p = 0 \tag{6}$$

when the $p$ generalized parity-check equations $C_1, C_2, \ldots, C_p$ are dependent on each other.

**Connected graph**. A graph is *connected* if there exists a path from any vertex to any other vertices in the graph. If a graph is not connected, we call it a *disjoint* graph.

**Relative complement of a subgraph $\mathcal{S}$ in a Tanner graph $\mathcal{G}$**. Let $\mathcal{G}$ be a Tanner graph and $\mathcal{S}$ be a subgraph of $\mathcal{G}$, i.e., $\mathcal{S} \subset \mathcal{G}$. We use the symbol $\mathcal{G} \backslash \mathcal{S}$ to denote the subgraph that contains the nodes and edges in $\mathcal{G}$, but not in $\mathcal{S}$. For example, let $C_1, C_2, \ldots, C_k$ be $k$ check nodes in a Tanner graph $\mathcal{G}$. The subgraph $\mathcal{G} \backslash \{C_1, C_2, \ldots, C_k\}$ represents the remaining graph after deleting check nodes $C_1, C_2, \ldots, C_k$ from $\mathcal{G}$. Assume $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$ are $k$ subgraphs in a Tanner graph $\mathcal{G}$. The notation $\mathcal{G} \backslash \{\mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \cup \mathcal{G}_k\}$ represents the subgraph where nodes and edges are in $\mathcal{G}$, but not in $\mathcal{G}_i$, $1 \le i \le k$.

## III. "Label-and-Decide" Encoding Algorithm

Initially, Tanner graphs [15] were developed to explain the decoding process for LDPC codes; in fact, they can be used for the encoding of LDPC codes as well [12]. To encode an LDPC code using its Tanner graph, we identify information bits and parity bits through a labeling process on the graph. After determining the information bits and the parity bits, we start by assigning numerical values to the bit nodes labeled as information bits and then in a second step, calculate the missing values of the parity bits sequentially. This encoding approach is named *label-and-decide*. It is described in Algorithm 1.

**Algorithm 1** Label-and-decide algorithm

```
Preprocessing (carry out only once):
Label every bit node either as information bit or parity bit on the Tanner graph.
Encoding:
$Flag \leftarrow 0$;
Get the values of all the bits labeled as information bits;
while there are parity bits undetermined do
    if there exists one undetermined parity bit $x$ that can be uniquely computed from the values of the information bits and the already determined parity bits then
        Compute the value of $x$.
    else
        $Flag \leftarrow 1$, exit the while loop.
    end if
end while
if $Flag = 1$ then
    Encoding is unsuccessful.
else
    Output the encoded codeword.
end if
```

*Example:* Fig. 1 shows on the left an LDPC code whose Tanner graph is a tree. Initially, all its bit nodes are unlabeled. First, we randomly pick bit nodes $x_1$, $x_2$, and $x_3$ to be information bits. According to the parity-check equation $C_1$, the value of bit $x_4$ depends on the values of the bits $x_1$, $x_2$, and $x_3$ such that $x_4 = x_1 \oplus x_2 \oplus x_3$. Therefore, $x_4$ should be labeled as a parity bit. Similarly, we label bits $x_5$, $x_6$, $x_8$, $x_9$ as information bits and label bits $x_7$, $x_{10}$ as parity bits. We represent information bits by solid circles and parity bits by empty circles. The labeling result is shown on the right in Fig. 1.

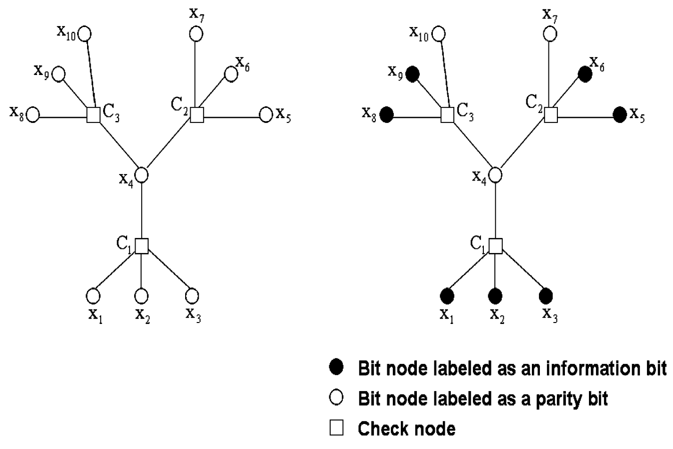
By the above labeling process, we decide the systematic component of the codeword $\overrightarrow{x} = (x_1\, x_2\, x_3\, x_4\, x_5\, x_6\, x_7\, x_8\, x_9\, x_{10})$ to be $\overrightarrow{s} = (x_1\, x_2\, x_3\, x_5\, x_6\, x_8\, x_9)$ and the parity component to be $\overrightarrow{p} = (x_4\, x_7\, x_{10})$. The label-and-decide encoding on the code in Fig. 1 then has the following steps:

Step 1. Get the values of the information bits $x_1$, $x_2$, $x_3$, $x_5$, $x_6$, $x_8$, and $x_9$ from the encoder input.

Step 2. Compute the parity bit $x_4$ from the parity-check equation $C_1 : x_4 = x_1 \oplus x_2 \oplus x_3$.

Step 3. Compute the parity bit $x_7$ from the parity-check equation $C_2$: $x_7 = x_4 \oplus x_5 \oplus x_6$; compute the parity bit $x_{10}$ from the parity-check equation $C_3$: $x_{10} = x_4 \oplus x_8 \oplus x_9$.

In fact, any tree code (whose Tanner graph is cycle-free) can be encoded in linear complexity by the label-and-decide algorithm. We will prove this fact in Corollary 2 in Section V. Further, the label-and-decide algorithm can be used to encode a particular type of Tanner graphs with cycles, i.e., the pseudo-tree we propose in the next section.



x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
C1
C2
C3
● Bit node labeled as an information bit
○ Bit node labeled as a parity bit
□ Check node

Fig. 1. Left: A Tanner graph. Right: Labeling bit nodes on the Tanner graph shown on the left.

## IV. Pseudo-Tree

A *pseudo-tree* is a connected Tanner graph that satisfies the following conditions (A1) through (A4).

(A1) It is composed of $2P+1$ tiers where $P$ is a positive integer. We number these tiers from 1 to $2P+1$, starting from the top. The $(2i-1)$th tier $(i = 1, 2, \ldots, P+1)$ contains only bit nodes, while the $(2i)$th tier $(i = 1, 2, \ldots, P)$ contains only check nodes.

(A2) Each bit node in the first tier has degree one and is connected to one and only one check node in the second tier.

(A3) For each check node $C_\alpha$ in the $(2i)$th tier, where $i$ can take any value from 1 to $P$, there is one and only one bit node $x_\alpha$ in the $(2i-1)$th tier (immediate upper tier) that connects to $C_\alpha$, and there are no other bit nodes in the upper tiers that connect to $C_\alpha$. We call $x_\alpha$ the *parent* of $C_\alpha$ and $C_\alpha$ the *child* of $x_\alpha$.

(A4) For each bit node $x_\beta$ in the $(2i-1)$th tier, where $i$ can take any value from 2 to $P$, there is at most one check node $C_\beta$ in the $(2i)$th tier (immediate lower tier) that connects to $x_\beta$, and there are no other check nodes in the lower tiers that connect to $x_\beta$.

For example, Fig. 2 shows a pseudo-tree with seven tiers. It contains many cycles. Each check node $C_i$ in the pseudo-tree is connected to a unique bit node in the immediate upper tier, while each bit node $x_i$ in the pseudo-tree may connect to multiple check nodes in the upper tiers.

An important characteristic of a pseudo-tree is that it can be encoded in linear complexity by the label-and-decide algorithm. This is proved in the following lemma.

*Lemma 1:* Any LDPC code whose Tanner graph is a pseudo-tree is linear time encodable.

*Proof:* Let a pseudo-tree contain $2P+1$ tiers, $N$ bit nodes, and $M$ check nodes. Condition (A3) guarantees that each check node is connected to one and only one parent bit node in the immediate upper tier. Condition (A4) guarantees that different check nodes are connected to different parent bit nodes. Therefore, there are $M$ parent bit nodes for the check nodes. We label

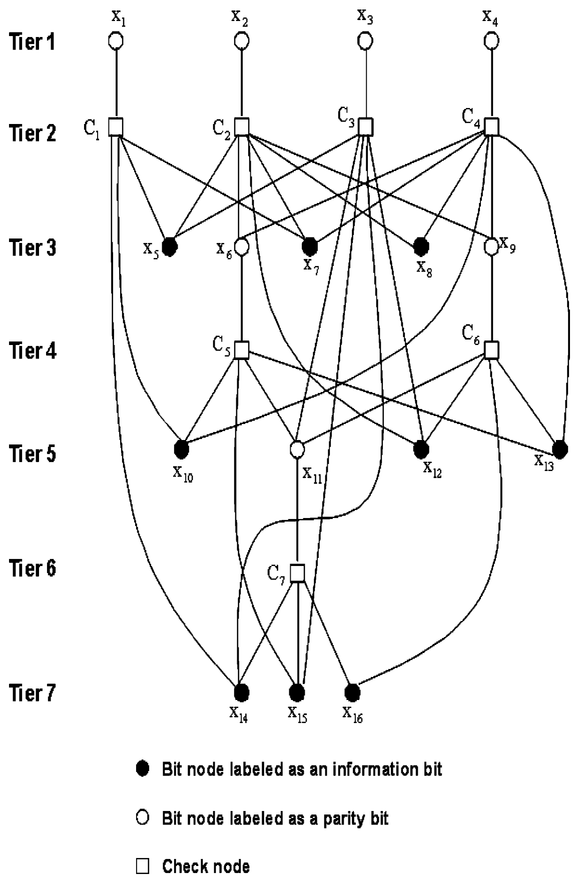


Fig. 2. A pseudo-tree.

these $M$ parent bit nodes as parity bits and the other $N-M$ bit nodes as information bits.

The inputs of the encoder provide the values for all the information bits. The task of the encoder is to compute the values for all the parity bits. Let $x_\alpha$ be an arbitrary parity bit in the $(2i-1)$th tier. By conditions (A3) and (A4), there is only one check node $C_\alpha$ in the lower tiers that connects to $x_\alpha$. The value of $x_\alpha$ is uniquely determined by the parity-check equation represented by $C_\alpha$. According to condition (A3), all the bit nodes involved in $C_\alpha$ except for $x_\alpha$ are in tiers below the $(2i-1)$th tier. Therefore, the value of $x_\alpha$ depends only on the values of the bit nodes below the $(2i-1)$th tier. For example, as shown in Fig. 2, parity bit $x_9$ is the parent bit node of the check node $C_6$. From the parity-check equation $C_6$, we see that the value of $x_9$ is computed from the values of $x_{11}$, $x_{12}$, $x_{16}$, and $x_{13}$, which are located below $x_9$. We compute the values of the parity bits tier by tier, starting from the $(2P-1)$th tier (bottom tier) and then progressing upwards. Each time we compute the value of a parity bit, we only need the values of those bits (both information bits and parity bits) in lower tiers, which are already known. Hence, this encoding process can proceed. The encoding process is repeated until the values of all the parity bits in the first tier are known.

We evaluate the computation complexity of the above encoding process. Let $k_i$, $i=1,2,\ldots,M$, denote the number of bits involved in the $i$th parity-check equation. The $i$th parity-check equation determines the value of a parity bit with $(k_i-2)$ XOR operations. So, $\sum_{i=1}^{M}(k_i-2)$ XOR operations are required to obtain all the $M$ parity bits. Let $\bar{k}=\frac{1}{M}\sum_{i=1}^{M}k_i$ denote the average number of bits in the $M$ parity-check equations, then the encoding complexity is $\mathcal{O}(M(\bar{k}-2))$. For LDPC codes with uniform row weight $k$, the encoding complexity is $\mathcal{O}(M(k-2))$. The above analysis shows that the encoding process is accomplished in linear time. This completes the proof. □

The linear-complexity encoding process described in the proof of Lemma 1 is summarized in Algorithm 2.

**Algorithm 2** Linear-complexity encoding algorithm for a pseudo-tree with $2P+1$ tiers.

```
Preprocessing:
Label the parent bit nodes in the pseudo-tree as parity bits
and the other bit nodes as information bits.
Encoding:
Get the values of all the information bits from the encoder
input.
for $i = P$ to 1 STEP-1 do
    Compute the values of all the parity bits in tier $2i-1$
    based on the values of the bits below the $(2i-1)$th tier.
end for
Output the encoded codeword.
```

We look at an example. We encode the pseudo-tree in Fig. 2 as follows:

Step 1. Determine the values of all the information bits $x_{14}$, $x_{15}$, $x_{16}$, $x_{10}$, $x_{12}$, $x_{13}$, $x_5$, $x_7$, and $x_8$.

Step 2. Compute the parity bit $x_{11}$ from the parity-check equation $C_7 : x_{11} = x_{14} \oplus x_{15} \oplus x_{16}$.

Step 3. Compute the parity bit $x_6$ from the parity-check equation $C_5 : x_6 = x_{10} \oplus x_{15} \oplus x_{11} \oplus x_{13}$; compute the parity bit $x_9$ from the parity-check equation $C_6 : x_9 = x_{11} \oplus x_{12} \oplus x_{16} \oplus x_{13}$.

Step 4. Compute the parity bits $x_1$, $x_2$, $x_3$, and $x_4$ in the first tier by the parity-check equations $C_1$, $C_2$, $C_3$, and $C_4$, respectively: $x_1 = x_{14} \oplus x_{10} \oplus x_5 \oplus x_7$, $x_2 = x_5 \oplus x_6 \oplus x_{12} \oplus x_7 \oplus x_8 \oplus x_9$, $x_3 = x_5 \oplus x_{11} \oplus x_{15} \oplus x_{14} \oplus x_{12}$, $x_4 = x_6 \oplus x_7 \oplus x_8 \oplus x_{10} \oplus x_9 \oplus x_{13}$.

The above encoding process requires only 25 XOR operations.

## V. Encoding Stopping Set

An *encoding stopping set* in a Tanner graph is a connected subgraph such that:

(B1) If a check node $C$ is in an encoding stopping set, then all the bit nodes involved in $C$ and the edges that are incident on $C$ are also in the encoding stopping set.

(B2) Any bit node in an encoding stopping set is connected to at least two check nodes in the encoding stopping set.

(B3) All the check nodes contained in an encoding stopping set are independent of each other, i.e., any parity-check equation cannot be represented as the binary sums of other parity-check equations.

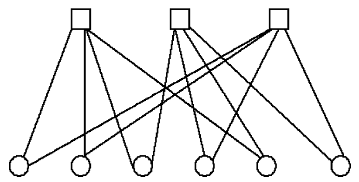


Fig. 3. A pseudo-encoding stopping set.

The number of check nodes in an encoding stopping set is called its size. If a connected Tanner graph satisfies conditions (B1) and (B2) but not condition (B3), we call this Tanner graph a *pseudo-encoding stopping set*. For example, the Tanner graph shown in Fig. 3 is not an encoding stopping set but a pseudo-encoding stopping set since it satisfies conditions (B1) and (B2) but not condition (B3). The Tanner graph shown in Fig. 4 is an encoding stopping set. Its size is 9. Every bit node in this encoding stopping set has degree greater than or equal to 2, and every check node is independent of each other. Please note that the "encoding stopping set" defined in this paper is different from the "stopping set" defined in [16]. Stopping sets are used for the finite-length analysis of LDPC codes on the binary erasure channel, while encoding stopping sets are used here to develop efficient encoding methods for LDPC codes. From the above definitions of pseudo-tree and encoding stopping set, we have the following lemma.

*Lemma 2:* Any pseudo-tree or union of pseudo-trees does not contain ecoding stopping sets.

*Proof:* See Appendix A.

We will show next that the label-and-decide algorithm cannot successfully encode encoding stopping sets.

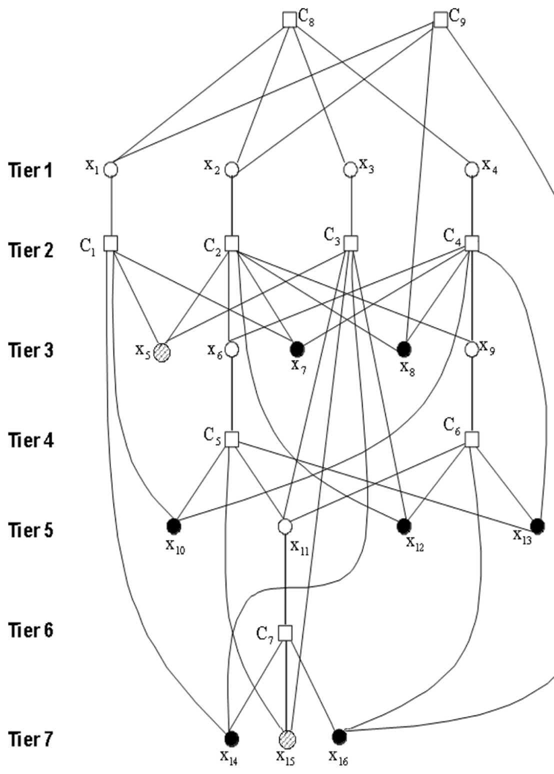
*Theorem 1:* An encoding stopping set cannot be encoded successfully by the label-and-decide algorithm.

*Proof:* Let $\mathcal{E}_f$ be an encoding stopping set and suppose $\mathcal{E}_f$ can be encoded successfully by the label-and-decide algorithm. Let $x_\alpha$ be the last parity bit being determined during the encoding process. Since $\mathcal{E}_f$ is an encoding stopping set, $x_\alpha$ is connected to at least two check nodes $C_\beta$ and $C_\gamma$ by condition (B2). Further, by condition (B3), all the check nodes in $\mathcal{E}_f$, including $C_\beta$ and $C_\gamma$, are independent of each other. Hence, for certain encoder inputs, $C_\beta$ and $C_\gamma$ provide different values for the parity bit $x_\alpha$. This contradicts the fact that every parity bit can be uniquely determined successfully by the label-and-decide algorithm. Hence, the label-and-decide algorithm cannot encode an encoding stopping set. This completes the proof. □

Conversely, if a Tanner graph does not contain any encoding stopping set, there must exist a linear complexity encoder for the corresponding code.

*Theorem 2:* If a Tanner graph $\mathcal{G}$ does not contain any encoding stopping set, then it can be encoded in linear time by the label-and-decide algorithm.

*Proof:* We first delete all redundant check nodes (i.e., dependent on other check nodes) from the Tanner graph $\mathcal{G}$. Next, we restrict our attention to the case that $\mathcal{G}$ is a connected graph. We will show that $\mathcal{G}$ can be equivalently transformed into a pseudo-tree if it is free of any encoding stopping set. Since $\mathcal{G}$ does not contain any encoding stopping set, $\mathcal{G}$ itself is not an encoding stopping set. Hence, there exist some degree-one bit nodes in $\mathcal{G}$. We generate a multilayer graph $\mathcal{T}$ and place those degree-one bit nodes in the first tier of $\mathcal{T}$. Next, the check nodes that connect to the degree-one bit nodes in the first tier of $\mathcal{T}$ are placed in the second tier of $\mathcal{T}$. Notice that there exist at least one bit node $x_\alpha$ in $\mathcal{G}\backslash\mathcal{T}$ such that $x_\alpha$ connects to at most one check node in $\mathcal{G}\backslash\mathcal{T}$. This statement is true. Otherwise, $\mathcal{G}\backslash\mathcal{T}$ becomes an encoding stopping set, which contradicts the fact that $\mathcal{G}$ does not contain any encoding stopping set. We pick all the bit nodes in $\mathcal{G}\backslash\mathcal{T}$ that connect to at most one check node in $\mathcal{G}\backslash\mathcal{T}$ and place them in the third tier of $\mathcal{T}$. Correspondingly, those check nodes in $\mathcal{G}\backslash\mathcal{T}$ that connect to the bit nodes in the third tier of $\mathcal{T}$ are placed in the fourth tier of $\mathcal{T}$. Each time we find bit nodes in $\mathcal{G}\backslash\mathcal{T}$ that connect to at most one check node in $\mathcal{G}\backslash\mathcal{T}$, we place those bit nodes in a new tier $2s+1$ of $\mathcal{T}$ and place the check nodes connecting to those bit nodes in the following new tier $2s+2$ of $\mathcal{T}$. We continue finding such bit nodes and increasing tiers till all the nodes in $\mathcal{G}$ are included in $\mathcal{T}$. Up to now, the multilayer structure constructed so far satisfies the conditions (A1), (A2), and (A4). Condition (A3) may fail to be satisfied. For example, as shown on the top in Fig. 5, the check node $C_4$ in tier 4 is connected to two bit nodes $x_6$ and $x_7$ in tier 3, which contradicts condition (A3). To satisfy condition (A3), we further adjust the positions of the bit nodes. If a check node in tier $2i$ is connected to $k$ bit nodes in the upper tiers of $\mathcal{T}$, we pick



● Bit node labeled as an information bit
▨ Bit node labeled as a reevaluated bit
○ Bit node labeled as a parity bit
□ Check node

Fig. 4. An encoding stopping set that contains a pseudo-tree shown in Fig. 2.

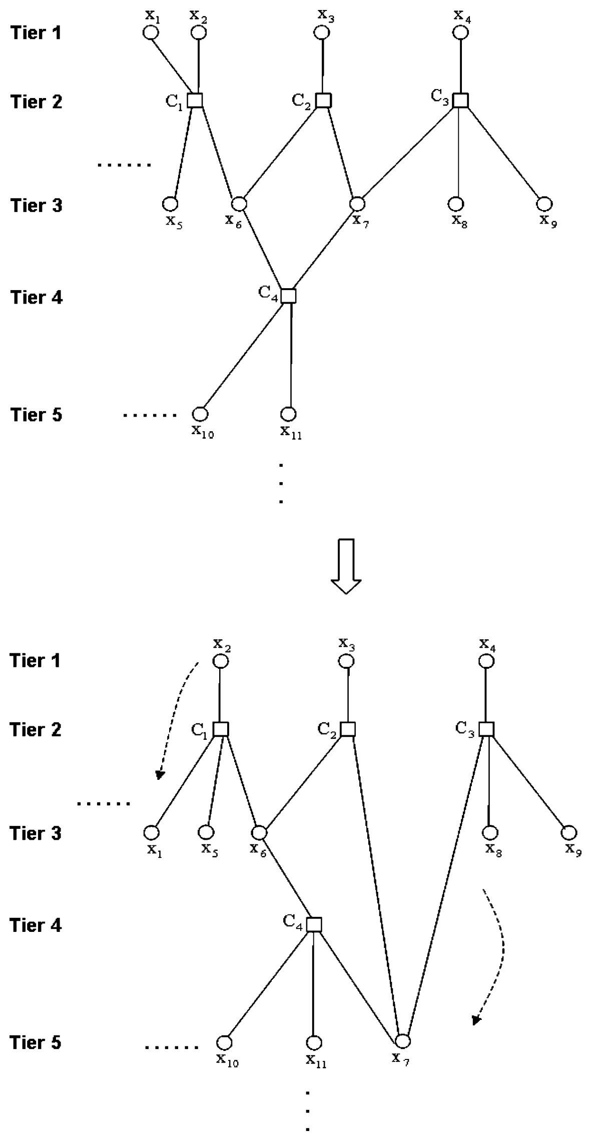


Fig. 5. Top: A multilayer structure but not a pseudo-tree. (Note that $C_4$ has two parents $x_6$ and $x_7$ and $C_1$ has two parents $x_1$ and $x_2$.) Bottom: The pseudo-tree that evolves from the multilayer structure shown on the top.

one bit node in tier $2i-1$ from these $k$ bit nodes and leave its position unchanged. Next, we drag the other $k-1$ bit nodes from their initial positions in tier $2i-1$ to the $(2i+1)$th tier. To illustrate, let us focus on Fig. 5 again. We drag the bit node $x_7$ from tier 3 to tier 5 and drag the bit node $x_1$ from tier 1 to tier 3. The newly formed graph is shown at the bottom in Fig. 5, which follows condition (A3). By tuning the positions of the bit nodes in this way, the resulting hierarchical graph satisfies conditions (A1) to (A4). In this way, we transform $\mathcal{G}$ into a pseudo-tree. By Lemma 1, a pseudo-tree is linear time encodable. Therefore, the encoding complexity of $\mathcal{G}$ is $\mathcal{O}(M)$ where $M$ denotes the number of independent check nodes contained in $\mathcal{G}$.

We now prove the case that $\mathcal{G}$ is a disjoint graph. Let $\mathcal{G}$ contain $p$ connected subgraphs: $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_p$. By the above analysis, the complexity of encoding $\mathcal{G}_i$ is $\mathcal{O}(M_i), i = 1, 2, \ldots, p$, where $M_i$ denotes the number of independent check nodes contained in $\mathcal{G}_i$. Since $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \cup \mathcal{G}_p$, then the encoding complexity of $\mathcal{G}$ is $\sum_{i=1}^{p} \mathcal{O}(M_i) = \mathcal{O}(\sum_{i=1}^{p} M_i) = \mathcal{O}(M)$ where $M$ is the number of independent check nodes in $\mathcal{G}$. This completes the proof. □

In the proof of Theorem 2, we detailed the process of transforming a connected Tanner graph that is free of any encoding stopping sets into a pseudo-tree. We further summarize the above transformation process in Algorithm 3.

**Algorithm 3** Transform a connected Tanner graph $\mathcal{G}$ into a pseudo-tree $\mathcal{T}$.

```
Remove all the redundant check nodes from the Tanner graph $\mathcal{G}$.
$\mathcal{T} = \phi$; $Flag \leftarrow 0$; $i = 1$.
**if** there exist degree-one bit nodes in $\mathcal{G}$ **then**
    Place all the degree-one bit nodes in the first tier of $\mathcal{T}$.
    Place all the check nodes that connect to the degree-one bit nodes in the second tier of $\mathcal{T}$.
**else**
    $Flag \leftarrow 1$.
**end if**
**while** $Flag = 0$ and $\mathcal{G} \neq \mathcal{T}$ **do**
    **if** there exist bit nodes in $\mathcal{G} \backslash \mathcal{T}$ that connect to at most one check node in $\mathcal{G} \backslash \mathcal{T}$ **then**
        Pick all the bit nodes in $\mathcal{G} \backslash \mathcal{T}$ that connect to at most one check node in $\mathcal{G} \backslash \mathcal{T}$ and place them in a new tier $2i+1$ of $\mathcal{T}$.
        Pick all those check nodes in $\mathcal{G} \backslash \mathcal{T}$ that connect to the bit nodes in the tier $2i+1$ of $\mathcal{T}$ and place them in a new tier $2i+2$ of $\mathcal{T}$.
    **else**
        $Flag \leftarrow 1$.
    **end if**
    $i = i + 1$.
**end while**
**if** $Flag = 0$ **then**
    **for** $s = 1$ to $i$ **do**
        **while** a check node in tier $2s$ is connected to $k$ $(k > 1)$ bit nodes in the upper tiers of $\mathcal{T}$ **do**
            Pick one bit node in tier $2s-1$ from these $k$ bit nodes and leave its position unchanged.
            Drag the other $k-1$ bit nodes from their initial positions in tier $2s-1$ to the $(2s+1)$th tier, as shown in Fig. 5.
        **end while**
    **end for**
    Output the generated pseudo-tree $\mathcal{T}$ derived from the original Tanner graph $\mathcal{G}$.
**else**
    The Tanner graph $\mathcal{G}$ can NOT be transformed into a pseudo-tree.
**end if**
```

From Theorem 2, we easily derive the following four corollaries.

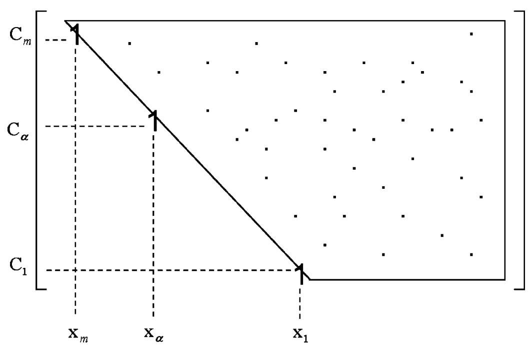


Fig. 6. A parity-check matrix in upper triangular form.

*Corollary 1:* If a Tanner graph does not contain any encoding stopping set, then it can be represented by a pseudo-tree or a union of pseudo-trees.

*Proof:* The proof of Corollary 1 can be found in the proof of Theorem 2. □

*Corollary 2:* The label-and-decide algorithm can encode any tree LDPC codes (whose Tanner graphs are cycle-free) with linear complexity.

*Proof:* Let $\mathcal{T}$ be the Tanner graph of a tree LDPC code and $\mathcal{S}$ be an arbitrary subgraph of $\mathcal{T}$. Since the Tanner graph $\mathcal{T}$ is a tree, its subgraph $\mathcal{S}$ is either a tree or a union of trees. Therefore, the graph $\mathcal{S}$ contains at least one bit leaf node with degree one. Since the graph $\mathcal{S}$ contains a degree-one bit node, $\mathcal{S}$ cannot be an encoding stopping set. Since no subgraph of $\mathcal{T}$ is an encoding stopping set, by Theorem 2, the tree code $\mathcal{T}$ can be encoded in linear complexity by the label-and-decide algorithm. This completes the proof. □

*Corollary 3:* A regular LDPC code with column weight 2 (cycle code) can be encoded in linear complexity by the label-and-decide algorithm.

*Proof:* We prove Corollary 3 by showing that a cycle code does not contain any encoding stopping set. Assume the cycle code contains an encoding stopping set $\mathcal{E}_f$. By the definition of cycle code and condition (B2), all the bit nodes in $\mathcal{E}_f$ have uniform degree two. It follows that the binary sum of all the parity-check equations in $\mathcal{E}_f$ is a vector of 0's. Then, at least one check node in $\mathcal{E}_f$ is dependent on the other check nodes. This contradicts condition (B3) that all the check nodes in an encoding stopping set are independent of each other. Hence, a cycle code does not contain any encoding stopping set. By Theorem 2, a cycle code is linear time encodable by the label-and-decide algorithm. This completes the proof. □

An alternative proof can be found in [4].

*Corollary 4:* Let $\boldsymbol{H}$ be the parity-check matrix of an LDPC code. If $\boldsymbol{H}$ can be transformed into an upper triangular matrix $\boldsymbol{U}$ by row and column permutations, then the LDPC code can be encoded in linear time by the label-and-decide algorithm.

*Proof:* We label the rows of the upper triangular matrix $\boldsymbol{U}$ one by one as $C_1, C_2, \ldots, C_m$, from the bottom to the top, as shown in Fig. 6. We notice that if $i > j$, then there exists at least one bit that is involved in $C_i$ but not in $C_j$. Assume the Tanner graph of the code contains an encoding stopping set $\mathcal{E}_f$ that contains check nodes $C_{i_1}, C_{i_2}, \ldots, C_{i_p}$. Let $q = \max(i_1, i_2, \ldots, i_p)$. There exists at least one bit node $x_q$ in $\mathcal{E}_f$ that only connects to $C_q$. This contradicts the fact that every bit node in an encoding stopping set is connected to at least two check nodes in the encoding stopping set. Hence, $\mathcal{E}_f$ is not an encoding stopping set. Since the Tanner graph of the LDPC code does not contain any encoding stopping set, by Theorem 2 it is linear time encodable. This completes the proof. □

Theorems 1 and 2 show that encoding stopping sets prevent the application of the label-and-decide algorithm. However, we will show in the next section that encoding stopping sets can also be encoded in linear complexity.

## VI. Linear Complexity Encoding Approach For Encoding Stopping Sets

Let $\mathcal{E}_f$ be an encoding stopping set. We say $\mathcal{E}_f$ is a *$k$-fold-constraint encoding stopping set* if the following two conditions hold.

(C1) There exist $k$ check nodes $C_1, C_2, \ldots, C_k$ in $\mathcal{E}_f$ such that $\mathcal{E}_f \backslash \{C_1, C_2, \ldots, C_k\}$ does not contain any encoding stopping set. We call the $k$ check nodes $C_1$, $C_2, \ldots, C_k$ *key check nodes*. We always specify the $k$ key check nodes $C_1, C_2, \ldots, C_k$ when defining a $k$-fold-constraint encoding stopping set.

(C2) For any $k-1$ check nodes $C_1, C_2, \ldots, C_{k-1}$ in $\mathcal{E}_f$, $\mathcal{E}_f \backslash \{C_1, C_2, \ldots, C_{k-1}\}$ contains an encoding stopping set.

The notation $\mathcal{E}_f \backslash \{C_1, C_2, \ldots, C_k\}$ denotes the remaining graph after deleting check nodes $C_1, C_2, \ldots, C_k$ from $\mathcal{E}_f$. Fig. 4 shows a twofold-constraint encoding stopping set with key check nodes $C_8$ and $C_9$. After deleting the two key check nodes $C_8$ and $C_9$ from this encoding stopping set, the Tanner graph turns into a pseudo-tree, see Fig. 2. We will focus on onefold-constraint and twofold-constraint encoding stopping sets in this paper, since we will show later that all types of LDPC codes can be decomposed into onefold or twofold constraint encoding stopping sets and pseudo-trees.

Let us first look at a twofold-constraint encoding stopping set $\mathcal{E}_f$ with two key check nodes $C_\alpha$ and $C_\beta$. The twofold-constraint encoding stopping set $\mathcal{E}_f$ has size $M$. By definition, the subgraph $\mathcal{E}_f \backslash \{C_\alpha, C_\beta\}$ does not contain any encoding stopping set. Therefore, we can transform the subgraph $\mathcal{E}_f \backslash \{C_\alpha, C_\beta\}$ into a pseudo-tree $\mathcal{T}$ by Algorithm 3 and then encode $\mathcal{T}$ in linear complexity. Based on the above analysis, we encode the twofold-constraint encoding stopping set $\mathcal{E}_f$ in three steps.

In the first step, we encode $\mathcal{E}_f \backslash \{C_\alpha, C_\beta\}$ using the label-and-decide algorithm according to Theorem 2. During encoding, $M-2$ bit nodes are labeled as parity bits and the remaining bit nodes are labeled as information bits. In the above encoding process, if we change the value of an information bit $x^*$ and keep the values of all the other information bits unchanged, some parity bits $x_1, x_2, \ldots, x_q$ will also change their values. We say that the parity bits $x_1, x_2, \ldots, x_q$ are *affected* by the bit $x^*$.

In the second step, we verify the two key check nodes $C_\alpha$ and $C_\beta$ based on the bit values acquired in Step 1. For convenience, we first define the value of a key check node as follows. Let a key check node $C_\alpha$ connect to $k$ bit nodes $x_1, x_2, \ldots, x_k$. The binary sum $x_1 \oplus x_2 \oplus \cdots \oplus x_k$ is defined to be the *value* of the check node $C_\alpha$. If the parity-check equation associated with a key check node $C_\alpha$ is satisfied, then the value of the key check node $C_\alpha$ is 0. Otherwise, the value of the key check node $C_\alpha$ is 1. If the values of the two key check nodes $C_\alpha$ and $C_\beta$ are both 0, the encoding results acquired in Step 1 are reliable. If at least one of the two key check nodes $C_\alpha$ and $C_\beta$ has value 1, then the encoding results acquired in Step 1 need correction. To correct the previously obtained encoding results, we notice that the existence of the key check nodes $C_\alpha$ and $C_\beta$ indicates that two bits $x_\gamma$ and $x_\delta$ that were previously labeled as information bits are actually parity bits, and their values are determined by $C_\alpha$ and $C_\beta$. We call the two bits $x_\gamma$ and $x_\delta$ *reevaluated bits.*

The reevaluated bits $x_\gamma$ and $x_\delta$ satisfy the following three conditions.

(D1) If $x_\gamma$ changes its value while $x_\delta$ and other information bits keep their values unchanged, then the value of the key check node $C_\alpha$ must also be changed. We say that $C_\alpha$ is affected by $x_\gamma$. Although $C_\alpha$ may not directly connect to $x_\gamma$, $C_\alpha$ is affected by $x_\gamma$ through other check nodes and parity bits.

(D2) If $x_\delta$ changes its value while $x_\gamma$ and other information bits keep their values unchanged, then the value of the key check node $C_\beta$ must also be changed. We say that $C_\beta$ is affected by $x_\delta$.

(D3) If $x_\gamma$ and $x_\delta$ both change their values while the other information bits keep their values unchanged, then the value of at least one of the two key check nodes $C_\alpha$ and $C_\beta$ is changed.

Since $C_\alpha$, $C_\beta$, and the other check nodes in $\mathcal{E}_f$ are independent of each other, there must exist bit nodes $x_\gamma$ and $x_\delta$ that satisfy conditions (D1) to (D3). An algorithm for finding reevaluated bits $x_\gamma$ and $x_\delta$ from a twofold-constraint encoding stopping set is presented in Appendix B. We also prove the validness of the algorithm in Appendix B. Notice that the workload to find the two reevaluated bits is a preprocessing step that is carried out only once. Assume in Step 1 that $x_\gamma$ and $x_\delta$ are randomly assigned initial values $x_\gamma^0$ and $x_\delta^0$, respectively. If the parity-check equations $C_\alpha$ and $C_\beta$ are both satisfied, the initial values $x_\gamma^0$ and $x_\delta^0$ are the correct values for $x_\gamma$ and $x_\delta$. If $C_\alpha$, or $C_\beta$, or both, are not satisfied, we need to recompute the values of $x_\gamma$ and $x_\delta$ from the values of the key check nodes $C_\alpha$ and $C_\beta$. Let $\triangle x_\gamma = \widetilde{x_\gamma} - x_\gamma^0$ and $\triangle x_\delta = \widetilde{x_\delta} - x_\delta^0$ where $\widetilde{x_\gamma}$ and $\widetilde{x_\delta}$ are the correct values of $x_\gamma$ and $x_\delta$, respectively, and let $\widetilde{C_\alpha}$ and $\widetilde{C_\beta}$ be the values of the key check nodes $C_\alpha$ and $C_\beta$, respectively. If $C_\alpha$ is affected by $x_\gamma$ alone, $C_\beta$ is affected by both $x_\gamma$ and $x_\delta$, we derive the following equations:

$$\begin{aligned} \triangle x_\gamma &= -\widetilde{C_\alpha} = \widetilde{C_\alpha} \\ \triangle x_\gamma \oplus \triangle x_\delta &= -\widetilde{C_\beta} = \widetilde{C_\beta}. \end{aligned} \tag{7}$$

If $C_\alpha$ is affected by both $x_\gamma$ and $x_\delta$ and $C_\beta$ is affected by $x_\delta$ alone, we derive the following equations:

$$\begin{aligned} \triangle x_\gamma \oplus \triangle x_\delta &= -\widetilde{C_\alpha} = \widetilde{C_\alpha} \\ \triangle x_\delta &= -\widetilde{C_\beta} = \widetilde{C_\beta}. \end{aligned} \tag{8}$$

If $C_\alpha$ is affected by $x_\gamma$ alone and $C_\beta$ is affected by $x_\delta$ alone, we have the following equations:

$$\begin{aligned} \triangle x_\gamma &= -\widetilde{C_\alpha} = \widetilde{C_\alpha} \\ \triangle x_\delta &= -\widetilde{C_\beta} = \widetilde{C_\beta}. \end{aligned} \tag{9}$$

From (7)–(9), we can get the correct values of $x_\gamma$ and $x_\delta$. Notice that, condition (D3) prevents the case that $C_\alpha$ is affected by both $x_\gamma$ and $x_\delta$ and $C_\beta$ is also affected by both $x_\gamma$ and $x_\delta$.

**Algorithm 4** Determine the parity bits in a twofold-constraint encoding stopping set $\mathcal{E}_f$ that are affected by the reevaluated bits $x_\gamma$ and $x_\delta$. The two key check nodes of $\mathcal{E}_f$ are $C_\alpha$ and $C_\beta$.

```
Use algorithm 3 to transform $\mathcal{E}_f \backslash \{C_\alpha, C_\beta\}$ into a
pseudo-tree $\mathcal{T}$. Assume the reevaluated bit $x_\gamma$ is in the
$(2i-1)$th tier of $\mathcal{T}$ and the reevaluated bit $x_\delta$ is in the
$(2k-1)$th tier of $\mathcal{T}$.
for $d = i-1$ to 1 STEP-1 do
  for Each parity bit node in tier $2d-1$ of $\mathcal{T}$ do
    if the child (a check node) of a parity bit node $x$ is
    connected to $x_\gamma$ and is also connected to an even
    number (including 0) of parity bits that are affected
    by $x_\gamma$ and are located below the $(2d-1)$th tier then
      Label $x$ as a parity bit that is affected by $x_\gamma$.
    else if the child of the parity bit node $x$ is not
    connected to $x_\gamma$ but is connected to an odd number
    of parity bits that are affected by $x_\gamma$ and are located
    below the $(2d-1)$th tier then
      Label $x$ as a parity bit that is affected by $x_\gamma$.
    else
      The parity bit $x$ is not affected by $x_\gamma$.
    end if
  end for
end for
for $d = k-1$ to 1 STEP-1 do
  for Each parity bit node in tier $2d-1$ of $\mathcal{T}$ do
    if the child (a check node) of a parity bit node $x$ is
    connected to $x_\delta$ and is also connected to an even
    number (including 0) of parity bits that are affected
    by $x_\delta$ and are located below the $(2d-1)$th tier then
      Label $x$ as a parity bit that is affected by $x_\delta$.
    els if the child of the parity bit node $x$ is not connected
    to $x_\delta$ but is connected to an odd number of parity
    bits that are affected by $x_\delta$ and are located below the
    $(2d-1)$th tier then
      Label $x$ as a parity bit that is affected by $x_\delta$.
    else
      The parity bit $x$ is not affected by $x_\delta$.
    end if
  end for
end for
Output the labels (whether a parity bit is affected by $x_\gamma$, by
$x_\delta$, or by both, or by neither) of all the parity bits.
```

In the third step, we determine the parity bits $x_1, x_2, \ldots, x_q$ that are affected by the reevaluated bits $x_\gamma$ and $x_\delta$. We provide Algorithm 4 to find those parity bits $x_1, x_2, \ldots, x_q$ that are affected by the reevaluated bits $x_\gamma$ and $x_\delta$. The output of Algorithm 4 may label a parity bit as being affected by $x_\gamma$ alone, by $x_\delta$ alone, by both $x_\gamma$ and $x_\delta$, or by neither $x_\gamma$ nor $x_\delta$. Next, we recompute those parity bits $x_1, x_2, \dots, x_q$ that are affected by $x_\gamma$ and/or $x_\delta$ based on the correct values of $x_\gamma$ and $x_\delta$. This encoding method is named *label–decide–recompute* and is described in Algorithm 5.

**Algorithm 5** Label-decide-recompute algorithm for a twofold-constraint encoding stopping set $\mathcal{E}_f$ with two key check nodes $C_\alpha$ and $C_\beta$. $\mathcal{E}_f$ contains $M$ check nodes.

```
Preprocessing (carry out only once):
Use Algorithm 11 to pick two information bits $x_\gamma$ and $x_\delta$ that satisfy conditions (D1) to (D3) as reevaluated bits.
Use Algorithm 4 to determine the parity bits $x_{p_1}, x_{p_2}, \dots, x_{p_s}$ that are affected by $x_\gamma$ and/or $x_\delta$.
Encoding:
Fill the values of the information bits except for $x_\gamma$ and $x_\delta$;
Assign $x_\gamma = 0$ and $x_\delta = 0$;
Encode $\mathcal{E}_f \backslash \{C_\alpha, C_\beta\}$ using Algorithm 2. Compute the values of the $M-2$ parity bits;
Compute the values $\widetilde{C_\alpha}$ and $\widetilde{C_\beta}$ of the key check nodes $C_\alpha$ and $C_\beta$, respectively;
if $\widetilde{C_\alpha} \neq 0$ AND/OR $\widetilde{C_\beta} \neq 0$ then
    Recompute the values of $x_\gamma$ and $x_\delta$ from $\widetilde{C_\alpha}$ and $\widetilde{C_\beta}$ by (7)–(9);
    for $i = 1$ to $s$ do
        Recompute the value of the parity bit $x_{p_i}$ based on the new values of $x_\gamma$ and $x_\delta$;
    end for
end if
Output the encoding result.
```

Next, we analyze the computation complexity of the label–decide–recompute algorithm when encoding a twofold-constraint encoding stopping set with size $M$. Every check node except for the two key check nodes $C_\alpha$ and $C_\beta$ are computed at most twice in the label–decide–recompute encoding (label-and-decide step and recompute step) while the values of the two key check nodes $C_\alpha$ and $C_\beta$ need to be computed only once. In addition, we may need one extra XOR operation to compute the two reevaluated bits $x_\gamma$ and $x_\delta$ by (7)–(9). Hence, the encoding complexity of the label–decide–recompute algorithm is less than or equal to $2 \cdot \sum_{i=1}^{M-2}(k_i - 2) + (k_\alpha - 1) + (k_\beta - 1) + 1$, where $k_i$, $1 \le i \le M-2$, are the degrees of the check nodes other than $C_\alpha$, $C_\beta$ and $k_\alpha$, $k_\beta$ are the degrees of the check nodes $C_\alpha$ and $C_\beta$, respectively. The encoding complexity of the label–decide–recompute algorithm can be further simplified to be less than $2 \cdot M \cdot (\bar{k} - 1)$ where $M$ is the number of check nodes in the encoding stopping set and $\bar{k}$ is the average number of bit nodes involved in each check node in the encoding stopping set. This shows that the label–decide–recompute algorithm encodes any twofold-constraint encoding stopping set in linear time. The preprocessing (determining reevaluated bits, and parity bits affected by the reevaluated bits) is done offline and does not count towards encoder complexity.

We look at an example. Fig. 4 shows a twofold-constraint encoding stopping set $\mathcal{E}_f$ with key check nodes $C_8$ and $C_9$. After deleting the two key check nodes $C_8$ and $C_9$, $\mathcal{E}_f$ becomes the pseudo-tree shown in Fig. 2. Next, we determine the two reevaluated bits following Algorithm 11. We represent the two key check nodes $C_8$ and $C_9$ as functions of the information bits as follows:

$$C_8 = x_5 \oplus x_7 \oplus x_{13} \oplus x_{14} \oplus x_{16} \tag{10}$$

$$C_9 = x_{14} \oplus x_{15} \tag{11}$$

From (10) and (11), we see that the value of the bit node $x_5$ only affects key check node $C_8$ and the value of the bit node $x_{15}$ only affects key check node $C_9$. The two bit nodes $x_5$ and $x_{15}$ satisfy conditions (D1) to (D3). Hence, we can choose the two bits $x_5$ and $x_{15}$ as reevaluated bits (there exist other options of the reevaluated bits).

After determining the reevaluated bits, we use Algorithm 4 to find the parity bits that are affected by the reevaluated bits $x_5$ and $x_{15}$. We start from tier 5, the parity bit $x_{11}$ is affected by the reevaluated bit $x_{15}$. In tier 3, the parity bit $x_9$ is also affected by $x_{15}$. However, since the parity bit $x_6$ has child $C_5$ and $C_5$ is connected to both $x_{15}$ and the parity bit $x_{11}$ that is affected by $x_{15}$, the parity bit $x_6$ is not affected by $x_{15}$. Similarly, in tier 1, we determine that the parity bits $x_2$ and $x_4$ are affected by $x_{15}$ while the parity bits $x_1$ and $x_3$ are not affected by $x_{15}$. In the same way, we label the parity bits $x_1$, $x_2$, and $x_3$ as being affected by the reevaluated bit $x_5$ while the parity bit $x_4$ is not affected by $x_5$. We notice that the parity bit $x_2$ is affected by both $x_{15}$ and $x_5$ while the parity bit $x_6$ is affected by neither $x_{15}$ nor $x_5$.

After finishing the above preprocessing, we use the label–decide–recompute algorithm to encode $\mathcal{E}_f$ as follows.

Step 1. Assign $x_5 = 0$ and $x_{15} = 0$. Encode the pseudo-tree part following the procedures in Section IV.

Step 2. Compute the values of the key check nodes $C_8$ and $C_9$, e.g., $\widetilde{C_8} = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ and $\widetilde{C_9} = x_1 \oplus x_2 \oplus x_8 \oplus x_{16}$.

Step 3a. If $\widetilde{C_8} = 0$ and $\widetilde{C_9} = 0$, stop encoding and output the codeword $[x_1\ x_2\ \dots\ x_{16}]$.

Step 3b. If $\widetilde{C_8} = 1$ and/or $\widetilde{C_9} = 1$, recompute the values of $x_5$ and $x_{15}$ as follows: $x_5 = \widetilde{C_8}$ and $x_{15} = \widetilde{C_9}$, where $\widetilde{C_8}$ and $\widetilde{C_9}$ are the values of the parity-check equations $C_8$ and $C_9$, respectively. Recompute the parity bits $x_{11}, x_9, x_1, x_2, x_3$, and $x_4$ based on the new values of $x_5$ and $x_{15}$. Output the codeword $[x_1\ x_2\ \dots\ x_{16}]$.

The label–decide–recompute algorithm can be further simplified. We restudy the third step of the label–decide–recompute method. Assume $x_1, x_2, \dots, x_q$ are the parity bits whose values need to be updated. In order to get the new values of the parity bits $x_1, x_2, \dots, x_q$, we need to recompute those parity-check equations that involve $x_1, x_2, \dots, x_q$. In fact, instead of recomputing the parity-check equations involving parity bits $x_1, x_2, \dots, x_q$, we can directly flip the values of the parity bits $x_1, x_2, \dots, x_q$ since in the binary field the value of a bit is either 0 or 1.

For example, if the correct value of the reevaluated bit $x_\gamma$ is different from its originally assigned value $x_\gamma^0$, we simply flip the values of those parity bits that are affected by $x_\gamma$. We name the above encoding method *label–decide–flip* and describe it in Algorithm 6. The encoding complexity of Algorithm 6 is at most $M \cdot (\bar{k}-1)$ XOR operations plus two vector flipping operations. We, again, look at an example. The twofold-constraint encoding stopping set shown in Fig. 4 can be encoded by Algorithm 6 as follows.

**Algorithm 6** Label–decide–flip algorithm for a twofold-constraint encoding stopping set $\mathcal{E}_f$ with two key check nodes $C_\alpha$ and $C_\beta$. $\mathcal{E}_f$ contains $M$ check nodes.

```
Preprocessing (carry out only once):
Use Algorithm 11 to pick two information bits $x_\gamma$ and $x_\delta$ that satisfy conditions (D1) to (D3) as reevaluated bits.
Use Algorithm 4 to determine the parity bits $x_{u_1}, x_{u_2}, \ldots, x_{u_r}$ that are affected by $x_\gamma$ alone and group $x_{u_1}, x_{u_2}, \ldots, x_{u_r}$ in a vector $\overrightarrow{X_\gamma} = [x_{u_1}\ x_{u_2}\ \ldots\ x_{u_r}]$.
Use Algorithm 4 to determine the parity bits $x_{p_1}, x_{p_2}, \ldots, x_{p_s}$ that are affected by $x_\delta$ alone and group $x_{p_1}, x_{p_2}, \ldots, x_{p_s}$ in a vector $\overrightarrow{X_\delta} = [x_{p_1}\ x_{p_2}\ \ldots\ x_{p_s}]$.
Use Algorithm 4 to determine the parity bits $x_{q_1}, x_{q_2}, \ldots, x_{q_t}$ that are affected by both $x_\gamma$ and $x_\delta$ and group $x_{q_1}, x_{q_2}, \ldots, x_{q_t}$ in a vector $\overrightarrow{X_\epsilon} = [x_{q_1}\ x_{q_2}\ \ldots\ x_{q_t}]$.
Encoding:
Fill the values of the information bits except for $x_\gamma$ and $x_\delta$;
Assign $x_\gamma = 0$ and $x_\delta = 0$;
Encode $\mathcal{E}_f \backslash \{C_\alpha, C_\beta\}$ using Algorithm 2. Compute the values of the $M-2$ parity bits;
Compute the values $\widetilde{C_\alpha}$ and $\widetilde{C_\beta}$ of the parity-check equations $C_\alpha$ and $C_\beta$, respectively;
if $\widetilde{C_\alpha} \neq 0$ AND/OR $\widetilde{C_\beta} \neq 0$ then
    Recompute the values of $x_\gamma$ and $x_\delta$ from $\widetilde{C_\alpha}$ and $\widetilde{C_\beta}$ by (7)–(9);
    if $x_\gamma = 1$ AND $x_\delta = 0$ then
        Flip the vectors $\overrightarrow{X_\gamma}$ and $\overrightarrow{X_\epsilon}$.
    else if $x_\gamma = 0$ AND $x_\delta = 1$ then
        Flip the vectors $\overrightarrow{X_\delta}$ and $\overrightarrow{X_\epsilon}$.
    else if $x_\gamma = 1$ AND $x_\delta = 1$ then
        Flip the vectors $\overrightarrow{X_\gamma}$ and $\overrightarrow{X_\delta}$.
    end if
end if
Output the encoding result.
```

**Preprocessing:** We choose the reevaluated bits to be $x_5$ and $x_{15}$. We also determine that the parity bits $x_1$ and $x_3$ are affected by $x_5$ alone and the parity bits $x_{11}$, $x_9$, $x_4$ are affected by $x_{15}$ alone. Further, we determine that the parity bit $x_2$ is affected by both $x_5$ and $x_{15}$.

**Encoding:**

Step 1. Assign $x_5 = 0$ and $x_{15} = 0$. Encode the pseudo-tree part following the procedures in Section IV.

Step 2. Compute the values of the parity-check equations $C_8$ and $C_9$, e.g., $\widetilde{C_8} = x_1 \oplus x_2 \oplus x_3 \oplus x_4$ and $\widetilde{C_9} = x_1 \oplus x_2 \oplus x_8 \oplus x_{16}$.

Step 3a. If $\widetilde{C_8} = 0$ and $\widetilde{C_9} = 0$, stop encoding and output the codeword $[x_1\ x_2\ \ldots\ x_{16}]$.

Step 3b. If $\widetilde{C_8} = 1$ and/or $\widetilde{C_9} = 1$, recompute the values of $x_5$ and $x_{15}$ as the following: $x_5 = \widetilde{C_8}$ and $x_{15} = \widetilde{C_9}$ where $\widetilde{C_8}$ and $\widetilde{C_9}$ are the values of the parity-check equations $C_8$ and $C_9$, respectively. If $x_5 = 1$ and $x_{15} = 0$, flip the values of the vectors $[x_1\ x_3]$ and $[x_2]$ to be $[\sim x_1\ \sim x_3]$ and $[\sim x_2]$, respectively. If $x_5 = 0$ and $x_{15} = 1$, flip the values of the vectors $[x_{11}\ x_9\ x_4]$ and $[x_2]$ to be $[\sim x_{11}\ \sim x_9\ \sim x_4]$ and $[\sim x_2]$, respectively. If $x_5 = 1$ and $x_{15} = 1$, flip the values of the vectors $[x_1\ x_3]$ and $[x_{11}\ x_9\ x_4]$ to be $[\sim x_1\ \sim x_3]$ and $[\sim x_{11}\ \sim x_9\ \sim x_4]$, respectively. Output the codeword $[x_1\ x_2\ \ldots\ x_{16}]$.

It is easy to revise Algorithm 5 and Algorithm 6 to encode a onefold-constraint encoding stopping set. For example, Algorithm 7 shows the label–decide–recompute algorithm for a onefold-constraint encoding stopping set. The encoding complexity of Algorithm 7 is less than $2 \cdot M \cdot (\bar{k}-1)$ where $M$ is the number of check nodes in the encoding stopping set and $\bar{k}$ is the average number of bit nodes involved in each check node in the encoding stopping set.

**Algorithm 7** Label-decide-recompute algorithm for a onefold-constraint encoding stopping set $\mathcal{E}_f$ with key check node $C^*$. $\mathcal{E}_f$ contains $M$ check nodes.

```
Preprocessing (carry out only once):
Use Algorithm 11 to pick an information bit $x^*$ that affects the key parity-check equation $C^*$.
Use Algorithm 4 to determine the parity bits $x_{p_1}, x_{p_2}, \ldots, x_{p_s}$ that are affected by $x^*$.
Encoding:
Fill the values of the information bits except for $x^*$.
Assign $x^* = 0$.
Encode $\mathcal{E}_f \backslash C^*$ using Algorithm 2, compute the values of the $M-1$ parity bits.
Verify the parity-check equation $C^*$.
if the parity-check equation $C^*$ is not satisfied then
    $x^* \leftarrow 1$.
    for $i = 1$ to $s$ do
        Recompute the value of the parity bit $x_{p_i}$ based on the new value of $x^*$;
    end for
end if
Output the encoding result.
```

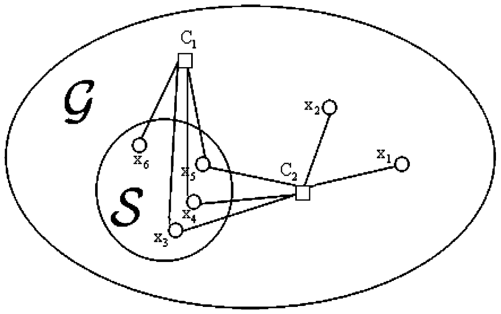


Fig. 7. Outsider nodes.

## VII. Linear Complexity Encoding for General LDPC Codes

In this section, we propose a linear complexity encoding method for general LDPC codes. We will show that any Tanner graph can be decomposed into pseudo-trees and encoding stopping sets that are onefold-constraint or twofold-constraint. By encoding each pseudo-tree or encoding stopping set using Algorithm 2 , Algorithm 5, or Algorithm 7, we achieve linear time encoding for arbitrary LDPC codes.

To proceed, we provide the following definition. Given a Tanner graph $\mathcal{G}$ and its subgraph $\mathcal{S}$, we call the bit nodes in $\mathcal{G}$ but not in $\mathcal{S}$ the *outsider nodes of* $\mathcal{S}$. For example, Fig. 7 shows a Tanner graph $\mathcal{G}$ and its subgraph $\mathcal{S}$. Since in Fig. 7 the two bit nodes $x_1$ and $x_2$ are in $\mathcal{G}$ but not in $\mathcal{S}$, $x_1$ and $x_2$ are outsider nodes of $\mathcal{S}$. The check node $C_2$ involves two outsider nodes of $\mathcal{S}$, i.e., is connected to two outsider nodes. The check node $C_1$ involves zero outsider nodes of $\mathcal{S}$.

We start from LDPC codes with maximum column weight 3 by proving the following lemma.

*Lemma 3:* Assume the maximum bit node degree of a Tanner graph $\mathcal{G}$ is three, then one of the following statements must be true.

(E1) There are no pseudo-encoding stopping sets or encoding stopping sets in $\mathcal{G}$.

(E2) There exists a pseudo-encoding stopping set in $\mathcal{G}$. All the bit nodes in the pseudo-encoding stopping set have uniform degree 2.

(E3) There exists a onefold-constraint or a twofold-constraint encoding stopping set in $\mathcal{G}$. The key check nodes for the encoding stopping set are specified.

*Proof:* We only need to prove either condition (E2), or condition (E3), is true if $\mathcal{G}$ contains a pseudo-encoding stopping set, or an encoding stopping set, respectively. We prove this statement by constructing a subgraph $\mathcal{S}$ from the Tanner graph $\mathcal{G}$. Initially $\mathcal{S}$ is empty. We pick a check node $C_1$ from $\mathcal{G}$ that involves the smallest number of bit nodes. Next, we add $C_1$ and all the bit nodes involved in $C_1$ to $\mathcal{S}$. We keep adding check nodes and all their associated bit nodes to $\mathcal{S}$ till $\mathcal{S}$ contains a pseudo-encoding stopping set or an encoding stopping set. Each time we add a check node to $\mathcal{S}$, we always pick the check node that involves the fewest outsider nodes of $\mathcal{S}$. If $\mathcal{S}$ contains an encoding stopping set, we also add all the check nodes in $\mathcal{G}\backslash\mathcal{S}$ that involves zero outsider nodes to $\mathcal{S}$. Next, we discuss two different cases.

**$\mathcal{S}$ contains an encoding stopping set**. Assume $\mathcal{S}$ contains $k$ check nodes and the $j$th added check node $C_j$ is the last check node that introduces outsider nodes to $\mathcal{S}$. We will show that $k-j \leq 2$. Assume $C_j$ adds $p$ outsider nodes $x_{j_1}, x_{j_2}, \ldots, x_{j_p}$ to $\mathcal{S}$. We will prove that the $(j+1)$th added check node $C_{j+1}$ connects to all the $p$ bit nodes $x_{j_1}, x_{j_2}, \ldots, x_{j_p}$. If $C_{j+1}$ does not connect to all the $p$ bit nodes, then $C_{j+1}$ involves a smaller number of outsider nodes than $C_j$ does and should be added earlier than $C_j$ since we always pick the check node that involves the smallest number of outsider nodes and add it first to $\mathcal{S}$. This contradicts the fact that $C_{j+1}$ is added to $\mathcal{S}$ after $C_j$. Therefore, $C_{j+1}$ should connect to all the $p$ bit nodes $x_{j_1}, x_{j_2}, \ldots, x_{j_p}$. Similarly, $C_{j+2}, \ldots, C_k$ connect to all the $p$ bit nodes $x_{j_1}, x_{j_2}, \ldots, x_{j_p}$. Since any bit node can connect to at most three check nodes, it follows that $k-j \leq 2$, which means at most two check nodes are added to $\mathcal{S}$ after $C_j$. Further, we can prove that $\mathcal{S}$ does not contain any encoding stopping set before adding $C_{j+1}$. The corresponding proof is shown in Appendix C. Hence, the encoding stopping set in $\mathcal{S}$ is either a onefold-constraint encoding stopping set or a twofold-constraint encoding stopping set. The last added $k-j$ check nodes $C_{j+1}, \ldots, C_k$ are key check nodes for the constructed onefold-constraint or twofold-constraint encoding stopping set. Condition (E3) is satisfied.

**$\mathcal{S}$ is a pseudo-encoding stopping set**. It follows that the binary sum of all the check nodes in $\mathcal{S}$ is zero. So, the degree of every bit node in $\mathcal{S}$ is an even number. Since the maximum bit node degree is three, the degree of each bit node in $\mathcal{S}$ is two. Condition (E2) is satisfied.

This completes the proof. □

We detail the method of determining a pseudo-encoding stopping set or an encoding stopping set in Algorithm 8.

**Algorithm 8** Find a pseudo-encoding stopping set or an encoding stopping set (onefold-constraint or twofold-constraint) from a Tanner graph $\mathcal{G}$ with maximum bit node degree 3.

```
$\mathcal{S} = \phi$.
$Flag \leftarrow 0$.
$i = 1$.
while Flag $= 0$ and $\mathcal{G} \neq \mathcal{S}$ do
    Find a check node $C_i$ in $\mathcal{G}\backslash\mathcal{S}$ that involves the smallest
    number of outsider nodes of $\mathcal{S}$.
    Add $C_i$ and all its associated outsider nodes to $\mathcal{S}$.
    if $C_i$ does not introduce new bit nodes to $\mathcal{S}$ then
        $\mathcal{A} \leftarrow \mathcal{S}$.
        while there exists a bit node $x$ of degree one in $\mathcal{A}$
            Delete the degree-one bit node $x$ and the check
            node connecting to $x$ from $\mathcal{A}$.
        end while
        if $\mathcal{A} = \phi$ then
            $\mathcal{S}$ does not contain any pseudo encoding stopping
            set or encoding stopping set.
        else
            Flag $\leftarrow 1$.
        end if
    end if
    $i = i + 1$.
end while
```

```
if $\mathrm{Flag} = 1$ then
    if all the bit nodes in $\mathcal{A}$ are of degree 2 then
        The subgraph $\mathcal{A}$ is a pseudo encoding stopping set.
    else
        The subgraph $\mathcal{A}$ is an encoding stopping set.
    end if
    Output $\mathcal{A}$.
else
    The Tanner graph $\mathcal{G}$ does not contain pseudo-encoding
    stopping sets or encoding stopping sets.
end if
```

Next, we present our main theorem.

*Theorem 3:* Let $\mathcal{G}$ be the Tanner graph of an LDPC code. If the maximum bit node degree of $\mathcal{G}$ is three, then the LDPC code can be encoded in linear time and the encoding complexity is less than $2 \cdot M \cdot (\bar{k} - 1)$ where $M$ is the number of independent check nodes in $\mathcal{G}$ and $\bar{k}$ is the average number of bit nodes involved in each check node.

*Proof:* If the Tanner graph $\mathcal{G}$ does not contain any encoding stopping set, then the corresponding LDPC code can be encoded in linear time by Theorem 2. Therefore, we only need to prove Theorem 3 for the case that $\mathcal{G}$ contains encoding stopping sets. Since the maximum bit node degree of $\mathcal{G}$ is three, by Lemma 3 there exists a pseudo-encoding stopping set or an encoding stopping set $\mathcal{G}_1$ in $\mathcal{G}$. If $\mathcal{G}_1$ is a pseudo-encoding stopping set, we simply delete a redundant check node from $\mathcal{G}_1$ and $\mathcal{G}_1$ becomes a pseudo-tree. If $\mathcal{G}_1$ is an encoding stopping set, it is either a onefold-constraint or a twofold-constraint encoding stopping set by Lemma 3.

Next, we look at the subgraph $\mathcal{G}\backslash\mathcal{G}_1$. We first transform the parity-check equations in $\mathcal{G}\backslash\mathcal{G}_1$ into generalized parity-check equations by moving the bits contained in $\mathcal{G}_1$ from the left-hand side of the equation to the right-hand side of the equation. Let a parity-check equation $C$ involves $k$ bit nodes $x_1, x_2, \ldots, x_k$ where the bits $x_{q+1}, \ldots, x_k$ are also in $\mathcal{G}_1$, then the parity-check equation $C : x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0$ can be rewritten as

$$x_1 \oplus x_2 \oplus \cdots \oplus x_q = x_{q+1} \oplus x_{q+2} \oplus \cdots \oplus x_k = b. \quad (12)$$

In (12), the parameter $b$ becomes a constant after we encode $\mathcal{G}_1$ and get the values of all the bits in $\mathcal{G}_1$. Since the maximum bit node degree of $\mathcal{G}\backslash\mathcal{G}_1$ is less than or equal to three, we, again, find a pseudo-encoding stopping set or an encoding stopping set $\mathcal{G}_2$ from $\mathcal{G}\backslash\mathcal{G}_1$. If $\mathcal{G}_2$ is an encoding stopping set, $\mathcal{G}_2$ is either a onefold-constraint or a twofold-constraint encoding stopping set by Lemma 3. If $\mathcal{G}_2$ is a pseudo-encoding stopping set and we assume $\mathcal{G}_2$ contains the following $m$ generalized parity-check equations:

$$\begin{array}{ccccccc} x_{1,1} & \oplus & x_{1,2} & \oplus & \ldots & x_{1,a_1} & = & b_1 \\ x_{2,1} & \oplus & x_{2,2} & \oplus & \ldots & x_{2,a_2} & = & b_2 \\ \vdots & & \vdots & & & \vdots & & \vdots \\ x_{m,1} & \oplus & x_{m,2} & \oplus & \ldots & x_{m,a_m} & = & b_m \end{array} \quad (13)$$

we derive that

$$b_1 \oplus b_2 \oplus \cdots \oplus b_m = 0. \quad (14)$$

Hence, we can replace any generalized parity-check equation in (13) by the new parity check (14). From the above analysis, we can delete any check node from $\mathcal{G}_2$ to make $\mathcal{G}_2$ a pseudo-tree if $\mathcal{G}_2$ is a pseudo-encoding stopping set. To maintain the code structure, we also generate a new check node $C^*$ that represents the parity check (14). Since the parity check (14) only involves bits in $\mathcal{G}_1$, we add the new check node $C^*$ to $\mathcal{G}_1$ and regenerate encoding stopping sets or pseudo-trees in the graph $\mathcal{G}_1 \cup C^*$.

Generally, we can find a pseudo-encoding stopping set or an encoding stopping set $\mathcal{G}_{i+1}$ from the subgraph $\mathcal{G}\backslash\{\mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \cup \mathcal{G}_i\}$. If $\mathcal{G}_{i+1}$ is an encoding stopping set, $\mathcal{G}_{i+1}$ is either a onefold-constraint or a twofold-constraint encoding stopping set by Lemma 3. If $\mathcal{G}_{i+1}$ is a pseudo-encoding stopping set, we operate in three steps. In the first step, we sum up all the generalized parity-check equations in $\mathcal{G}_{i+1}$ to generate a new parity-check equation $C^*$. In the second step, we delete one check node from $\mathcal{G}_{i+1}$ to make $\mathcal{G}_{i+1}$ a pseudo-tree. In the third step, we add the new check node $C^*$ to $\mathcal{G}_i$ and regenerate pseudo-tree or encoding stopping sets in $\mathcal{G}_i \cup C^*$. Notice that the new parity-check equation $C^*$ in (14) does not incur extra cost to compute variables $b_1, b_2, \ldots, b_m$ since these variables have already been computed in those generalized parity-check equations in $\mathcal{G}_{i+1}$, as shown in (13). Practically, we can compute these variables $b_1$, $b_2, \ldots, b_m$ only once and store them. Later, we can apply the stored values $b_1, b_2, \ldots, b_m$ to both (14) and (13). Hence, the new parity-check equation $C^*$ only needs $m-1$ additional XOR operations to compute the summation of $b_1, b_2, \ldots, b_m$. Since the cost of encoding the pseudo-tree $\mathcal{G}_{i+1}$ is $(m-1) \cdot (\bar{k} - 2)$ where $\bar{k}$ is the average degree of the remaining $m-1$ check nodes in $\mathcal{G}_{i+1}$, the overall cost of encoding $\mathcal{G}_{i+1}$ and the new parity-check equation $C^*$ is $(m-1) \cdot (\bar{k} - 1)$.

By continuing to find pseudo-tree or encoding stopping sets in this way, we reach the stage where $\mathcal{G}\backslash\{\mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \cup \mathcal{G}_i\} = \phi$ or $\mathcal{G}\backslash\{\mathcal{G}_1 \cup \mathcal{G}_2 \cup \ldots \cup \mathcal{G}_i\}$ does not contain pseudo-encoding stopping sets or encoding stopping sets.

By the above analysis, we decompose the Tanner graph $\mathcal{G}$ into a sequence of $p$ subgraphs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_p$ where $\mathcal{G}_i$, $1 \leq i \leq p$, is either a onefold-constraint encoding stopping set, a twofold-constraint encoding stopping set, or a pseudo-tree. If $\mathcal{G}_i$ is a onefold-constraint or a twofold-constraint encoding stopping set, the key check nodes for $\mathcal{G}_i$ are specified by Lemma 3. Therefore, we apply Algorithm 7 or Algorithm 5 to encode $\mathcal{G}_i$ and the resulting encoding complexity is less than $2 \cdot M_i \cdot (\overline{k_i} - 1)$ where $M_i$ denotes the number of independent check nodes in $\mathcal{G}_i$ and $\overline{k_i}$ denotes the average number of bit nodes involved in each check node in $\mathcal{G}_i$. If $\mathcal{G}_i$ is a pseudo-tree, we apply Algorithm 1 to encode $\mathcal{G}_i$ and the corresponding encoding complexity is less than $M_i \cdot (\overline{k_i} - 1)$. The overall computation complexity of encoding $\mathcal{G}$ is linear on the number of independent check nodes $M$ in $\mathcal{G}$ and is bounded by $\sum_{i=1}^{p}(2 \cdot M_i \cdot (\overline{k_i} - 1)) = 2 \cdot M \cdot (\bar{k} - 1)$, where $\bar{k}$ denotes the average number of bits involved in each independent check node of $\mathcal{G}$. This completes the proof. □

We summarize the algorithm of decomposing a Tanner graph with maximum bit node degree 3 into pseudo-trees and encoding stopping sets in Algorithm 9 and the algorithm to encode such LDPC codes in Algorithm 10.

**Algorithm 9** Decompose a Tanner graph $\mathcal{G}$ with maximum bit node degree 3 into onefold-constraint encoding stopping sets, twofold-constraint encoding stopping sets, and pseudo-trees.

```
Find a pseudo-encoding stopping set or an encoding
stopping set $\mathcal{G}_1$ from $\mathcal{G}$ using Algorithm 8.
$\mathcal{G} = \mathcal{G}\backslash\mathcal{G}_1$.
**if** $\mathcal{G}_1$ is a pseudo-encoding stopping set **then**
    Delete a check node in $\mathcal{G}_1$. $\mathcal{G}_1$ becomes a pseudo-tree.
**end if**
$i = 1$.
**while** there exists a pseudo-encoding stopping set or an
encoding stopping set in $\mathcal{G}$ **do**
    Find a pseudo-encoding stopping set or an encoding
    stopping set $\mathcal{G}_{i+1}$ from $\mathcal{G}$ using Algorithm 8. Assume
    $\mathcal{G}_{i+1}$ contain $m$ check nodes $C_1, C_2, \ldots, C_m$.
    $\mathcal{G} = \mathcal{G}\backslash\mathcal{G}_{i+1}$
    **if** $\mathcal{G}_{i+1}$ is a pseudo-encoding stopping set **then**
        $\mathcal{G}_{i+1} = \mathcal{G}_{i+1}\backslash C_m$. $\mathcal{G}_{i+1}$ becomes a pseudo-tree.
        Generate a new check node $C^* = C_1 \oplus C_2 \oplus \cdots \oplus C_m$.
        Add $C^*$ to $\mathcal{G}_i$ and regenerate pseudo-trees and
        encoding stopping sets in $\mathcal{G}_i \cup C^*$.
    **end if**
    $i = i + 1$.
**end while**
$\mathcal{G}_{i+1} = \mathcal{G}$.
Output a sequence of subgraphs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_p$ where $\mathcal{G}_i$,
$1 \le i \le p$, is either a pseudo-tree or an encoding stopping
set (onefold-constraint or twofold-constraint.)
```

**Algorithm 10** Linear complexity encoding algorithm for LDPC codes with maximum bit node degree 3

```
**Preprocessing (carry out only once):**
Apply Algorithm 9 to decompose the Tanner graph
$\mathcal{G}$ of the code into $p$ subgraphs $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_p$
where $\mathcal{G}_i$, $i = 1, 2, \ldots, p$, is either a pseudo-tree
or a onefold-constraint encoding stopping set or a
twofold-constraint encoding stopping set with the key check
nodes being specified.
**Encoding:**
**for** $i = 1$ to $p$ **do**
    Compute the constants on the right-hand side of the
    generalized parity-check equations of $\mathcal{G}_i$ based on the
    already known bit values of $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{i-1}$.
    **if** $\mathcal{G}_i$ is a pseudo-tree **then**
        Encode $\mathcal{G}_i$ using Algorithm 1.
    **else**
        **if** $\mathcal{G}_i$ is a onefold-constraint encoding stopping set
        **then**
            Encode $\mathcal{G}_i$ using Algorithm 7.
        **else**
            Encode $\mathcal{G}_i$ using Algorithm 5.
        **end if**
    **end if**
**end for**
Output the encoded codeword.
```

Next, we extend the linear time encoding method described in Theorem 3 to LDPC codes with arbitrary column weight and row weight.

*Theorem 4:* Any LDPC code $\mathcal{C}$ with arbitrary column weight distribution and row weight distribution can be encoded in linear time, and the corresponding encoding complexity is less than $4 \cdot M \cdot (\bar{k} - 1)$ where $M$ is the number of independent check nodes in $\mathcal{C}$ and $\bar{k}$ is the average degree of check nodes.

*Proof:* We first show that an LDPC code with arbitrary column weight distribution and row weight distribution can be equivalently transformed into an LDPC code with maximum column weight three. For example, Fig. 8 on the top shows a bit node $x$ of degree 4. It can be split into two bit nodes $x$ and $x'$ of degree 3 and an auxiliary check node $C'$, as shown on the bottom in Fig. 8. The auxiliary check node $C'$ is represented as $x \oplus x' = 0$, which means $x$ is equivalent to $x'$. Originally, the bit node $x$ connects to four check nodes $C_1$, $C_2$, $C_3$, and $C_4$. After node splitting, $x'$ connects to $C_1$, $C_2$, and $x$ connects to $C_3$, $C_4$. Hence, the Tanner graph on the top in Fig. 8 is equivalent to the Tanner graph on the bottom in Fig. 8. Similarly, a bit node of degree 5 can be split into three bit nodes $x$, $x'$, and $x''$ and two auxiliary check nodes $C'$ and $C''$, as shown in Fig. 9. Generally, a bit node of degree $k$ can be equivalently transformed into $k-2$ bit nodes of degree 3 and $k-3$ auxiliary check nodes, as shown in Fig. 10. Assume an LDPC code $\mathcal{C}$ contains $M$ check nodes and $N$ bit nodes. The $M$ check nodes have degrees $k_1, k_2, \ldots, k_M$, respectively. The $N$ bit nodes have degrees $j_1, j_2, \ldots, j_N$, respectively. Among the $N$ bit nodes in $\mathcal{C}$, there are $s$ bit nodes whose degrees are greater than 3 and their degrees are $j_1, j_2, \ldots, j_s$. This LDPC code can be equivalently transformed into another LDPC code $\mathcal{C}'$ with maximum column weight 3. The new code $\mathcal{C}'$ has $M + (\sum_{i=1}^{s} j_i - 3s)$ check nodes and $N + (\sum_{i=1}^{s} j_i - 3s)$ bit nodes. By Theorem 3, the LDPC code $\mathcal{C}'$ can be encoded in linear time and the encoding complexity is less than $2 \cdot M' \cdot (\overline{k'} - 1)$, where $M'$ is the number of independent check nodes in $\mathcal{C}'$ and $\overline{k'}$ is the average degree of independent check nodes in $\mathcal{C}'$. Since there are $\sum_{i=1}^{s} j_i - 3s$ auxiliary check nodes in $\mathcal{C}'$ that have degree 2, we derive that

$$\begin{aligned} &2 \cdot M' \cdot (\overline{k'} - 1) \\ &\quad = 2 \cdot \left( M \cdot (\bar{k} - 1) + \left( \sum_{i=1}^{s} j_i - 3s \right) \cdot (2 - 1) \right) \\ &\quad < 2 \cdot \left( M \cdot (\bar{k} - 1) + \left( \sum_{i=1}^{N} j_i - N \right) \right) \\ &\quad < 2 \cdot \left( M \cdot (\bar{k} - 1) + \left( \sum_{i=1}^{M} k_i - M \right) \right) \\ &\quad = 4 \cdot M \cdot (\bar{k} - 1) \end{aligned} \tag{15}$$

Therefore, the overall computation cost of encoding $\mathcal{C}'$ is less than $4 \cdot M \cdot (\bar{k} - 1)$. As the LDPC code $\mathcal{C}$ is equivalent to

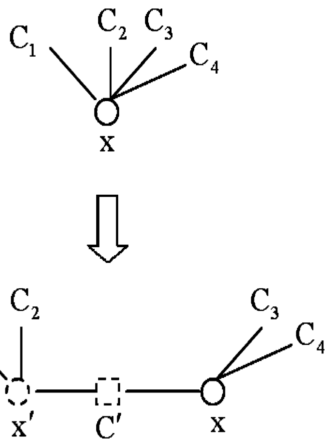


Fig. 8. Transform a bit node of degree 4 into two bit nodes of degree 3 and an auxiliary check node of degree 2.

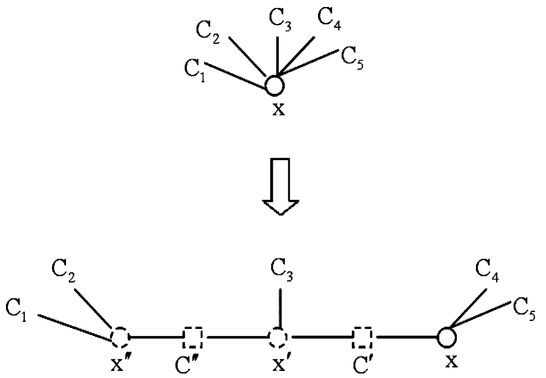


Fig. 9. Transform a bit node of degree 5 into three bit nodes of degree 3 and two auxiliary check nodes of degree 2.

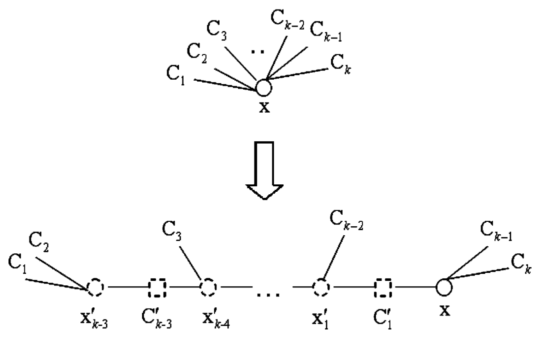


Fig. 10. Transform a bit node of degree $k$ into $k-2$ bit nodes of degree 3 and $k-3$ auxiliary check nodes of degree 2.

the LDPC code $\mathcal{C}'$, the complexity of encoding $\mathcal{C}$ is less than $4 \cdot M \cdot (\bar{k}-1)$. This completes the proof. □

Let us look at an example. The parity-check matrix of a $(13, 26)$ LDPC code with column weight 3 is shown in Fig. 11. Assume the values of the 13 information bits are $0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1$, and 1. We apply the proposed linear complexity encoding method to encode this code.

**Preprocessing**. We construct an encoding stopping set from the LDPC code shown in Fig. 11 using Algorithm 8. We start from an empty graph $\mathcal{S}$ and add check nodes and their associated bit nodes to $\mathcal{S}$. Each time we add a check node, we always pick the check node that involves the smallest number of outsider nodes of $\mathcal{S}$. After adding seven check nodes, the resulting graph is a pseudo-tree, as shown in Fig. 12. When nine check nodes are considered, we get the twofold-constraint encoding stopping set $\mathcal{E}_1$ with key check nodes $C_{10}$ and $C_{12}$ shown in Fig. 13. The bits $x_9$, $x_{13}$, $x_{22}$, $x_{23}$, $x_5$, $x_{16}$, $x_4$, $x_{12}$, $x_{17}$ are information bits. The two bit nodes $x_1$ and $x_{18}$ are chosen to be reevaluated bits by Algorithm 11.

After finding the encoding stopping set $\mathcal{E}_1$, the remaining Tanner graph of the code can be constructed to be a twofold-constraint encoding stopping set $\mathcal{E}_2$ with key check nodes $C_3$ and $C_5$, as shown in Fig. 14. Therefore, the LDPC code can be partitioned into two encoding stopping sets $\mathcal{E}_1$ and $\mathcal{E}_2$ that are shown in Fig. 14. The bits $x_{11}$, $x_{15}$, $x_{19}$, $x_{25}$ in the encoding stopping set $\mathcal{E}_2$ are information bits. The two bit nodes $x_3$ and $x_6$ are chosen to be reevaluated bits of $\mathcal{E}_2$ by Algorithm 11.

**Encoding**.

**Encode $\mathcal{E}_1$**:

Step 1. Fill the values of the information bits, i.e., $[x_9\ x_{13}\ x_{22}\ x_{23}\ x_5\ x_{16}\ x_4\ x_{12}\ x_{17}] = [0\,1\,1\,1\,0\,1\,1\,0\,0]$. Assign $x_1 = 0$ and $x_{18} = 0$.

Step 2. Encode the pseudo-tree shown in Fig. 12. Compute the parity bits $x_{21}$, $x_{14}$, $x_8$, $x_7$, $x_{10}$, $x_{24}$, $x_2$ as follows:

$$\begin{aligned} xsx_{21} &= x_1 \oplus x_9 \oplus x_{13} \oplus x_{22} \oplus x_{23} = 1 \\ x_{14} &= x_9 \oplus x_{13} \oplus x_5 \oplus x_{16} \oplus x_{18} = 0 \\ x_8 &= x_{23} \oplus x_{16} \oplus x_{18} \oplus x_4 \oplus x_{12} = 1 \\ x_7 &= x_{13} \oplus x_{23} \oplus x_5 \oplus x_{12} \oplus x_{17} = 0 \\ x_{10} &= x_1 \oplus x_{22} \oplus x_5 \oplus x_4 \oplus x_{17} = 0 \\ x_{24} &= x_{21} \oplus x_{14} \oplus x_8 \oplus x_7 \oplus x_{10} = 0 \\ x_2 &= x_{22} \oplus x_{24} \oplus x_{16} \oplus x_7 \oplus x_4 = 1. \end{aligned}$$

Step 3. Compute the values of the key parity-check equations $C_{10}$ and $C_{12}$. $\widetilde{C_{10}} = x_2 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{18} = 1$, and $\widetilde{C_{12}} = x_1 \oplus x_2 \oplus x_8 \oplus x_{17} \oplus x_{21} = 1$.

Step 4. Since $\widetilde{C_{10}} = 1$ and $\widetilde{C_{12}} = 1$, the correct values of the reevaluated bits $x_1$ and $x_{18}$ are $x_1 = \widetilde{C_{10}} = 1$ and $x_{18} = \widetilde{C_{12}} = 1$.

Step 5. Recompute the parity bits $x_{21}$, $x_{14}$, $x_8$, and $x_{10}$ based on the new values of $x_1$ and $x_{18}$. We derive that $x_{21} = 0$, $x_{14} = 1$, $x_8 = 0$, $x_{10} = 1$.

**Encode $\mathcal{E}_2$**:

Step 6. Fill the values of the information bits, i.e., $[x_{11}\ x_{15}\ x_{19}\ x_{25}] = [1\,0\,1\,1]$. Assign $x_3 = 0$ and $x_6 = 0$.

Step 7. Compute the parity bits $x_{20}$, $x_{26}$ as follows:

$$\begin{aligned} x_{20} &= x_3 \oplus x_6 \oplus x_{11} \oplus x_{15} \oplus x_{19} = 0 \\ x_{26} &= x_3 \oplus x_6 \oplus x_{11} \oplus x_{19} \oplus x_{25} \oplus x_{24} = 1. \end{aligned}$$

Notice that the value of the parity bit $x_{26}$ is based on the value of the bit $x_{24}$ in $\mathcal{E}_1$.

Step 8. Compute the values of the key parity-check equations $C_3$ and $C_5$. $\widetilde{C_3} = x_3 \oplus x_{20} \oplus x_{11} \oplus x_{15} \oplus x_{26} \oplus x_{25} = 1$, and $\widetilde{C_5} = x_6 \oplus x_{20} \oplus x_{15} \oplus x_{19} \oplus x_{26} \oplus x_{25} = 1$.

$$
\mathbf{H} = \begin{bmatrix}
1&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&1&1&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&1&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&1&1\\
0&0&0&0&0&0&1&1&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1&1\\
0&0&0&0&1&0&1&0&0&0&0&1&1&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&1&0&0&0&1&1&0&1&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1\\
0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&1&0&0\\
0&1&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&1&0&0&0&1&0&0&0&1&1&0&0&0&0&0&0\\
1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0\\
1&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0
\end{bmatrix}
$$

Fig. 11. The parity-check matrix of a $(13, 26)$ LDPC code.

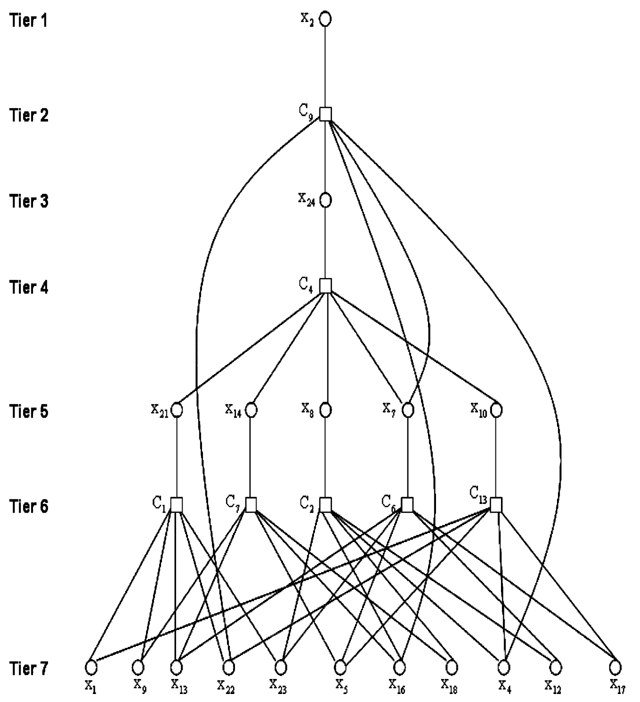
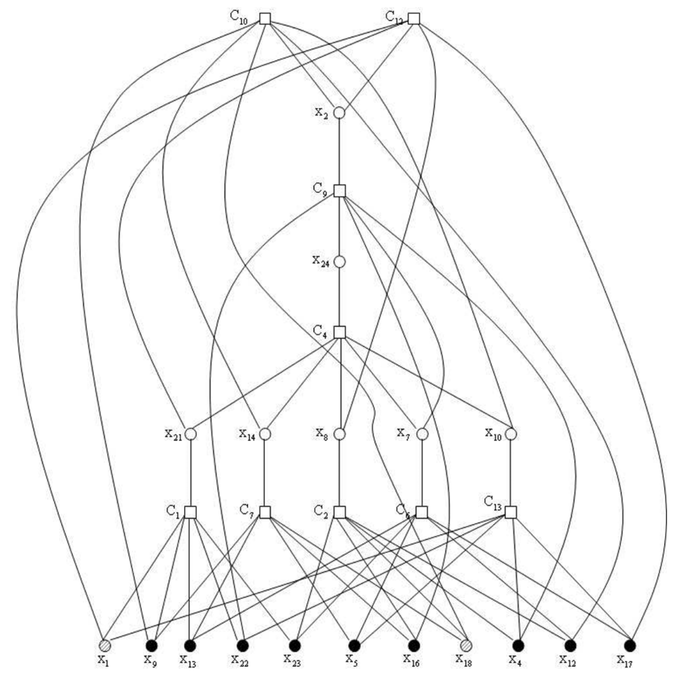


Fig. 12. A pseudo-tree built from the LDPC code described in Fig. 11.



● Bit node labeled as an information bit

▨ Bit node labeled as a reevaluated bit

○ Bit node labeled as a parity bit

□ Check node

Fig. 13. An encoding stopping set developed from the LDPC code described in Fig. 11.

Step 9. Since $\widetilde{C_3} = 1$ and $\widetilde{C_5} = 1$, the correct values of the reevaluated bits $x_3$ and $x_6$ are $x_3 = \widetilde{C_3} = 1$ and $x_6 = \widetilde{C_5} = 1$.

The encoded codeword is $[x_9\ x_{13}\ x_{22}\ x_{23}\ x_5\ x_{16}\ x_4\ x_{12}\ x_{17}\ x_1\ x_{18}\ x_{21}\ x_{14}\ x_8\ x_7\ x_{10}\ x_{24}\ x_2\ x_{11}\ x_{15}\ x_{19}\ x_{25}\ x_3\ x_6\ x_{20}\ x_{26}] = [0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1]$.

## VIII. Conclusion

This paper proposes a linear complexity encoding method for general LDPC codes by analyzing and encoding their Tanner graphs. We show that two particular types of Tanner graphs: pseudo-trees and encoding stopping sets can be encoded in linear time. Then, we prove that any Tanner graph can be decomposed into pseudo-trees and encoding stopping sets. By encoding the pseudo-trees and encoding stopping sets in a sequential order, we achieve linear complexity encoding for arbitrary LDPC codes. The proposed method can be applied to a wide range of codes; it is not limited to LDPC codes. It is applicable to both regular LDPC codes and irregular LDPC

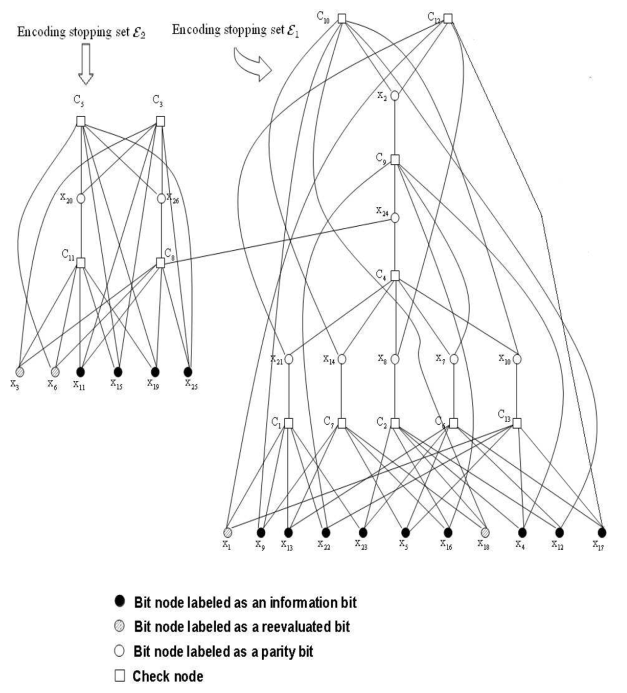

Encoding stopping set $\mathcal{E}_2$
Encoding stopping set $\mathcal{E}_1$
● Bit node labeled as an information bit
◎ Bit node labeled as a reevaluated bit
○ Bit node labeled as a parity bit
□ Check node

Fig. 14. Two encoding stopping sets developed from the LDPC code described in Fig. 11.

codes. In fact, the proposed linear time encoding method is applicable to any type of block codes. It removes the problem of high encoding complexity for all long block codes that historically are commonly encoded by matrix multiplication.

## Appendix A
## Proof of Lemma 2

We prove Lemma 2 by contradiction. Assume a pseudo-tree $\mathcal{T}$ contains an encoding stopping set $\mathcal{E}_f$. Let $C'$ be an arbitrary check node in the encoding stopping set $\mathcal{E}_f$ and $C'$ is located in the $2i$th tier of $\mathcal{T}$. According to condition (B1), all the bit nodes that connect to $C'$, which include the parent bit node of $C'$, are also in the encoding stopping set $\mathcal{E}_f$. By condition (B2), there exists another check node $C''$ in $\mathcal{E}_f$ that connects to the parent bit node of $C'$. By condition (A4), the check node $C''$ must be in a tier above $2i-1$. By the above reasoning, we conclude that the encoding stopping set $\mathcal{E}_f$ must contain a check node above the $2i$th tier if $\mathcal{E}_f$ contains a check node in the $2i$th tier. By repeatedly using the above conclusion, we derive that the encoding stopping set $\mathcal{E}_f$ must contain a check node $C'''$ in tier 2. By condition (B1), the parent bit node of $C'''$, which is in the first tier of $\mathcal{T}$, is also in the encoding stopping set $\mathcal{E}_f$. However, by condition (A2), the parent bit node of $C'''$ has degree one, which violates condition (B2) that every bit node in $\mathcal{E}_f$ has degree greater than or equal to two. Therefore, the assumption is wrong. Any pseudo-tree cannot contain an encoding stopping set. Similarly, we can prove that the union of pseudo-trees cannot contain any encoding stopping set. This completes the proof. □

## Appendix B
## Finding Reevaluated Bits $x_\gamma$ and $x_\delta$ in a Twofold-Constraint Encoding Stopping Set With Key Check Nodes $C_\alpha$ and $C_\beta$

The details are described in Algorithm 11.

**Algorithm 11** Finding reevaluated bits $x_\gamma$ and $x_\delta$ in a twofold-constraint encoding stopping set with key check nodes $C_\alpha$ and $C_\beta$.

```
Transform the two key check equations $C_\alpha$ and $C_\beta$ into
equivalent parity-check equations $C'_\alpha$ and $C'_\beta$ that involve
only information bits. Assume $C'_\alpha$ involves $q$ information
bits $x_{\alpha_1}, x_{\alpha_2}, \ldots, x_{\alpha_q}$, and $C'_\beta$ involves $p$ information bits
$x_{\beta_1}, x_{\beta_2}, \ldots, x_{\beta_p}$.
$Flag \leftarrow 0$.
for $i = 1$ to $q$ do
    if $x_{\alpha_i}$ is involved in $C'_\alpha$ but not in $C'_\beta$ then
        $Flag \leftarrow 1$.
        Choose the reevaluated bit $x_\gamma$ to be $x_\gamma = x_{\alpha_i}$. Exit
        the for loop.
    end if
end for
if $Flag = 1$ then
    Choose the reevaluated bit $x_\delta$ to be $x_\delta = x_{\beta_p}$.
else
    Choose the reevaluated bit $x_\gamma$ to be $x_\gamma = x_{\alpha_q}$.
    for $i = 1$ to $p$ do
        if $x_{\beta_i}$ is involved in $C'_\beta$ but not in $C'_\alpha$ then
            Choose the reevaluated bit $x_\delta$ to be $x_\delta = x_{\beta_i}$. exit
            the for loop.
        end if
    end for
end if
Output the two chosen reevaluated bits $x_\gamma$ and $x_\delta$.
```

Next, we prove that Algorithm 11 can successfully find two bits $x_\gamma$ and $x_\delta$ that satisfy conditions (D1) to (D3).

*Proof:* We discuss two different cases.

**Algorithm 11 chooses a bit $x_\gamma$ that is involved in $C'_\alpha$ but not in $C'_\beta$.** Since the bit $x_\gamma$ is directly involved in the parity-check equations $C'_\alpha$ and $C'_\alpha$ is equivalent to $C_\alpha$, the value change of $x_\gamma$ alone will affect the value of $C_\alpha$. Hence, condition (D1) is satisfied. Since Algorithm 11 also chooses another bit $x_\delta$ that is directly involved in the parity-check equation $C'_\beta$ and $C'_\beta$ is equivalent to $C_\beta$, the value change of $x_\delta$ alone will affect the value of $C_\beta$. Hence, condition (D2) is satisfied. Since the parity-check equation $C'_\beta$ only involves the bit $x_\delta$ but not the bit $x_\gamma$ and $C'_\beta$ is equivalent to $C_\beta$, the value changes of both $x_\delta$ and $x_\gamma$ will change the value of $C_\beta$. Hence, condition (D3) is also satisfied.

**Algorithm 11 chooses a bit $x_\gamma$ that is involved in both $C'_\alpha$ and $C'_\beta$.** Since the bit $x_\gamma$ is directly involved in the parity-check equations $C'_\alpha$ and $C'_\alpha$ is equivalent to $C_\alpha$, the value change of $x_\gamma$ alone will affect the value of $C_\alpha$. Hence, condition (D1) is satisfied. Since Algorithm 11 also chooses another bit $x_\delta$ that is directly involved in the parity-check equation $C'_\beta$ but not in $C'_\alpha$ and $C'_\beta$ is equivalent to $C_\beta$, the value change of $x_\delta$ alone will affect the value of $C_\beta$. Hence, condition (D2) is satisfied. Since the parity-check equation $C'_\alpha$ only involves the bit $x_\gamma$ but not the bit $x_\delta$ and $C'_\alpha$ is equivalent to $C_\alpha$, the value changes of both $x_\delta$ and $x_\gamma$ will change the value of $C_\alpha$. Hence, condition (D3) is also satisfied.

This completes the proof. □

## Appendix C
## In the Proof of Lemma 3, the Subgraph $\mathcal{S}$ Does not Contain any Encoding Stopping set Before Adding a Check Node that Involves Zero Outsider Nodes

*Proof:* We prove this fact by contradiction. Assume the subgraph $\mathcal{S}$ contains an encoding stopping set $\mathcal{E}_f$ before adding a check node that involves zero outsider nodes. Let the encoding stopping set $\mathcal{E}_f$ contain $m$ check nodes $C_1, C_2, \ldots, C_{m-1}, C_m$, and the check node $C_m$ is being added to the subgraph $\mathcal{S}$ later than all the other check nodes $C_1, C_2, \ldots, C_{m-1}$ in $\mathcal{E}_f$. From the construction process of $\mathcal{S}$, all the bit nodes that are involved in check nodes $C_1, C_2, \ldots, C_{m-1}$ have already been in $\mathcal{S}$ before the check node $C_m$ is being added to $\mathcal{S}$. By definition, all the bit nodes that are involved in check nodes $C_1, C_2, \ldots, C_{m-1}$ are not outsider nodes when $C_m$ is being added to $\mathcal{S}$. Further, $C_m$ introduces at least one outsider node $x_o$ to $\mathcal{S}$. Therefore, the outsider node $x_o$ is not connected to any of the check nodes in the encoding stopping set $\mathcal{E}_f$ except for $C_m$. Since $x_o$ connects to the check node $C_m$ and $C_m$ is in the encoding stopping set $\mathcal{E}_f$, $x_o$ is also in $\mathcal{E}_f$ by condition (B1). However, $x_o$ is only connected to the check node $C_m$ in $\mathcal{E}_f$, which violates condition (B2) that every bit node in an encoding stopping set is connected to at least two check nodes in the encoding stopping set. Hence, the assumption is wrong. The subgraph $\mathcal{S}$ does not contain any encoding stopping set before adding a check node that involves zero outsider nodes. This completes the proof. □

## Acknowledgment

The authors would like to thank the anonymous reviewers for their suggestions which significantly improved the presentation of the results in this paper and to Dr. Igal Sason 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, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

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

[4] J. Lu, J. M. F. Moura, and H. Zhang, "Efficient encoding of cycle codes: A graphical approach," in *Proc. 37th Asilomar Conf. Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2003, pp. 69–73.

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

[6] T. Mittelholzer, "Efficient encoding and minimum distance bounds of Reed-Solomon-type array codes," in *Proc. IEEE Int. Symp. Information theory (ISIT 2002)*, Lausanne, Switzerland, Jun. 2002, p. 282.

[7] J. Lu and J. M. F. Moura, "TS-LDPC codes: Turbo-structured codes with large girth," *IEEE Trans. Inf. Theory*, vol. 53, no. 3, pp. 1080–1094, Mar. 2007.

[8] 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.

[9] S. J. Johnson and S. R. Weller, "A family of irregular LDPC codes with low encoding complexity," *IEEE Commun. Lett.*, vol. 7, no. 2, pp. 79–81, Feb. 2003.

[10] S. Freundlich, D. Burshtein, and S. Litsyn, "Approximately lower triangular ensembles of LDPC codes with linear encoding complexity," *IEEE Trans. Inf. Theory*, vol. 53, no. 4, pp. 1484–1494, Apr. 2007.

[11] M. Yang, W. E. Ryan, and Y. Li, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 564–570, Apr. 2004.

[12] D. Haley, A. Grant, and J. Buetefuer, "Iterative encoding of low-density parity-check codes," in *Proc. IEEE Globecom 2002*, Taipei, Taiwan, ROC, Nov. 2002, vol. 2, pp. 1289–1293.

[13] D. Haley and A. Grant, "High rate reversible LDPC codes," in *Proc. 5th Australian Communications Theory Workshop*, Newcastle, Australia, Feb. 2004, pp. 114–117.

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

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

[16] C. Di, D. Proietti, E. Telatar, T. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

**Jin Lu** (S'01–M'06) received the bachelor and master degrees from Tsinghua University, Beijing, China, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 2004.

Then he joined Sun Microsystems, Broomfield, CO, as a researcher. His research interests focus on developing and designing structured LDPC codes with large girth, efficient LDPC encoding and decoding algorithms, hardware implementation of LDPC codes, and advanced signal detection and timing recovery schemes for high-density magnetic recording channels. He holds four issued or pending patents.

**José M. F. Moura** (S'71-M'75-SM'90-F'94) received the engenheiro electrotécnico degree from Instituto Superior Técnico (IST), Lisbon, Portugal, and the M.Sc., E.E., and the D.Sc. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge.

He is a Professor of Electrical and Computer Engineering and of BioMedical Engineering at Carnegie Mellon University, Pittsburgh, PA, and a founding Codirector of the Center for Sensed Critical Infrastructures Research (CenSCIR). During 2006–2007, he is a Visiting Professor of Electrical Engineering at MIT. He was on the faculty at IST (1975–1984) and has held visiting faculty appointments at MIT (1984–1986 and 1999–2000) and as a research scholar at the University of Southern California (USC), Los Angeles (summers of 1978–1981). His research interests include statistical and algebraic signal processing and digital communications, on which he has published extensively and he holds six patents.

Dr. Moura is the President Elect of the IEEE Signal Processing Society (SPS) and was the Editor-in-Chief for the IEEE Transactions in Signal Processing (1975–1999). He has been on the Editorial Board of the Proceedings of the IEEE, the IEEE Signal Processing Magazine, and the *ACM Transactions on Sensor Networks* and on the program committee of numerous Conferences and Workshops. He is a Fellow of the American Association for the Advancement of Science (AAAS), and a corresponding member of the Academy of Sciences of Portugal (Section of Sciences). He was awarded the 2003 IEEE Signal Processing Society meritorious service award and in 2000 the IEEE Millennium Medal. He is affiliated with several IEEE societies, Sigma Xi, AMS, AAAS, IMS, and SIAM.