IEICE TRANS. FUNDAMENTALS, VOL.E89–A, NO.10 OCTOBER 2006

PAPER Special Section on Information Theory and Its Applications

Encoding LDPC Codes Using the Triangular Factorization

Yuichi KAJI[†a)], ***Member***

SUMMARY An algorithm for encoding low-density parity check (LDPC) codes is investigated. The algorithm computes parity check symbols by solving a set of sparse equations, and the triangular factorization is employed to solve the equations efficiently. It is shown analytically and experimentally that the proposed algorithm is more efficient than the Richardson's encoding algorithm if the code has a small gap.

key words: *LDPC codes, encoding algorithm, triangular factorization, sparse matrix*

1. Introduction

The encoding can be a computational bottleneck of a communication system which uses a *low density parity check* (*LDPC*) code. It has been considered for long years that encoding is "easier" operation than decoding. Indeed, for classic linear block codes, encoding is possible in the quadratic-order in the code length, while the maximum likelihood decoding is, for example, $\mathcal{NP}$-complete in general. Consequently major efforts of coding theorists have been aimed to reduce the decoding complexity, and little attention have been paid to reduce the encoding complexity. The situation is, however, quite different for LDPC codes. Thanks to the sparse structure of LDPC codes, a belief propagation algorithm with relatively small number of iterations can estimate the transmitted codeword with near-optimum precision. In other words, we have sufficiently powerful and efficient, say almost linear-order, decoding algorithms for LDPC codes. Consequently, for LDPC codes, encoding becomes "more complicated" operation than decoding. To make the ability of LDPC codes fully effective, we usually need to use a very long code, say several thousand bits for example. Therefore the gap of the encoding and decoding complexities may cause serious problems in the implementation of communication systems.

One solution to this issue is to investigate a subclass of LDPC codes which are easy to encode and also show good performance. For example, the *irregular repeated accumulate code* (*IRA code*) [3] with optimized profile show very good performance. The IRA code can be regarded as an LDPC code whose parity check matrix contains a triangular submatrix, and hence the encoding is possible in the linear order to the code length by utilizing the back substitution technique (or by a simple accumulation circuit). Another example in this approach is a *quasi-cyclic LDPC code*. The quasi-cyclic LDPC codes are a class of LDPC codes which have favorable structure for efficient encoding and decoding. It is known that appropriately constructed quasi-cyclic LDPC codes show very good error correcting performance. See [1], [2] and [7] for recent results on quasi-cyclic LDPC codes. It is promising and significant to investigate these classes of LDPC codes with certain structures, but we need to remind that this kind of structure may restrict the performance and characteristics of codes. To promote studies for wider classes of LDPC codes, it is also important to develop encoding algorithms which can be used for arbitrary LDPC codes. A widely known results in this philosophy is the algorithm proposed by Richardson and Urbanke [8]. The Richardson's algorithm fully makes use of the sparseness of the parity check matrices of LDPC codes, and achieve much smaller complexity than the conventional encoding algorithms. The complexity of the Richarson's algorithm is $O(g^2 + N)$ where g is a parameter called the *gap* of the code, and N is the code length. Asymptotically, the gap g is proportional to the code length N in general, and therefore the complexity of the Richardson's algorithm is still quadratic-order to N. However, it is also shown in [8] that g is expected to be very small compared to N. With respect to practical and quantitative complexity measure, instead of asymptotic order-notation, we can say that the Richardson's algorithm is much more efficient than naive encoding algorithms.

In this paper, we investigate a new encoding algorithm for general LDPC codes. It has been widely recognized that the encoding of a linear block code can be regarded as solving a system of equations. This fact however did not attract researchers because solving a system of equations is another difficult problem. In this paper, we consider to use the *triangular factorization*, also known as the *LU-factorization*, to solve a system of sparse equations. Similar to the case of the Richardson's algorithm, we need to permute the row and column vectors of the parity check matrix beforehand to reduce the complexity of the encoding algorithm. Unfortunately it seems quite difficult to find the optimum permutation which makes the algorithm most efficient, and therefore we investigate two sub-optimum permutations in this paper. To compare the complexity of encoding algorithms in detail, *the number of operations* which are needed in an encoding operation is evaluated for the proposed and the Richardson's algorithms. Intuitively, the number of operations is a mea-

Manuscript received January 25, 2006.
Manuscript revised April 12, 2006.
Final manuscript received June 6, 2006.

[†]The author is with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630-0101 Japan.
a) E-mail: kaji@is.naist.jp

DOI: 10.1093/ietfec/e89–a.10.2510

sure of the number of logic gates in a hardware implemented encoder. It is shown that the proposed algorithm consumes small number of operations if an LDPC code has a small gap. To justify this result, we also present some results of computer simulation.

2. Preliminary

2.1 Two-Stage Encoding

Let C be a *low-density parity check matrix* (*LDPC*) code with code length N and dimension K. We write $P = N - K$ to represent the number of *parity check symbols* of C. We consider, for simplicity, the case that C is a binary code. The extension of our result to the non-binary case is straightforward. It is assumed that operations and computations over matrices and vectors are over modulo 2, and we will omit "mod2" in equations if it is clear from a context. Let $\Sigma = \{0, 1\}$ be the set of binary symbols. For a nonnegative integer n, we write Σ^n for the set of vectors of length n over Σ. The concatenation of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is written as $\boldsymbol{x} \cdot \boldsymbol{y}$, and the transposition of a vector $\boldsymbol{x}$ is written as $\boldsymbol{x}^T$.

Let H be a parity check matrix of the code C, and consider to represent H as $H = [H_1 H_2]$ where H_1 and H_2 are $P \times K$ and $P \times P$ submatrices, respectively. Assume that H_2 is nonsingular. This assumption is equivalent to assuming that C is systematic and the first K symbols of a codeword of C are *information symbols*. For two vectors $\boldsymbol{s} \in \Sigma^K$ and $\boldsymbol{p} \in \Sigma^P$, their concatenation $\boldsymbol{s} \cdot \boldsymbol{p}$ is a correct codeword of C if and only if $H(\boldsymbol{s} \cdot \boldsymbol{p})^T = 0 \bmod 2$. Since $H = [H_1 H_2]$, this equation is equivalent to $H_1 \boldsymbol{s}^T = H_2 \boldsymbol{p}^T \bmod 2$. Therefore, the *encoding* can be regarded as a procedure to find the vector $\boldsymbol{p} \in \Sigma^P$ which satisfies $H_1 \boldsymbol{s}^T = H_2 \boldsymbol{p}^T \bmod 2$ for a given vector $\boldsymbol{s} \in \Sigma^K$ of information symbols. According to this observation, we can consider the following *two-stage encoding* procedure.

Stage 1: Compute $\boldsymbol{u}^T = H_1 \boldsymbol{s}^T$.
Stage 2: Solve $H_2 \boldsymbol{p}^T = \boldsymbol{u}^T \bmod 2$ with respect to $\boldsymbol{p}$.

If C is an LDPC code, then H, H_1 and H_2 are all sparse (low-density). Therefore, by using the algorithm for the sparse-matrix multiplication, the computation in the stage 1 above is possible in a linear order (precisely, the complexity is proportional to the number of nonzero components in H_1). On the other hand, the complexity needed to execute the stage 2 is beyond linear order in general. For example, we may consider to solve the equation by computing H_2^{-1} first and then by computing $\boldsymbol{p}^T = H_2^{-1} \boldsymbol{u}^T$. The complexity of this procedure will be in the order of P^2 in general since H_2^{-1} is no more sparse even if H_2 is sparse. If there are efficient means for solving $H_2 \boldsymbol{p}^T = \boldsymbol{u}^T \bmod 2$, then we can realize an efficient encoding algorithm for LDPC codes. In this study, we consider to use the *triangular factorization* for solving the equation efficiently.

2.2 Triangular Factorization

The *triangular factorization*, which is sometimes refereed as the *LU-factorization* in some textbooks, has been long studied in a branch of mathematics [9]. The existing studies are mostly for continuous systems, but we can do similar discussion for the binary case.

Definition 2.1: Let A be an $n \times n$ binary matrix. The *k-th order leading principle* of A, where $1 \le k \le n$, is the $k \times k$ submatrix of A which consists of the first k rows and the first k columns of A. The matrix A is said to satisfy the *LP-condition* if its k-th order leading principle is nonsingular for all k with $1 \le k \le n$. □

It follows from the definition that if A satisfies the LP-condition, then A is nonsingular. The converse does not hold in general, but if A is nonsingular, then we can permute rows and columns of A in such a way that the matrix satisfy the LP-condition.

Lemma 2.1: Let A be an $n \times n$ matrix and assume that A is nonsingular (the rank of A is n). Then there exist permutations of row and column vectors of A which makes A to satisfy the LP-condition. □

The proof is presented in the Appendix.

Lemma 2.2: If A satisfies the LP-condition, then there are a binary lower-triangular matrix L and a binary upper-triangular matrix U satisfying $A = LU \bmod 2$. Furthermore, L and U are unique, and diagonal components of L and U are all one.

Proof. The proof is by induction on n. For $n = 1$, A satisfies the LP-condition if and only if $A = (1)$. In this case, $L = U = (1)$. The uniqueness of L and U is obvious, and the diagonal components of L and U are all one. For the inductive step, assume that A is an $n \times n$ matrix. The matrix A can be written as

$$A = \begin{pmatrix} D & \boldsymbol{x} \\ \boldsymbol{y} & z \end{pmatrix},$$

where D, $\boldsymbol{x}$ and $\boldsymbol{y}$ are $(n-1) \times (n-1)$, $(n-1) \times 1$ and $1 \times (n-1)$ binary matrices, respectively, and z is either zero or one. Since A is assumed to satisfy the LP-condition, D also satisfies the LP-condition. Therefore, there are L_D and U_D which satisfies $D = L_D U_D$ by the inductive hypothesis. Note that the inductive hypothesis also implies that the diagonal components of L_D and U_D are all one, and therefore L_D and U_D have inverse matrices L_D^{-1} and U_D^{-1}, respectively. Now define

$$L = \begin{pmatrix} L_D & 0 \\ \boldsymbol{y} U_D^{-1} & \boldsymbol{y} U_D^{-1} L_D^{-1} \boldsymbol{x} + z \end{pmatrix},$$

$$U = \begin{pmatrix} U_D & L_D^{-1} \boldsymbol{x} \\ 0 & 1 \end{pmatrix}.$$

We can easily see that L and U are triangular, $A = LU \bmod 2$ and all components in L and U are determined uniquely. Remind again that the diagonal components of L_D and U_D are all one. This obviously implies that the diagonal components of U are all one. Assume that the (n, n) component of L is zero. In this case, the n-th column of L is a

zero vector, and the rank of L must be $n-1$ or less. Because $A = LU \bmod 2$, this means that the rank of A must be $n-1$ or less, which cannot occur because A satisfies the LP-condition and therefore nonsingular (i.e. has rank n). Consequently, the (n,n) component of L is one and the diagonal components of L are all one. This completes the proof. □

If $A = LU$, then we say that L and U are *factors* of A. Consider to solve an equation $A\boldsymbol{x} = \boldsymbol{b} \bmod 2$, and assume that L and U are factors of A. Then $A = LU$ and $A\boldsymbol{x} = \boldsymbol{b}$ is transformed into

$$\boldsymbol{x} = A^{-1}\boldsymbol{b} = U^{-1}L^{-1}\boldsymbol{b}.$$

Since L is triangular, we can use the *back substitution* technique to compute the vector $\boldsymbol{y} = L^{-1}\boldsymbol{b}$. Using another back substitution (forward substitution) to this $\boldsymbol{y}$, we can compute $\boldsymbol{x} = U^{-1}\boldsymbol{y}$. The complexity for the two back substitutions depends on the density of L and U. Indeed, the complexity for the back substitution is small if L and U have small number of nonzero components (i.e. the matrices are sparse). Unfortunately, the relation among the density of A, L and U is not known clearly. However, in most cases, L and U are expected to be sparse if A is sparse, even though they are less sparse than A.

Example 2.1: Consider the following matrix A.

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

This matrix can be represented as $A = LU$ where

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & \underline{1} & 0 \\ 0 & 1 & 1 & 1 & \underline{1} & 1 \end{pmatrix},$$

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & \underline{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Readers may notice that the matrices L and U have similar zero-one patterns to A. The lower-triangular part of A is inherited to L, and the upper-triangular part of A is inherited to U. The underlined components in L and U indicates that they are different from their counterparts in A. □

We remark that the order of the rows and columns in A is important for the sparseness of L and U. Permuting row and/or column vectors in A (and corresponding components in $\boldsymbol{x}$ and $\boldsymbol{b}$) does not change the solution of the equation $A\boldsymbol{x} = \boldsymbol{b}$, but changes the density of L and U in general. For example, let A' be the matrix obtained from A in Example 2.1 by moving the fifth row vector of A to the top row position;

$$A' = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix},$$

then A' is represented as $A' = L'U'$ where

$$L' = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & \underline{1} & 0 & 0 & 0 & 0 \\ 0 & 1 & \underline{1} & 0 & 0 & 0 \\ 0 & 0 & 1 & \underline{1} & 0 & 0 \\ 1 & \underline{1} & 0 & 1 & 1 & 0 \\ 0 & 1 & \underline{0} & \underline{0} & 0 & 1 \end{pmatrix},$$

$$U' = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & \underline{1} & \underline{1} & \underline{1} & 0 & 0 \\ 0 & 0 & \underline{1} & \underline{1} & 1 & 1 \\ 0 & 0 & 0 & \underline{1} & \underline{0} & \underline{0} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Compare L' and U' with L and U in Example 2.1, and readers may notice that more zero components in A' become nonzero in L' and U'. This suggests that the back substitutions for L' and U' will consume more complexity than those for L and U in Example 2.1. To solve the equation $A\boldsymbol{x} = \boldsymbol{b}$ efficiently, we should permute the row and column vectors of A so that its factors are as sparse as possible.

3. Proposed Algorithm

Recall the two-stage encoding procedure which was considered in 2.1. As we have seen, the stage 1 can be executed efficiently, while the stage 2 for solving $H_2\boldsymbol{p}^T = \boldsymbol{u}^T$ consumes large complexity in general. Now consider to factor H_2 by the triangular factorization, then we can solve the equation by two back substitutions. The procedure for encoding is sketched as follows.

Algorithm 3.1: We need to execute the following pre-computation step only once before we actually perform encoding.

[pre-computation] Permute row vectors and column vectors of the parity check matrix H so that the H_2-part of H satisfies the LP-condition, and the triangular matrices L and U with $H_2 = LU \bmod 2$ are sparse. We will discuss later how to obtain good permutations.

[encoding] Given an information vector $\boldsymbol{s} \in \Sigma^K$, the parity check vector $\boldsymbol{p} \in \Sigma^P$ for $\boldsymbol{s}$ is computed as follows.

Stage 1: Compute $\boldsymbol{u}^T = H_1\boldsymbol{s}^T$.

Stage 2: Solve $H_2\boldsymbol{p}^T = \boldsymbol{u}^T$ as follows;

1. compute $\boldsymbol{v}^T = L^{-1}\boldsymbol{u}^T$ by a back substitution for L, and then

2. compute $\boldsymbol{p}^T = U^{-1}\boldsymbol{v}^T$ by a back substitution for U.

We call this algorithm the *TSTF* (*Two-Stage encoding with the Triangular Factorization*) in this paper. □

Regarding the above procedure, there are two issues which we need to discuss. The first issue is the LP-condition of the submatrix H_2. For a randomly constructed LDPC code, we cannot assume that the submatrix H_2 satisfies the LP-condition. If H_2 does not satisfy the LP-condition, then the above procedure is not available because we cannot factor H_2 to two triangular matrices. To avoid this issue, we need to permute row and column vectors of the parity check matrix. We can easily extend Lemma 2.1 to non-square matrices, and obtain the following lemma.

Lemma 3.1: If H is a $P \times N$ matrix with rank P, where $P < N$, then there exist permutations of row and column vectors of H which transforms H to $H' = [H'_1 H'_2]$ in such a way that H'_2 satisfies the LP-condition. □

The proof is omitted since it is essentially the same as the proof of Lemma 2.1. We remark that a permutation of row vectors of H does not change the code itself. In general, a permutation of column vectors of H changes the symbol order in a codeword, but the change does not affect code profiles such as the minimum distance, weight distributions, girth and so on. Thus, the permutation is not essential for error correction capability of the code. Thus, we can say that the above procedure can be used for arbitrary LDPC codes.

The second issue concerning the above procedure is the complexity. If back substitutions consume more complexity than simply computing $H_2^{-1}\boldsymbol{u}^T$, then the above procedure is useless. To evaluate the complexity of the proposed encoding algorithm, we first introduce a quantitative and detailed measure for the complexity. The order notation which is widely used in the algorithm theory is a good measure to discuss asymptotic behavior of algorithms, but it is too coarse for comparing algorithms for a fixed size instances. In this paper, we will discuss the complexity of encoding algorithms by means of *the number of operations*. Let $|M|$ denote the number of nonzero components in a matrix M, and let $\boldsymbol{x}$ and $\boldsymbol{y}$ be vectors with an appropriate length. We define

- *The number of operations* necessary for computing $M\boldsymbol{x}$ is $|M|$.
- *The number of operations* necessary for computing $T^{-1}\boldsymbol{x}$ is $|T|$ where T is a triangular matrix.
- *The number of operations* necessary for computing $\boldsymbol{x} + \boldsymbol{y}$ is the length of $\boldsymbol{x}$ ($\boldsymbol{y}$).

The above definition is derived by observing a typical algorithm for each computation. The number can be used to estimate the number of logic gates in a hardware-implemented encoder. Using the above definition, the number of operations needed in the TSTF algorithm, denoted as γ_{TSTF}, is represented as $\gamma_{\mathrm{TSTF}} = |H_1| + |L| + |U|$. To reduce the number of operations, we need to reduce $|H_1|$ or $|L|$ or $|U|$. we make use of the permutation in the pre-computation step of the algorithm to realize this requirement.

4. Permutations of Vectors of a Matrix

To make the TSTF algorithm efficient, we need to find a good permutation of the check matrix so that the factors L and U are as sparse as possible. To the author's knowledge, unfortunately, there is no efficient method for finding the optimum permutation which makes $|L|$ and $|U|$ smallest. In this section, we investigate two sub-optimum strategy to make $|L|$ and $|U|$ small.

4.1 Permutations based on Approximate Triangular Matrices

4.1.1 Approximate Triangular Matrices

A parity check matrix in an *approximate triangular* form is investigated by Richardson [8] to reduce the complexity of his encoding algorithm. It is shown in this section that a parity check matrix in an approximate triangular form is also useful for the TSTF algorithm.

A parity check matrix H is said to be an *approximate triangular matrix* (*ATM* for short) if

$$H = \begin{pmatrix} A & B & T \\ C & D & E \end{pmatrix}, \tag{1}$$

where T is a triangular matrix. The number of row vectors in the second row block (C, D and E) is called the *gap* of this matrix, and the second column block (B and D) is taken so that the number of column vectors in the block equals to the gap. In [8], some algorithms are investigated for transforming a parity check matrix into an ATM form by permuting row and column vectors in the matrix. Remark that the matrix obtained in this way has the same density as the original parity check matrix. It is also shown in [8] that the gap g is asymptotically proportional to the code length N but its ratio g/N will be quite small for most practical LDPC codes.

From the parity check matrix in an ATM form in (1), exchange the second and the third column blocks, and define

$$H_1 = \begin{pmatrix} A \\ C \end{pmatrix}, \quad H_2 = \begin{pmatrix} T & B \\ E & D \end{pmatrix}. \tag{2}$$

Remark that the k-th order leading principle of H_2 with $1 \le k \le P - g$ is obviously nonsingural because the left-upper submatrix of H_2 is a triangular matrix T with size $P - g$. The k-th order leading principle with $P - g < k \le P$ may not be nonsingular, but we can permute rows and columns in the second row block and the second column block of H_2, respectively, to make the all leading principles nonsingular (see Lemma 2.1). Consequently, we can always make H_2 satisfy the LP-condition. Now Consider to apply the triangular factorization to H_2. The factors L and U of H_2 must be of the form

$$L = \begin{pmatrix} T & 0 \\ E & D_L \end{pmatrix}, \quad U = \begin{pmatrix} I & B_U \\ 0 & D_U \end{pmatrix}, \tag{3}$$

where I is an identity matrix, D_L, B_U and D_U are matrices satisfying $B = TB_U$ and $D = EB_U + D_L D_U$. Remark that T and E are sparse if H is sparse, while D_L, B_U and D_U are not sparse in general. However, if the gap g is small, then D_L, B_U and D_U occupy small portion of the matrices L and U. In this case, L and U are "mostly" sparse and we can expect that L and U contain small number of nonzero components. We have seen in the previous section that the number of operations γ_{TSTF} is given as $|H_1| + |L| + |U|$. By using (2), (3) and the fact that $|I| = P - g$,

$$\begin{aligned}\gamma_{\mathrm{TSTF}} &= |H_1| + |L| + |U| \\ &= |A| + |C| + |T| + |E| + |D_L| \\ &\quad + P - g + |B_U| + |D_U|.\end{aligned}$$

To compare the above complexity with the complexity of the Richardson's algorithm, we evaluate the number of operations needed in the Richardson's algorithm. We omit the details of the algorithm but it consists of some steps of matrix operations. Table 1 shows how many operations are needed to execute each step in the Richardson's algorithm, where $\boldsymbol{p} = \boldsymbol{p}_1 \cdot \boldsymbol{p}_2$, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ have length g and $P - g$, respectively, and $\phi = -ET^{-1}B + D$ (see [8] for the details). The total number of operations needed for the Richardson's algorithm is

$$\begin{aligned}\gamma_{\mathrm{RU}} &= |A| + |T| + |E| + |C| + g + |\phi^{-1}| \\ &\quad + |B| + P - g + |T|.\end{aligned}$$

Consider the difference between γ_{TSTF} and γ_{RU}, and define

$$\begin{aligned}\Delta &= \gamma_{\mathrm{TSTF}} - \gamma_{\mathrm{RU}} \\ &= |D_L| + |B_U| + |D_U| - g - |\phi^{-1}| - |B| - |T| \qquad (4)\end{aligned}$$

If $\Delta < 0$, then the TSTF algorithm consumes smaller number of operations than the Richardson's algorithm.

We would like to evaluate Δ but it is difficult to estimate the number of nonzero components in the matrices in (4) precisely. Thus we approximate Δ under the following assumptions.

Assumption 1: A dense matrix has almost equal number of zeros and ones.

Assumption 2: Nonzero components in H distribute "uniformly" in the matrix H except the triangular zero-part of H.

From the assumption 1, we approximate $|\phi^{-1}| = g^2/2$ since ϕ^{-1} has g^2 components, and about half the components are considered to be nonzero. Similarly we approximate $|B_U| = g(P - g)/2$, though this approximation of B_U is "too conservative" as we will see later. According to the similar assumption, $|D_L|$ and $|D_U|$ are both approximated to be $g^2/4$, since the matrices are dense triangular; half the components above (or below) diagonal components are all zero, and the remaining half contains almost equal number of zeros and ones. As for the assumption-2, let ρ be the ratio of nonzero components in H, and consider that a submatrix of H which has m components contains $m\rho$ nonzero components. This assumption implies that $|B| = g(P - g)\rho$ and $|T| = (P - g)^2\rho/2$.

Substitute terms in (4) with the above approximated value, then

$$\Delta = -\frac{1}{2}((1 - \rho)g^2 - (P - 2)g + P^2\rho). \qquad (5)$$

From (5), we can see that Δ approaches to $-P^2\sigma/2$ as the gap g approaches to zero. This suggests that if the gap g is very small, then the TSTF algorithm is more efficient (with respect to the number of operations) than the Richardson's algorithm.

For the practical viewpoint, the size of gaps of LDPC codes are of great interest. Richardson gives theoretical discussion on the gap size [8], and showed that the gap size has strong relation to the erasure decoding. The conclusion in [8] is that; good codes have small gaps. There is good correspondence between Richardson's encoding algorithm and the message-passing erasure decoding algorithm. The gap size intuitively corresponds to the number of erasures which cannot be recovered by a message-passing erasure decoding algorithm. Thus, if the code is good enough and has sufficient ability for erasure decoding, then the gap is expected to be small. For the detailed discussion, see [8]. This result suggests that the proposed algorithm will be more efficient than Richardson's algorithm for wide range of "good" LDPC codes.

4.1.2 Improved Approximate Triangular Form

In the previous section, we assumed that dense submatrices in L and U are random matrices (the assumption 1). However, this assumption might be too conservative for some submatrices. Indeed, the density of the submatrix B_U is "controllable" to some extent, and we can manage so that $|B_U| \ll g(P - g)/2$. Making $|B_U|$ small reduces the number of operations of the TSTF algorithm. We consider to "improve" the matrix H_2 obtained in the previous section.

In the following, we consider to sweep out bad column vectors in H_2 to H_1. In other words, column vectors which belong to H_2 and make many nonzero components in U are exchanged with other column vectors in H_1. Assume that H has been transformed to an ATM form given in (2). The factors L and U are defined as (3). Let i be an integer with $1 \le i \le g$, and assume that we number columns in B from one to g. Consider to exchange the i-th column vector of B with a column vector in A. The corresponding column vectors in D and C are also exchanged. Let H_1', H_2' be matrices after this exchange, and L' and U' be factors of H_2', assuming that H_2' satisfies the LP-condition. The density of H_1 and H_1' will be almost the same because the exchanged column vectors are expected to have almost the same density. The matrices H_2 and H_2' have the same T-part and E-part, which means that L' and U' can be written as

$$L' = \begin{pmatrix} T & 0 \\ E & D_L' \end{pmatrix}, \quad U' = \begin{pmatrix} I & B_U' \\ 0 & D_U' \end{pmatrix}.$$

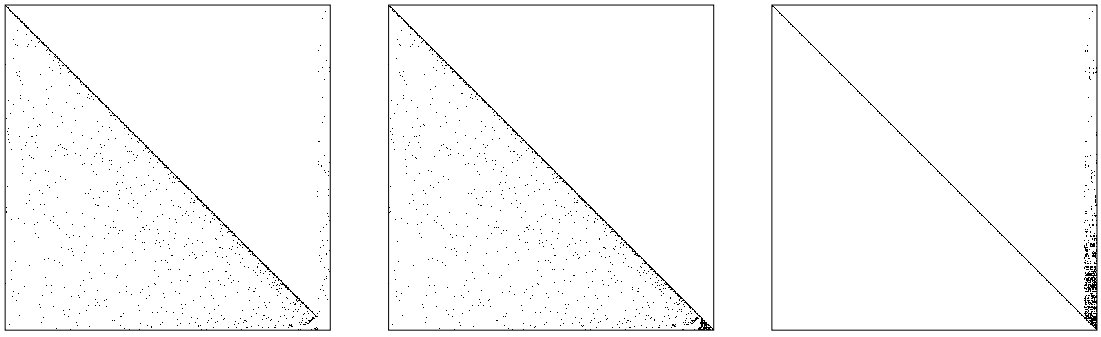
Table 1 The number of computations in the Richardson's algorithm.

computation of $\boldsymbol{p}_1^T = -\phi^{-1}(-ET^{-1}A + C)\boldsymbol{s}^T$

operation	# of operations
$A\boldsymbol{s}^T$	$\|A\|$
$T^{-1}[A\boldsymbol{s}^T]$	$\|T\|$
$-E[T^{-1}A\boldsymbol{s}^T]$	$\|E\|$
$C\boldsymbol{s}^T$	$\|C\|$
$[-ET^{-1}A\boldsymbol{s}^T] + [C\boldsymbol{s}^T]$	g
$-\phi^{-1}[-ET^{-1}A\boldsymbol{s}^T]$	$\|\phi^{-1}\|$

computation of $\boldsymbol{p}_2^T = T^{-1}(A\boldsymbol{s}^T + B\boldsymbol{p}_1^T)$

operation	# of operations
$A\boldsymbol{s}^T$	0 (already computed)
$B\boldsymbol{p}_1^T$	$\|B\|$
$[A\boldsymbol{s}^T] + [B\boldsymbol{p}_1^T]$	$P - g$
$T^{-1}[A\boldsymbol{s}^T + B\boldsymbol{p}_1^T]$	$\|T\|$



H_2 L U

Fig. 1 Factorization for the (1008, 504) Gallager code; ATM based permutation.

The matrix B'_U equals to B_U except the i-th column. The same property holds for D_L and D'_L, and also for D_U and D'_U. In other words, changing the i-th column of B and D changes the i-th column of B_U, D_L and D_U, but does not change other column vectors in L and U. Therefore, if the sum of nonzero components in the i-th columns of B'_U, D'_L and D'_U is smaller than that of B_U, D_L and D_U, then we will have $|H'_1| + |L'| + |U'| < |H_1| + |L| + |U|$. Based on the above discussion, we can examine if a column vector in B and D can be replaced by a better column vector in A and C. By "improving" the check matrix in this way, we can reduce the number of nonzero components in L and U. As we can see in the following example, $|B_U|$ can be much smaller than the approximated conservative value $g(P - g)/2$.

Example 4.1: Let C be the (1008, 504) Gallager code with column weight 3 [6]. Figure 1 shows H_2, L and U for this code. The gap of this matrix is 20, and the submatrix B_U has $504 - 20 = 484$ rows and 20 columns. If B_U is a random matrix, then the expected value of $|B_U|$ is $484 \times 20/2 = 4840$. In the case of the matrix U shown in Fig. 1, $|B_U| = 718$, much smaller than the expected value of 4840. The number of operations γ_{TSTF} and γ_{RU} for these matrices are 4399 and 5077, respectively. This means that, for this code, the TSTF algorithm is more efficient than the Richardson's algorithm, saving more than 600 operations. □

4.2 Permutations Based on a Greedy Strategy

Tewarson proposes a greedy algorithm which permutes row and column vectors of a given matrix so that its factors become sparse [9]. We first investigate a binary version of the Tewarson's algorithm, and consider to apply it to reduce the number of operations of the TSTF algorithm.

Meanwhile, consider that an $n \times n$ square matrix $A = (a_{i,j})$ is given, and we compute triangular matrices $L = (l_{i,j})$ and $U = (u_{i,j})$ satisfying $A = LU \bmod 2$. This equation implies that

$$a_{i,j} = \sum_{k=1}^{n} l_{i,k}u_{k,j} \bmod 2.$$

Remind that L and U are triangular matrices whose diagonal components are all one. Thus we have $l_{i,i} = u_{i,i} = 1$ for $1 \le i \le n$, $l_{i,j} = 0$ for $i < j$, and $u_{i,j} = 0$ for $i > j$. Therefore

$$a_{i,j} = \begin{cases} \displaystyle\sum_{k=1}^{i-1} l_{i,k}u_{k,j} + u_{i,j} \bmod 2 & (i < j), \\ \displaystyle\sum_{k=1}^{j-1} l_{i,k}u_{k,j} + l_{i,j} \bmod 2 & (i \ge j). \end{cases}$$

By using the equations, $l_{i,j}$ with $i \ge j$ and $u_{i,j}$ with $i < j$ are both written as

$$a_{i,j} + \sum_{k=1}^{\min(i,j)-1} l_{i,k}u_{k,j} \pmod 2. \tag{6}$$

We consider an iteration algorithm which transforms the matrix A. At the r-th iteration of the algorithm, we determine a column vector and a row vector which should be brought to the r-th column position and the r-th row position, respectively. Assume that we already have determined the first $r - 1$ row vectors and the first $r - 1$ column vectors of A. In this case, the first $r - 1$ column vectors of L and

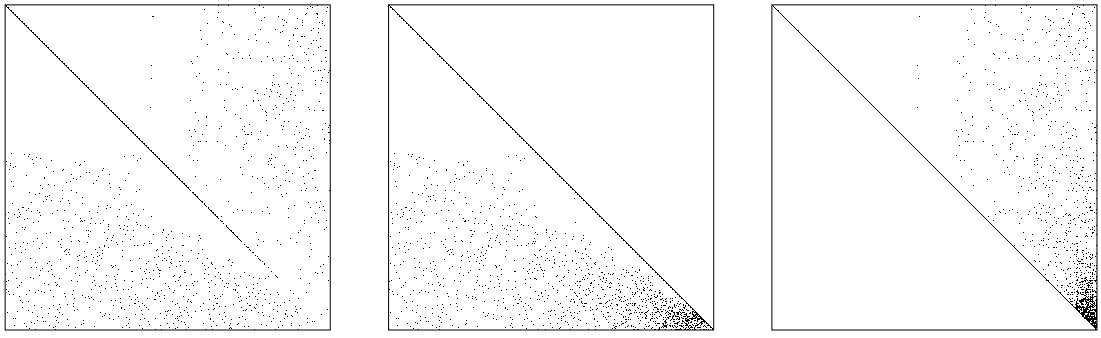


Fig. 2 Factorization for the (1008, 504) Gallager code; greedy permutation.

Table 2 The number of operations for several codes.

name	N	K	P	λ	gap	gap rate	Richardson	ATM	greedy
16383.2130.3.103	16,383	14,253	2,130	3	17	0.0010	57,608	51,717	N/A
4095.737.3.101	4,095	3,358	737	3	9	0.0022	15,089	13,215	N/A
1057.244.3.352	1,057	813	244	3	4	0.0038	4,090	3,472	3,751
1057.244.3.457	1,057	813	244	3	6	0.0057	4,070	3,485	3,777
1057.244.3.353	1,057	813	244	3	6	0.0057	4,072	3,483	3,811
4161.731.4.356	4,161	3,430	731	4	39	0.0093	24,644	23,399	N/A
4161.731.4.352	4,161	3,430	731	4	42	0.0101	24,745	23,744	N/A
4095.738.4.102	4,095	3,358	737	4	45	0.0110	20,111	20,672	N/A
1057.244.4.389	1,057	814	243	4	17	0.0161	5,254	4,920	5,496
1057.244.4.364	1,057	814	243	4	17	0.0161	5,255	4,971	5,484
1057.244.4.360	1,057	814	243	4	17	0.0161	5,236	4,975	5,406
8000.4000.3.483	8,000	4,000	4,000	3	148	0.0185	49,727	59,694	N/A
4000.2000.3.243	4,000	2,000	2,000	3	75	0.0188	22,218	23,143	N/A
816.3.174	816	410	406	3	16	0.0196	4,029	3,439	4,173
504.504.3.504	1,008	504	504	3	20	0.0198	5,077	4,399	5,025
4986.93xb.329	9,972	4,986	4,986	3	211	0.0212	75,743	87,462	N/A
4986.93y.654	9,972	4,986	4,986	3	212	0.0212	75,716	84,895	N/A
816.1A4.843	816	273	543	4	77	0.0944	8,190	15,012	10,984
816.55.156	816	408	408	5	77	0.0944	8,529	12,494	11,296
816.55.134	816	408	408	5	79	0.0968	8,671	12,693	11,204
816.55.178	816	408	408	5	81	0.0993	8,713	13,134	11,544

the first $r-1$ row vectors of U are computable by using (6). If we bring the j-th column vector of A, where $j \geq r$, to the r-th column position, then the r-th column of L must be defined as

$$l_{i,r} = a_{i,j} + \sum_{k=1}^{r-1} l_{i,k} u_{k,j} \pmod 2 \qquad (i \geq r).$$

Examine the above equation for each j with $j \geq r$, and we can find the "best" column vector which makes the weight of the r-th column vector of L minimum. Bring the best column at the r-th column position in A. In a similar way, find the "best" row vector which makes the weight of the r-th row vector of U minimum, and bring it to the r-th row position. One small issue we need to remark is the LP-condition. We need to choose the r-th row and r-th column vectors so that the the r-th order leading principle of H_2 is nonsingular. Thus, for the choice of vectors, we should exclude some combinations of vectors which fail to make the leading principle nonsingular. For this sake, we can use the technique used in the proof of Lemma 2.1, but the detailed discussion is omitted here.

The above algorithm can be used to matrices which have more columns than rows. Give the algorithm a $P \times N$ check matrix H, and execute the algorithm for P iterations. The algorithm determines the first P row vectors and the first P column vectors. Let H_2 be the $P \times P$ left submatrix of the resulted matrix, and let H_1 be the remaining $P \times K$ submatrix. The row and column vectors of H_2 have been chosen, in a greedy manner, so that they make L and U sparse.

Example 4.2: Let C be the Gallagar code considered in Example 4.1. Figure 2 shows H_2, L and U for the greedy permutation. In this case, $|H_1| = 1,512$, $|L| = 1,863$ and $|U| = 1,641$. The total number of operations is $\gamma_{\mathrm{TSTF}} = 5,025$, and for this code, the greedy permutation is not as good as the permutation based on the ATM form. □

4.3 Simulation Results

To compare the complexity of the proposed algorithm with that of the Richardson's algorithm, the number of operations needed in the algorithms is evaluated for several LDPC

codes found at [6]. Table 2 shows code parameters and the number of operations. The result is in the increasing order of the gap rate which is the rate of the gap g to the code length N. In the table, N, K, P and λ denote the code length, the number of information symbols, the number of parity check symbols and the column weight of the code, respectively. All the codes shown in the table are Gallager code. Some codes have the same parameters but they are constructed using different random seeds. The "name" is the actual file name of the check matrix found at [6]. The number of operations is evaluated for the Richardson's algorithm, the proposed algorithm with the ATM based permutation and the proposed algorithm with the greedy permutation. For codes with relatively large parameters, the data was not available (N/A in the table) for the greedy permutation due to big computational complexity for the greedy algorithm.

We can see that, for several codes, the proposed algorithm consumes smaller number of operations than the Richardson's algorithm. In general, the proposed algorithm with the ATM based permutation is more efficient than the Richardson's algorithm if the size of the gap is relatively small. This coincides with the analytical discussion in Sect. 4.1.1. On the other hand, for codes with relatively large gap, the proposed algorithm with the ATM based permutation consumes a large number of operations. This is because that the size of the dense submatrix B_U becomes large if the gap size is large. In this case, the greedy permutation is better than the ATM based permutation, though, the number of operations of the TSTF algorithm is still bigger than that of the Richardson's algorithm for these codes.

5. Concluding Remarks

An algorithm for encoding LDPC codes is proposed. The algorithm computes parity check symbols by solving a system of sparse equations. For solving the equations, we make use of the triangular factorizations to reduce the number of operations. We also investigated permutations of check matrices, and showed by computer simulation that the proposed algorithm consumes smaller number of operations than Richardson's algorithm, if the code has relatively small gap. As discussed in [8], good LDPC codes have small gaps in general, and this perspective suggests that the encoding algorithm proposed in this paper contributes to reduce the complexity for encoding LDPC codes. We would like to note that the permutations considered in this paper are just two of suboptimum solutions. The complexity of the TSTF algorithm can be further reduced if we could find more sophisticated permutations.

Acknowledgement

The author would like to thank anonymous referees for their valuable comments. The author is also grateful to Professor Shu Lin of the University of California Davis and Professor Marc P. C. Fossorier of the University of Hawaii at Manoa for motivating this study.

References

[1] L. Chen, L. Lan, S. Lin, and K. Abdel-Ghaffar, "An algebraic method for constructing quasi-cyclic LDPC codes," Proc. International Symposium on Information Theory and Its Applications, pp.535–539, Parma, Italy, 2004.
[2] H. Fujita and K. Sakaniwa, "Some classes of quasi-cyclic LDPC codes: Properties and efficient encoding method," IEICE Trans. Fundamentals, vol.E88-A, no.12, pp.3627–3635, Dec. 2005.
[3] A. Khandekar, Graph-Based Codes and Iterative Decoding, Ph.D. Thesis, California Institute of Technology, 2002.
[4] 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, 2001.
[5] D.J.C. MacKay, "Good error-correcting codes based on very sparse matrices," IEEE Trans. Inf. Theory, vol.45, no.2, pp.399–432, 1999.
[6] D.J.C. MacKay, Encyclopedia of Sparse Graph Codes, available at http://www.inference.phy.cam.ac.uk/mackay/
[7] S. Myung, K. Yang, and J. Kim, "Quasi-cyclic LDPC codes for fast encoding," IEEE Trans. Inf. Theory, vol.51, no.8, pp.2894–2901, 2005.
[8] 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, 2001.
[9] R.P. Tewarson, Sparse Matrices, Academic Press, 1973.

Appendix: The Proof of Lemma 2.1

The proof is given constructively. If A does not satisfy the LP-condition, then there exists an integer k such that the k-th order leading principle of A is nonsingular, but the $k+1$-st order leading principle of A is singular. Write

$$A = \begin{pmatrix} X & Y \\ Z & W \end{pmatrix},$$

where X is the k-th order leading principle of A (the size of X is therefore $k \times k$) and the size of the other submatrices are defined accordingly. The rank of X is k and we can use the Gaussian elimination technique to eliminate nonzero components in Z. This transforms A into

$$A_e = \begin{pmatrix} X & Y \\ O & W' \end{pmatrix},$$

where O is a zero matrix. Note that the rank of matrices is not changed by the Gaussian elimination and hence the rank of A_e must be n. This implies that W' is not a zero matrix, because if W' were a zero matrix then the rank of A_e is defined to be k, the rank of X, and a contradiction is derived. Let (i, j) with $k+1 \le i \le n$ and $k+1 \le j \le n$ be a position of a nonzero component in A_e. The position (i, j) points a nonzero component in W'. Now we come back to the matrix A, and define A' be the matrix which is obtained from A by exchanging the $k+1$-st row vector and the i-th row vector, and simultaneously by exchanging the $k+1$-st column vector and the j-th column vector. By applying the Gaussian elimination to the $k+1$-st leading principle of A', the leading principle is transformed to

$$A'_e = \begin{pmatrix} X & c \\ O & 1 \end{pmatrix}$$

where $\boldsymbol{c}$ is the $(j-k)$-th column vector of Y. This matrix A_e has rank $k+1$ and hence nonsingular. By repeatedly applying this kind of permutations, we can transform A into a matrix which satisfies the LP-condition.

Yuichi Kaji was born in Osaka, Japan, on December 23, 1968. He received the B.E., M.E., and Ph.D. degrees in information and computer sciences from Osaka University, Osaka, Japan, in 1991, 1992 and 1994, respectively. In 1994, he joined Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. In 2003 and 2004, he visited the University of California Davis and the University of Hawaii at Manoa as a visiting researcher. His current research interests include the theory of error correcting codes, fundamental techniques for information security, and the theory of automata and rewriting systems. He is a member of IPSJ, SITA and IEEE.