Chapter 4 Basic Notions of Regular Low-Density Parity-Check Codes

4.1 Introduction

In this chapter we will introduce a new kind of iteratively decodable codes, i.e., low-density parity-check (LDPC) codes, which was originally proposed by Gallager in his early thesis in 1963 [1]. One of the important innovations was the introduction of an iterative decoding algorithm (iterative probabilistic decoding [1]) that is now widely recognized as a special case of the notable belief propagation (BP) algorithm, i.e., BP-based approach. The BP algorithm can be applied to decode a more general category of iteratively decodable codes, including the previously mentioned turbo codes, serially concatenated codes as well as the LDPC codes. The decoder using BP algorithm is what we now called message-passing decoder [7][8][9]. Meanwhile, Gallager showed that his approach is capable of achieving a significant fraction of channel capacity at relatively low computational complexity. This class of LDPC codes of Gallager's type is now regarded as regular LDPC codes, which will be introduced explicitly in the next section.

For a quite long period of almost three decades between the first invention of LDPC and the advent of turbo codes, unfortunately, LDPC and their variants were largely neglected and few of researchers were deeply concerned with this subject. The notable exceptions are the work of Tanner, Zyablov, Pinsker, and Margulis [2-4].

Interest in LDPC codes greatly spurs the researchers to carry out further extensive studies, especially in the wake of the successfully discovery of turbo codes. Recently, LDPC codes were independently rediscovered by some pioneer researchers, such as MacKay and Neal [5][6], and Richardson and Urbanke [7-9], etc. They have brought many new achievements in this area during the past few years. First, new analytical tools and decoding algorithms were proposed for the throughout exploitations of LDPC codes in binary-symmetric channel (BSC) (under hard-decision message-passing decoding), binary-erasure channel (BEC), binary-input AWGN channel, and binary-input Laplacian channel, etc. [9]. Second, the regular LDPC codes are extended to a boarder class of category, i.e., irregular LDPC codes. Meanwhile, it has been reported that, asymptotically in the block length, the LDPC codes, in particular for the irregular LDPC codes, can extremely approach to the theoretical capacity over many noisy channels.

LDPC codes and turbo codes are similar in many respects. They both belong to the general class of powerful concatenated codes employing pseudo-random encoders and iterative decoders. But in many ways, LDPC codes can be considered serious competitors to turbo codes. In particular, LDPC codes exhibit an asymptotically better performance than turbo codes and they admit a wide range of tradeoffs between performance and decoding complexity. For encoding, one major drawback of the LDPC codes is their apparently high encoding complexity. Turbo codes can be encoded in linear time, while a straightforward encoder implementation for an LDPC code has complexity quadratic in the block length [7]. For decoding of an LDPC code, each check-parity usually involves a few bits compared with the typically very long block length [1]. Therefore, the reliable information builds up relatively slowly during the iterative decoding process, and global convergence is also slow. Whereas each constituent encoder of a turbo code, similar as the check-parity of the LDPC code, concerns with all block of information bits. Hence, the reliable information built up in the iterative decoding can achieve a fast global convergence. However, a turbo decoder, based on the BCJR algorithm [10], usually needs more memory and more operations per iteration as more information bits concerned for constituent decoding than an LDPC decoder. Due to these essential similarities between the both cases, it is really not surprising that a turbo code can be categorized as a special sort of LDPC codes [5][6].

As the performance comparison between the LDPC codes and turbo codes, for very long interleaver lengths, e.g., greater than 10000, it has been reported by some authors [8][12][26] that irregular LDPC codes can outperform the turbo codes. But with the shorter lengths, turbo codes still have better performance.

For instance, a rate-1/2 irregular LDPC code, constructed in [8], is only 0.13dB away from the Shannon capacity over the binary-input additive white Gaussian (BI-AWGN) channel conditioned on the block length 10^6 and a BER of 10^{-6}, surpassing the best turbo codes known so far. The threshold, which indicates the performance for infinite lengths, is a mere 0.06dB away from the capacity limit. Furthermore, for the same coding rate, BER and noisy channel as above, another well-designed irregular LDPC code in [12] can further achieve within 0.04dB from the capacity with a block length of 10^7. The corresponding threshold is within 0.0045dB of the capacity limit! For a rate-1/2 regular LDPC code of block length 10^4 over the same channel [9], it requires an SNR of 1.4dB to reach a BER of 10^{-5}, while an equivalent turbo code with comparable complexity achieves the same performance at an approximate SNR of 0.8dB.

In this chapter, we will briefly introduce the traditional regular LDPC codes of Gallager's type, including the mathematics definition of regular LDPC codes, basic encoding methods and the complexity analysis for LDPC encoding. These essential materials form the fundamental basis for further study of regular and irregular LDPC codes, their optimized designs and performance over noisy channels.

4.2 Regular LDPC Codes of Gallager's Type

In this section, we will first give a general definition of the regular LDPC codes of Gallager's type. Second, some efficient encoding methods for the regular LDPC codes are briefly introduced. Finally, the iterative decoding (or message-passing decoding) of the regular LDPC codes is addressed.

4.2.1 Basic Notions of Regular LDPC Codes

Before we present the definition of regular LDPC codes in a strict mathematical form, let us first review two examples of the simple parity-check codes.

Example: 4.1: Consider a very simple binary parity-check code that is represented by a parity-check matrix depicted in Fig. 4.1. The codewords of a parity-check code are formed by combining a block of binary information digits with a block of check digits.

Each check digit is modulo-2 sum of a pre-specified set of information digits. Apparently, this is a well-known systematic block (7, 4) Hamming code. Each codeword bit participates in 1, 2 or 3 parity-check equations and every such check equation involves 4 codeword bits. For instance, codeword bit d_1 participates all three parity checks, while d_1 only attends first check. There are altogether 12 ones among the 3×7=21 entries in the whole parity check matrix. □

$$\text{Parity-check equations}\left\{\begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \left[\begin{matrix} 1 \\ 1 \\ 1 \end{matrix}\right. & \begin{matrix} 1 \\ 1 \\ 0 \end{matrix} & \begin{matrix} 1 \\ 0 \\ 1 \end{matrix} & \begin{matrix} 0 \\ 1 \\ 1 \end{matrix} & \begin{matrix} 1 \\ 0 \\ 0 \end{matrix} & \begin{matrix} 0 \\ 1 \\ 0 \end{matrix} & \left.\begin{matrix} 0 \\ 0 \\ 1 \end{matrix}\right] \end{array}\right. \begin{array}{l} \\ x_5 = x_1 + x_2 + x_3 \\ x_6 = x_1 + x_2 + x_4 \\ x_7 = x_1 + x_3 + x_4 \end{array}$$

Fig. 4.1 An Example of a parity-check matrix of a systematic (7, 4) Hamming code.

Example 4.2: If we further study a special binary parity-check code whose parity-check matrix $H_{15\times20}$ is demonstrated in Fig. 4.2. This matrix has 20 columns with 3 ones in each column, 4 ones in each row, and zeros elsewhere, and the total number of the parity-check equations is 20×3/4=15. In other words, every codeword bit participate in exactly 3 parity-check equations, and every such equation involves exactly 4 codeword bits.

$$H_{15\times 20}=\left(\begin{array}{cccccccccccccccccccc}
\hline
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1\\
\hline
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0\\
0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1\\
\hline
1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&0\\
0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1\\
\hline
\end{array}\right)$$

Fig. 4.2 An example of a parity-check matrix of block code of length 20 with 4 ones in each row and 3 ones in each column.

The entire matrix is divided into 3 sub-matrices, each of which contains a single 1 in each column. The first of these submatrices contains all its 1's in descending order; that is, the i-th ($1\leq i\leq 5$) row contains 1's in columns $(i-1)k+1$ to ik, where k is equal to 5. The other submatrices are merely column permutations of the first subject to the constraint of 4 bits in each row. One important characteristic is that there are only $4\times 15= 60$ ones, which is rather sparse compared with $20\times 15=300$ entries of the whole matrix.

Let $C^{20}(3, 4)$ be the *ensemble* of the codes with parity-check matrices, of which there are 3 and 4 ones in each column and row, respectively. Obviously, the above mentioned particular parity-check code belongs to the $C^{20}(3, 4)$. □

Except for the aforementioned design method, there are many other effective ways to construct the parity-check codes of the *ensemble* $C^{20}(3,4)$. Pseudorandom approach is one of them. For a weight-4 sequence of length 20, there has $\binom{20}{4}$ distinct permutations, each row in a parity check matrix is resulted in by randomly selecting one of these permutations of the weight-4 sequence provided that the condition of 3 ones in each column is satisfied. Similarly, we can use the permutations of a weight-3 sequence of length 15 to construct the columns of the matrix and all the generated rows must meets the requirement of only 4 ones in each row.

Now we can define a broad family of binary parity-check codes mathematically, whose parity-check matrices are subject to some particular constraints.

Given a parity-check matrix $H_{m\times n}$ that takes elements in GF(2), a linear binary block code can be expressed as a set of vector solutions $\boldsymbol{x}$, which agree to the parity-check equations defined by $H_{m\times n}$.

$$H_{m\times n}\boldsymbol{x}^{\boldsymbol{T}}=\mathbf{0}^T \tag{4-1}$$

Definition 4.1: The linear block code is called a binary regular low-density parity-check (LDPC) code if the parity-check matrix has d_v ones in each column and d_c ones in each row, where both parameters d_v and d_c can be chosen under the constraint $d_c > d_v$. The number of parity-check equations m is an integer and equal to $(nd_v)/d_c$, where n is the length of the block code. This individual block code is denoted by $A(n,\ d_v,\ d_c)$. All these individual codes forms an ensemble denoted by $C^n(d_v,\ d_c)$. □

In other words, every codeword bit of $A(n,\ d_v,\ d_c)$ participates in exactly d_v parity-check equations and every such equation exactly involves d_c codeword bits. The word "low-density" conveys the fact that the fraction of nonzero entries in the parity-check matrix $H_{m\times n}$ is smaller as compared to the "random" linear codes.

Note that m parity-check equations might not be completely independent (or rank $(H_{m\times n}) \leq m$) so that the actual rate R of a given code may be higher than the design rate r, defined as follows:

$$r = \frac{n-m}{n} = 1 - \frac{d_v}{d_c} \tag{4-2}$$

The parity-check code in the Example 4.2 is a typical regular LDPC code $A(20, 3, 4)$ of design rate 1/4.

Usually, people do not focus on the particular regular LDPC codes individually, but rather analyze the performance of the codes *ensemble*. One way of constructing an *ensemble* of LDPC codes of length n would be to take the set of all parity-check matrices into account, which must fulfill the constraints of row, column and number of parity-check equations as defined above for a pair of specified parameters (d_v, d_c). However, it may be more convenient to characterize the *ensemble* of regular LDPC codes via bipartite graphs. A bipartite graph is uniquely in relation to a parity-check matrix of a regular LDPC code. This idea was originally suggested by Tanner [2][14] and extended by Forney [15] and Luby [17]. The following typical example is applied to show how this new method can effectively represent the individual regular LDPC code.

Example 4.3: Consider an $A(12, 3, 6)$ regular LDPC code illustrated by a bipartite graph shown in Fig. 4.3, in which there has two sets of nodes. One set of nodes are called variable nodes related to the binary digits of the codeword, another set of nodes are called check nodes corresponding to the parity-check constraints that define the whole code. Therefore, there are 12 variable nodes and 6 check nodes for the bipartite in the Fig. 4.3. The numbers 3 and 6 are called the degrees of the check and variable node in the bipartite, respectively. The edge, defined as the connection between variable and check node, exists if the variable is involved in the parity-check equation. Obviously, the number of the edges is identical to that of the nonzero elements in the parity-check matrix of the regular LDPC code.

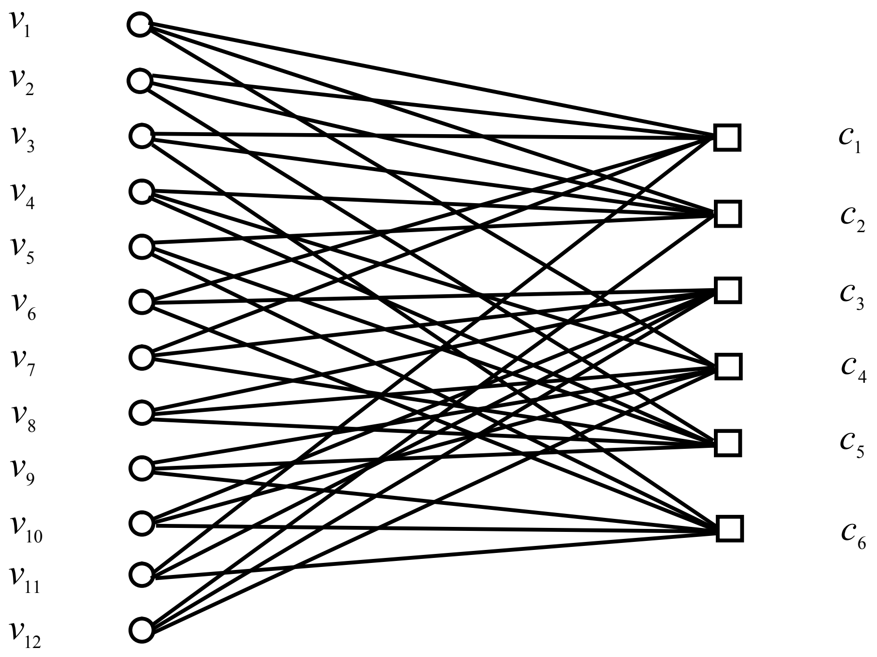

Fig. 4.3 An A(12, 3, 6)-regular LDPC code of length 12 and design rate-1/2 by bipartite. There are 12 variables and 6 check nodes. For each check node c_i (i=1, 2, …, 6) the sum over all the connected variable nodes is equal to zero.

On the other hand, one can easily derive the parity-check matrix of this regular LDPC code based on the relationships of the variable and check nodes characterized by the above bipartite. The equivalent matrix is

$$
H_{6\times 12} = \begin{array}{cccccccccccc c}
v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 & v_9 & v_{10} & v_{11} & v_{12} & \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & c_1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & c_2 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & c_3 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & c_4 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & c_5 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & c_6
\end{array} \qquad (4\text{-}3)
$$

Thus, the bipartite and parity-check matrix are uniquely corresponding with each other for a same regular LDPC code. Similarly, the bipartite and parity-check matrix of the regular LDPC code in the Example 4.2 can be also easily obtained.

4.2.2 Encoding of Regular LDPC Codes

In this section, we will develop an algorithm for efficient constructing of low-density parity-check codes. Given a parity-check matrix $\mathrm{H}_{m\times n}$ of a specified LDPC code over GF(2), the task of the encoder is to generate the n-tuples vector solutions $\boldsymbol{x}$ (n-tuples GF(2)) of the parity-check matrix $\mathrm{H}_{m\times n}$ that satisfy the condition defined by (4-1).

First, a general method is presented to encode the LDPC code by using the Gaussian elimination. Second, a more effective encoding approach with lower complexity is suggested by equivalently changing the parity-check matrix into an approximate lower triangular form, which is based on the inherit merits of sparseness of the parity-check matrix of the LDPC codes. Meanwhile, the approximate complexities for both encoding approaches are also compared.

Without loss of generality, the m row vectors of a parity-check matrix $\mathrm{H}_{m\times n}$ might not all be independent for arbitrary LDPC code. Hence, we have to eliminate the redundant rows from the original parity-check matrix and form a new matrix $\overline{H}_{m'\times n}$ whose m' ($m'\leq m$) row vectors are now completely independent.

Approach I: The Gaussian elimination can transform the initial sparse parity-check matrix $\overline{H}_{m'\times n}$ with rank m' into an equivalent lower triangular form $\tilde{H}_{m'\times n}$ formulated by

(4-4), where $h_{i,j} \in$GF(2) is the element of the *l*-th row and *j*-th column in the $\tilde{H}_{m' \times n}$. In particular, $h_{l,j+(n-m')}$=0 when $j > l$, and $h_{l,j+(n-m')}$=1 when $j = l$. Note that the resulting matrix will no longer be sparse. This is a preprocessing before the following actual encoding.

$$\tilde{H}_{m' \times n} = \begin{array}{c} \longleftarrow n-m' \longrightarrow \quad \longleftarrow \quad m' \quad \longrightarrow \\ \left(\begin{array}{cccccccc} \tilde{h}_{1,1} & \cdots & \tilde{h}_{1,(n-m')} & 1 & 0 & \cdots & \cdots & 0 \\ \vdots & & & \ddots & 1 & \ddots & & \vdots \\ \vdots & & & & \ddots & \ddots & \mathbf{0} & \vdots \\ \vdots & & & & & 1 & \ddots & \vdots \\ \vdots & & & & & \ddots & 1 & 0 \\ \tilde{h}_{m',1} & \cdots & \cdots & \cdots & \cdots & \cdots & \tilde{h}_{m',(n-1)} & 1 \end{array} \right) \end{array} \updownarrow m' \tag{4-4}$$

Then we split the vector $\boldsymbol{x}$ into a systematic part $\boldsymbol{s}$ and a parity-check part $\boldsymbol{p}$ so that $\boldsymbol{x}$=(s, p), where $\boldsymbol{s}$=(s_1, s_2,…, $s_{n-m'}$) and $\boldsymbol{p}$=(p_1, p_2,… $p_{m'}$) are $(n-m')$ and m'-tuples over GF(2), respectively. Consequently, construct a systematic encode as follows: i) Fill s with $(n-m')$ arbitrarily designed information bits. ii) Determine the m' parity-check bits in p using back-substitution. More precisely, we initially have

$$p_1 = \sum_{j=1}^{n-m'} \tilde{h}_{1,j}\, s_j \tag{4-5a}$$

and p_l (l=2, 3, …, m') can be computed iteratively using the following equation

$$p_l = \sum_{j=1}^{n-m'} \tilde{h}_{l,j}\, s_j + \sum_{j=1}^{l-1} \tilde{h}_{l,j+(n-m')}\, p_j \tag{4-5b}$$

Now, let us investigate the encoding complexity of this scheme. It can be easily obtained that bringing the sparse parity-check matrix $H_{m' \times n}$ into the desired form requires $\boldsymbol{O}(n^3)$ operations of preprocessing, and the actual encoding requires another $\boldsymbol{O}(n^3)$ operations on the non-sparse triangular matrix. Here the operations mean the "+" and "×" over the GF(2). This is a generic method that can apply to any parity-check code and hence the encoding complexity is very high.

One may think about a question: whether the encoding can be accomplished effectively while the computational complexity is constrained by $\boldsymbol{O}(n)$? Here, we introduce another encoding scheme with lower complexity that was first proposed by Richardson [7][8].

Approach II: We still consider the sparse parity-check matrix $\overline{H}_{m' \times n}$ with rank m' of the LDPC code in the first approach. Rather than the Gaussian elimination, here by performing the row and column permutations only, we can bring the matrix $\overline{H}_{m' \times n}$ into a so-called approximate lower triangular form $\hat{H}_{m' \times n}$ as represented by (4-6). Assume that the $\overline{H}_{m' \times n}$ can be expressed in terms its constituent submatrices as

$$\hat{H}_{m'\times n}=\begin{array}{ccc|c} \xleftrightarrow{\;n-m'\;} & \xleftrightarrow{\;g\;} & \xleftrightarrow{\;m'-g\;} & \\ \hline \boldsymbol{A} & \boldsymbol{B} & \begin{matrix} 1 & & \mathbf{0} \\ & 1 & \\ \boldsymbol{T} & & 1 \end{matrix} & \updownarrow m'-g \\ \hline \boldsymbol{C} & \boldsymbol{D} & \boldsymbol{E} & \updownarrow g \end{array} \qquad (4\text{-}6)$$

where A is a $(m'-g)\times(n-m')$, B is $(m'-g)\times g$, T is $(m'-g)\times(m'-g)$ lower triangular matrix, C is $g\times(n-m')$, D is $g\times g$ and E is $g\times(m'-g)$. Each of these matrices is sparse since it contains at most $\boldsymbol{O}(n)$ nonzero elements. The choice of the parameter g will be explained later. Note that all matrices are sparse and T is a matrix with ones along the diagonal. Multiplying $\hat{H}_{m'\times n}$ from the left by an $m'\times m'$ square matrix as below:

$$\begin{bmatrix} I_{\mathrm{u}} & 0 \\ -ET^{1} & \mathrm{I}_d \end{bmatrix} \qquad (4\text{-}7)$$

where I_u and I_d are the $(m'-g)\times(m'-g)$ and $g\times g$ identity matrices, respectively. Hence, the following equivalent matrix is resulted in

$$\begin{bmatrix} A & B & T \\ -ET^{1}A+C & -ET^{1}B+D & 0 \end{bmatrix} \tag{4-8}$$

Let the vector $\boldsymbol{x}$=(s, p_1, p_2) where s=(s_1, s_2,..., $s_{n-m'}$) is an ($n-m'$)-tuple over GF(2) and denotes the systematic part of a codeword, g-tuple p_1 and ($m'-g$)-tuple p_2 represent the parity parts of the codeword over GF(2), respectively. Hence, we have following two equations equivalently to satisfy the check relationship of (4.1) if the original parity check matrix $\hat{H}_{m'\times n}$ is replaced by (12.8), i.e.,

$$As^T + B p_1^T + Tp_2^T = \mathbf{0}^T \tag{4-9a}$$

$$(-ET^{-1}A+C)s^T + (-ET^{-1}B+D)p_1^T = \mathbf{0}^T \tag{4-9b}$$

For the sake of convenience, let the $g\times g$ matrix $-ET^{-1}B+D$ be defined as U and assumed to be invertible based on the above construction. Then, with the ($n-m'$) desired information bits in s^T, we can conclude from (4-9b) as follows

$$p_1^T = -U^{-1}(-ET^{-1}A+C)s^T \tag{4-10a}$$

and p_2^T is easily obtained from (4-9b) with the known s^T and p_1^T

$$p_2^T = -T^{-1}(As^T + B p_1^T) \tag{4-10b}$$

Hence, the whole vector (or codeword) $\boldsymbol{x}$=(s, p_1, p_2) is generated.

4.3. Complexity Analysis for LDPC Encoding

The procedure for the determination of p_1 is listed by the Table 4-1 that shows an efficient computation by several smaller steps.

Step	Operations	Comments	Complexity
1	As^T	Multiplication by a sparse matrix	$O(n)$
2	$T^{-1}As^T$	$T^{-1}[As^T]=y^T \Leftrightarrow [As^T]=Ty^T$	$O(n)$
3	$-E[T^{-1}As^T]$	Multiplication by a sparse matrix	$O(n)$
4	Cs^T	Multiplication by a sparse matrix	$O(n)$
5	$[-ET^{-1}As^T]+Cs^T$	Addition	$O(n)$
6	$-U^{-1}[-ET^{-1}As^T+Cs^T]$	Multiplication by a dense g×g matrix	$O(g^2)$

Table 4.1 The step by step computation of $p_1^T=-U^{-1}[-ET^{-1}As^T+Cs^T]$

The complexity of operation As^T is $O(n)$ since A is a sparse matrix. Let $y^T=T^{-1}[As^T]$ and this matrix operation is changed into an equivalent form as $As^T=Ty^T$, which can also be completed in $O(n)$ by back-substitution because T is a sparse and lower triangular matrix as formulated in (12.6). Meanwhile, the complexities of matrix operations in step 3, 4 and 5 are all $O(n)$ by multiplying or adding respective sparse matrix. Note that U is usually a $g \times g$ dense matrix, and the complexity of the last operation is thus $O(g^2)$.

Step	Operations	Comments	Complexity
1	As^T	Multiplication by a sparse matrix	$O(n)$
2	Bp_1^T	Multiplication by a sparse matrix	$O(n)$
3	$[As^T]+[Bp_1^T]$	Addition	$O(n)$
4	$-T^1[As^T+Bp_1^T]$	$-T^1[As^T+Bp_1^T]=y^T\Leftrightarrow-[As^T+Bp_1^T]=Ty^T$	$O(n)$

Table 4.2 The step by step computation of $p_2^T=-T^{\text{-}1}[As^T+Bp_1^T]$.

Similarly, the operation for determination of p_2 can be accomplished by a step by step manner in complexity $O(n)$ as shown in Table 4.2.

The entire complexity of the operations for determination of the codeword $\boldsymbol{x}=(s, p_1, p_2)$ is $O(n+g^2)$. Therefore, to reduce the complexity of the operation, we must set the value g as small as possible in the matrix divisions by (4-6).

In order to give an overall clear picture of above approach, we will demonstrate the following example, which adopts the same regular LDPC code $A(12, 3, 6)$ as the last example, to illustrate the encoding procedures in detail.

4.3.1 A Typical Example for LDPC Encoding Procedures

Example 4.4: If we simply reorder the columns of the parity-check matrix $\mathbf{H}_{6\times12}$, denoted by (4-3), in terms of its original order, the reordered columns are numbered by 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 8, 9 from left to right. Then, the new parity-check matrix becomes an approximate lower triangular form with $g=2$ as

$$\left[\begin{array}{c|c|c} A & B & T \\ \hline C & D & E \end{array}\right] = \left(\begin{array}{cccccc|cc|cccc} 1&1&1&0&0&1&1&0&1&0&0&0 \\ 1&1&1&1&1&0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&1&1&1&1&1&0 \\ 1&0&0&1&0&0&0&1&0&1&1&1 \\ \hline 0&1&0&1&1&0&1&0&0&0&1&1 \\ 0&0&1&0&1&1&0&1&1&0&0&1 \end{array}\right) \tag{4-11}$$

Unfortunately, $U = -ET^{-1}B + D = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ is not invertible. Thus, to avoid this difficulty, we have to change the columns of the original parity-check matrix in another order, and final order of the columns is arranged as 1, 2, 3, 4, 10, 6, 7, 5, 11, 12, 8, 9, and the equivalent parity-check matrix $\mathbf{H}$ is

$$\mathbf{H}_{6\times 12} = \left[\begin{array}{c|c|c} A & B & T \\ \hline C & D & E \end{array}\right] = \left(\begin{array}{cccccc|cc|cccc} 1&1&1&0&0&1&1&0&1&0&0&0 \\ 1&1&1&1&0&0&0&1&0&1&0&0 \\ 0&0&0&0&1&1&1&0&1&1&1&0 \\ 1&0&0&1&1&0&0&0&0&1&1&1 \\ \hline 0&1&0&1&0&0&1&1&0&0&1&1 \\ 0&0&1&0&1&1&0&1&1&0&0&1 \end{array}\right) \tag{4-12}$$

According to the submatrices defined by (4-6), it is easily to calculate the two submatrices of the above equivalent parity-check matrix $\mathbf{H}$

$$U = -ET^{-1}B + D = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{4-13a}$$

$$-ET^{-1}A+C=\begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \tag{4-13b}$$

Apparently, U is an invertible matrix, and thus p_1 can be finalized directly by applying (4-13) and (4-10a) if s is specified. For instance, given s be (1, 0, 0, 1, 0, 0), and we have

$$p_1^T=-U^{-1}(-ET^{-1}A+C)s^T=(1, 1)^T \tag{4-14a}$$

$$p_2^T=-T^{-1}(As^T+Bp_1^T)=(0, 1, 0, 1)^T \tag{4-14b}$$

Therefore, the entire codeword is equal to

$$\boldsymbol{x}=(s,\ p_1,\ p_2)=(1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1) \tag{4-15}$$

The correctness of above codeword is verified by satisfying the check equations $\mathbf{H}_{6\times12}\boldsymbol{x}^T=0$ as required, where $\mathbf{H}_{6\times12}$ is given by (4-12).