A Minimum Weight Test Procedure for a Certain Subclass of Array LDPC Codes

Kenji SUGIYAMA * Yuichi KAJI *

January 12, 2008

Abstract

LDPC codes is a class of linear codes introduced by Gallager in early 60's. Array LDPC (ALDPC) codes is a class of LDPC codes which are algebraically constructed from a family of array codes. This paper proposes a procedure to check if there is a codeword with specified weight in a certain ALDPC code. The minimum weight of a linear code has strong relationship to the performance of the code, but unfortunately it is difficult to compute the exact minimum weight of long and randomly constructed LDPC codes. We restrict ourselves to a class of complete array LDPC codes (C-ALDPC codes) which is a subclass of array LDPC codes, and investigate positions of nonzero components in a codeword. The code in the considered subclass is invariant under a doubly transitive group of affine permutations. This property gives significant constraint on the positions of nonzero components in a codeword, which means that the positions of nonzero components in a codeword can be classified into rather small number of patterns. Using these conditions, the proposed procedure checks if there exists a codeword with specified weight.

1 Introduction

This paper discusses a method for computing the *minimum weight* of a certain subclass of *low-density parity check codes* (*LDPC codes*). The minimum weight of a linear code

*Nara Institute of Science and Technology 8916-5 Takayama, Ikoma, Nara 630-0101, JAPAN E-mail: {kenji-su, kaji}@is.naist.jp

has strong relationship to the performance of the code, and much efforts have been devoted for constructing a linear code with large minimum weight. It is sometimes said that the minimum weight is not the most important parameter for designing good LDPC codes, because other parameters such as the girth of the code affect the error correcting performance of iterative decoding algorithms. However, it is widely considered that the minimum weight is still one of the most significant parameters which affect the performance of LDPC codes.

Unfortunately, it is difficult to compute the exact minimum weight of long and randomly constructed LDPC codes. Tanner discussed in [12] a certain bound on the minimum distance of LDPC codes by utilizing the Tanner Graph. Hu and Fossorier proposed a probabilistic procedure to compute the minimum weight of LDPC codes by using a decoding algorithm for LDPC codes[7]. Hirotomo et al. proposed another probabilistic procedure[6] which utilizes the Stern's algorithm. To the authors' knowledge, though, there is no efficient algorithm which can compute the exact minimum weight of arbitrary LDPC codes. For a class of LDPC codes with certain structures, we may make use of the structure to compute the minimum weight. MacKay showed that the minimum distance of regular-Quasi-Cyclic LDPC codes with column weight j is less than or equal to $(j-1)!$[8]. Mittelholzer derived an upper-bound limit of the minimum weight of *array* LDPC codes[9], and Yang et al. has computed exact minimum weights of array LDPC codes with small parameters[13] by making use of algebraic properties of array LDPC codes. The problem of Yang's approach is that the theoretical analysis becomes too complicated if the code parameter is not small. To get around this problem, one of the authors has proposed a semi-automatic procedure to compute the exact minimum weight of array LDPC codes. The procedure replaces certain operations in Yang's approach by computer search[11], and successfully revealed the minimum weight of the $C_A(7,5)$ array code (see the following sections for the parameters of array LDPC codes). However, we still have difficulty to use the procedure to compute the minimum weight of array codes with bigger parameters.

The procedure investigated in [11] determines, for a given integer w, if there is a codeword with weight w or not. If the code length is N, then there are ${}_NC_w$ possible patterns for positions of nonzero components in a vector of weight w. This number is quite large if N is large, though, we can eliminate certain patterns of nonzero positions due to the mathematical structure of array LDPC codes. This classifies the patterns of nonzero positions into some classes. For each class of patterns, we use computer search to test if there is a codeword which fulfills the symbol pattern. The number of patterns can be small for small parameters, but the number of patterns increases as the code parameter increases, and the problem is that we did not have a systematic way to enumerate possible patterns of nonzero symbols.

This paper proposes a procedure to enumerate possible patterns of positions which can accommodate given number w of nonzero symbols in a codeword. If no pattern is enumerated by the algorithm, then there is no chance that the code has a codeword with weight w. If there are possible patterns of positions, then, for each possible pattern, we examine if there really exist codewords which fit in the symbol pattern. This will contribute to reveal exact minimum weights of array LDPC codes with relatively large parameters.

2 Array LDPC Codes

An array LDPC codes[2, 3] is a class of LDPC codes which is constructed based on array codes[1, 4]. Let p be a prime number, and k and j be integers satisfying $k, j \leq p$. The binary *array LDPC code* $C_A(p, j, k)$ is a null space of the $pj \times pk$ binary matrix

$$H_A(p,j,k) = \begin{bmatrix} I & I & \cdots & I \\ I & P & \cdots & P^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ I & P^{j-1} & \cdots & P^{(j-1)(k-1)} \end{bmatrix}$$

where I is the $p \times p$ identity matrix and P is a cyclic shift matrix defined by

$$P = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}.$$

Now we consider a special case such that $k = p$, and call this special array LDPC code a *complete array LDPC code* (*C-ALDPC code* for short). For simplicity, $C_A(p, j, p)$ and $H_A(p, j, p)$ are respectively written as $C_A(p, j)$ and $H_A(p, j)$. We denote $d(p, j)$ the *minimum distance* of $C_A(p, j)$. The *rate* of $C_A(p, j)$ is $1 - (pj - j + 1)/p^2$. It can be easily shown that column vectors in $H_A(p, j)$ are all different, and every codeword has even weight. It is also obvious from the definition that a column vector of $H_A(p, j)$ can be decomposed into j subsequences which have length p and weight one. That is, if we write $H_A(p, j) = [h_{l,i}]$ $(1 \le i \le p^2$ and $1 \le l \le pj)$, then the weight of

$$\boldsymbol{h}_i^r = (h_{p(r-1)+1,i}, h_{p(r-1)+2,i}, \ldots, h_{p(r-1)+p,i})^T$$

is exactly one for any $1 \le i \le p^2$ and $1 \le r \le j$ (note: the indices starts from one). For a vector $\boldsymbol{v}$ with weight one, let $\phi(\boldsymbol{v})$ denote the position of the nonzero component in $\boldsymbol{v}$ where the position of the first component of $\boldsymbol{v}$ is regarded as zero. For example, $\phi((0, 1, 0, 0)^T) = 1$ and $\phi((0, 0, 0, 1)^T) = 3$. The value of ϕ is not defined for a vector whose weight is not one. Extend this notation ϕ to a column vector $\boldsymbol{h}_i = (h_{1,i}, h_{2,i}, \ldots, h_{pj,i})^T$ of $H_A(p, j)$ in such a way that

$$\phi(\boldsymbol{h}_i) = (\phi(\boldsymbol{h}_i^1), \ldots, \phi(\boldsymbol{h}_i^j))^T,$$

and also extend ϕ to the matrix $H_A(p, j)$ as

$$\phi(H_A(p, j)) = [\phi(\boldsymbol{h}_1), \ldots, \phi(\boldsymbol{h}_{p^2})].$$

For the matrix $H_A(3, 2)$, we have for example

$$\phi(H_A(3, 2)) = \begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \end{bmatrix}.$$

Lemma 2.1 *For a column position i with $1 \le i \le p^2$, let k and k' be integers satisfying $i = pk + k'$ and $1 \le k' \le p$. The i-th column vector of $\phi(H_A(p, j))$ can be written as*

$$(k' - 1, k' - 1 + k, \ldots, k' - 1 + (j - 1)k)^T \pmod{p}.$$

<u>*Proof.*</u> *Obvious from the construction of $H_A(p, j)$.*

Lemma 2.2 *Consider two different columns $\phi(\boldsymbol{h}_{i_1})$ and $\phi(\boldsymbol{h}_{i_2})$ in $\phi(H_A(p,j))$ (thus $i_1 \neq i_2$ and $1 \leq i_1, i_2 \leq p^2$). If the j_1-th components of $\phi(\boldsymbol{h}_{i_1})$ and $\phi(\boldsymbol{h}_{i_2})$ are the same, then, for any other j_2 with $j_1 \neq j_2$ and $1 \leq j_2 \leq j$, the j_2-th components of $\phi(\boldsymbol{h}_{i_1})$ and $\phi(\boldsymbol{h}_{i_2})$ cannot be the same.*

Proof. *Let k_1, k_1', k_2 and k_2' be integers satisfying $i_1 = pk_1 + k_1'$ and $i_2 = pk_2 + k_2'$. If there is an integer j_2 in the proposition of the lemma, then*

$$\begin{aligned}(j_1 - 1)k_1 + k_1' &\equiv (j_1 - 1)k_2 + k_2' \bmod p,\\ (j_2 - 1)k_1 + k_1' &\equiv (j_2 - 1)k_2 + k_2' \bmod p,\end{aligned}$$

from Lemma 2.1. The equations reduce to $(k_1 - k_2)(j_1 - j_2) \equiv 0 \bmod p$, but this cannot happen because both of $(k_1 - k_2)$ and $(j_1 - j_2)$ are relatively prime to p, a contradiction.

As for the notation ϕ, we have the following lemma.

Lemma 2.3 *Let $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ be binary vectors with length p and weight one, and define $N_a = \#\{v_i | \phi(v_i) = a\}$. We have that $\boldsymbol{v}_1 + \cdots + \boldsymbol{v}_n = \boldsymbol{0} \bmod 2$ if and only if N_a is an even number for any a with $0 \leq a \leq p-1$ (we call this condition a* cancel-out condition*).*

Proof. *Note that $\boldsymbol{v}_1 + \cdots + \boldsymbol{v}_n = (N_0, \ldots, N_{p-1})^T$. If this sum equals to zero in the modulus of two, then $N_a \equiv 0 \bmod 2$ for all a.*

Let $\boldsymbol{v} = (v_1, \ldots, v_{p^2})$ be a binary vector of length p^2. The vector $\boldsymbol{v}$ is a codeword of $C_A(p,j)$ if and only if $H\boldsymbol{v}^T = \boldsymbol{0} \bmod 2$. Now define

$$\mathrm{supp}(\boldsymbol{v}) = \{\phi(\boldsymbol{h}_i) | 1 \leq i \leq p^2, v_i = 1\},$$

and call it *the support of $\boldsymbol{v}$*[1]. The support contains the column vectors of $\phi(H_A(p,j))$ which correspond to nonzero components of $\boldsymbol{v}$. Since all column vectors in $H_A(p,j)$ are different, if the weight of $\boldsymbol{v}$ is w, then $\mathrm{supp}(\boldsymbol{v})$ contains exactly w column vectors. Order these w column vectors in an arbitrary way, and construct a matrix $S_{\boldsymbol{v}} = [\boldsymbol{s}_1, \ldots, \boldsymbol{s}_w]$. We call this matrix a support matrix of $\boldsymbol{v}$. Note that the vector $\boldsymbol{v}$ is a codeword of $C_A(p,j)$

[1]The word "support" is often used to denote the set of positions of nonzero components in a vector, but we use the word in slightly different manner.

if and only if $H\boldsymbol{v}^T = \boldsymbol{0} \bmod 2$. Thus, $\boldsymbol{v} \in C_A(p, j)$ if and only if the sum of all column vectors in

$$\{\boldsymbol{h}_i | 1 \leq i \leq p^2, v_i = 1\}$$

equals to zero in the modulus of two. By applying Lemma 2.3 to each row of $S_{\boldsymbol{v}}$, we have the following corollary.

Corollary 2.4 *The vector $\boldsymbol{v}$ is a codeword of $C_A(p, j)$ if and only if the cancel-out condition holds for any row of $S_{\boldsymbol{v}}$.*

For example, $\boldsymbol{v}_1 = 100101001$ is a correct codeword but $\boldsymbol{v}_1 = 100101000$ is not because

$$S_{\boldsymbol{v}_1} = \begin{bmatrix} 0 & 0 & 2 & 2 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \quad S_{\boldsymbol{v}_2} = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 0 \end{bmatrix}.$$

In $S_{\boldsymbol{v}_2}$, 2 occurs only once in the first row, and the cancel-out condition does not hold.

3 Support Matrices

3.1 Constraints on Positions of Zeros

If $\boldsymbol{v}$ is a codeword of $C_A(p, j)$ with weight w, then, without loss of generality, we can assume that the support matrix $S_{\boldsymbol{v}} = [s_{r,i}]$ of $\boldsymbol{v}$ satisfies the following conditions[13]:

- $s_{1,1} = s_{2,1} = \cdots = s_{j,1} = 0$, and
- $s_{1,k} = s_{1,k+1}$ for any odd k with $1 \leq k < w$.

By using the second property above and Lemma 2.1, we also have $s_{r,2} = (r-1)s_{2,2}$ for $2 \leq r \leq j$. Therefore $S_{\boldsymbol{v}}$ can be written as;

$$\left[\begin{array}{cc|cc|c|cc} 0 & 0 & e_1 & e_1 & \cdots & e_{w/2-1} & e_{w/2-1} \\ \hline 0 & s_{2,2} & s_{2,3} & s_{2,4} & \cdots & s_{2,w-1} & s_{2,w} \\ 0 & 2s_{2,2} & s_{3,3} & s_{3,4} & \cdots & s_{3,w-1} & s_{3,w} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & (j-1)s_{2,2} & s_{j,3} & s_{j,4} & \cdots & s_{j,w-1} & s_{j,w} \end{array}\right], \qquad (1)$$

where e_i with $1 \leq i \leq w/2 - 1$ satisfies $0 \leq e_i \leq p - 1$. Furthermore, a column vector of $S_{\boldsymbol{v}}$ (excluding the the first column) cannot have two or more zeros because of Lemma 2.2 and the fact that the first column of $S_{\boldsymbol{v}}$ consists of zeros only. That is, a column vector of $S_{\boldsymbol{v}}$ can include at most one zero (except the first column). We refer this property P1. On the other hand, if $\boldsymbol{v}$ is a codeword, then Corollary 2.4 assures that $S_{\boldsymbol{v}}$ must satisfy the cancel-out condition and therefore each row must contain even number of zeros. Because the first column of $S_{\boldsymbol{v}}$ consists of zeros, this means that each row must contain at least one zero between the second to the w-th position (we refer this property P2). The properties P1 and P2 can be regarded as a necessary condition for $\boldsymbol{v}$ to be a correct codeword. In the following, we consider classes of support matrices which satisfy the properties P1 and P2, and verify if there is a support matrix which fulfills the cancel-out condition in the considered class of matrices.

3.2 Zero Locaters

To deal with classes of matrices, we introduce the notion of a *zero locater*. A *zero locater for a weight w vector* is a w-tuple $\boldsymbol{z} = (z_1, \ldots, z_w)$ satisfying the following conditions.

- $z_i \in \{1, \ldots, j\} \cup \{\triangle, \perp\}$ for $1 \leq i \leq w$ where $\triangle$ and $\perp$ are special symbols, and
- no integer appears more than once in $\boldsymbol{z}$.

Intuitively, z_i with $1 \leq i \leq w$ denotes the row position of zero in the i-th column vector of a support matrix. If there is no zero in the i-th column, then $z_i = \perp$, and if the i-th column is zero-vector, then $z_i = \triangle$. For zero locaters which correspond to matrices of the form (1), the first two components are always $z_1 = \triangle, z_2 = 1$.

We say that a support matrix $S_{\boldsymbol{v}} = [s_{r,i}]$ of a vector $\boldsymbol{v}$ (with weight w) *fulfills* the zero locater $\boldsymbol{z}$ if $S_{\boldsymbol{v}}$ is of the form (1) and the (z_i, i) component of $S_{\boldsymbol{v}}$ is zero for all z_i with $z_i \neq \perp$. We also say that a vector $\boldsymbol{v}$ *fulfills* the zero locater $\boldsymbol{z}$ if at least one of the support matrices of $\boldsymbol{v}$ fulfills $\boldsymbol{z}$ (remind that the order of column vectors in a support matrix can be changed). Let denote $V(\boldsymbol{z})$ the set of vectors (of weight w) which fulfill the zero locater $\boldsymbol{z}$. We shall give an example of these in section 4.

The following lemmas can be shown easily because we can change the order of column vectors in a support matrix.

Lemma 3.1 *If*

$$\begin{aligned} z &= (\ldots, z_{2k+1}, z_{2k+2}, \ldots), \\ z' &= (\ldots, z_{2k+2}, z_{2k+1}, \ldots), \end{aligned}$$

(the $2k+1$-th component and the $2k+2$-th component are exchanged), then $V(z) = V(z')$.

Lemma 3.2 *If*

$$\begin{aligned} z &= (\ldots, z_{2k+1}, z_{2k+2}, \ldots, z_{2k'+1}, z_{2k'+2}, \ldots), \\ z' &= (\ldots, z_{2k'+1}, z_{2k'+2}, \ldots, z_{2k+1}, z_{2k+2}, \ldots), \end{aligned}$$

(two couples of column vectors are exchanged), then $V(z) = V(z')$.

The following lemma associate zero locaters and the properties P1 and P2.

Lemma 3.3 *If a support matrix of a vector $\boldsymbol{v}$ satisfies the properties P1 and P2, then there is a zero locater which $\boldsymbol{v}$ fulfills. That is, for a certain zero locater $\boldsymbol{z} = (z_1, \ldots, z_w)$, $\boldsymbol{v} \in V(\boldsymbol{z})$. Furthermore, $z_3 = 2$ without loss of generality.*
Proof. *The former part of the lemma is a obvious consequence from the property P1. The latter part is from the property P2 together with Lemmas 3.1 and 3.2.*

Remind that to satisfy the properties P1 and P2 is a necessary condition for a vector to be a codeword. Thus we have the following corollary.

Corollary 3.4 *For a codeword $\boldsymbol{v}$, there is a zero locater $\boldsymbol{z} = (\triangle, 1, 2, z_4, \ldots, z_w)$ satisfying $\boldsymbol{v} \in V(\boldsymbol{z})$.*

3.3 Procedure for the Weight Test

Summarizing the above discussion, we can consider the following procedure to determine if there is a codeword of weight w.

1. Enumerate all zero locaters of the form $z = (\triangle, 1, 2, z_4, \ldots, z_w)$.

2. Find support matrices (choices of column vectors of $\phi(H_A(p, j))$) which fulfill the zero locater.

3. Verify if the support matrix satisfies the cancel-out condition.

If we can find a support matrix which satisfies the cancel-out condition by the above procedure, then there exists a codeword of weight w. If we cannot find such a matrix by the above procedure, then the code does not contain codewords of weight w.

To realize this procedure, we first need to enumerate all of zero locaters. This can be possible by the following recursive procedure.

Procedure zero-loc(z, R, i): z is a zero locater, $R \subseteq \{1, \ldots, j\}$ is a set of row positions which are not used in z and i is a column position.

1. Let r be the smallest integer in R, and let $R \leftarrow R \setminus \{r\}$.

2. Let z' be the zero locater which is obtained from z by replacing the i-th component in z with r.

3. If $R = \emptyset$, then output z'. Otherwise, perform the following operations.

4. Execute **zero-loc**(z', R, $i + 2$), if $i + 2 \le w$.

5. For each $r' \in R$, define $z'_{r'}$ as the zero locater which is obtained from z' by replacing the $i+1$-th component in z' with r'. If $R \setminus \{r'\} = \emptyset$, then output $z'_{r'}$. If $R \setminus \{r'\} = \emptyset$ and $i + 2 \le w$, then execute **zero-loc**($z'_{r'}$, $R \setminus \{r'\}$, $i + 2$).

Let $z_\perp = (\triangle, 1, \perp, \ldots, \perp)$. By executing **zero-loc($z_\perp$, $\{2, \ldots, j\}$, 3)**, all zero locaters of the form $z = (\triangle, 1, 2, z_4, \ldots, z_w)$ is output.

Given a zero locater $z = (\triangle, 1, z_3, \ldots, z_w)$, support matrices which fulfill z can be computed by the following procedure. Let S be the support matrix which we will going to construct.

1. Compute all assignments for e_i with $1 \le i \le w/2 - 1$ in (1) exhaustively, and perform the following operations for each assignment.

$$\phi(H_A(5,3)) = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 0 & 2 & 3 & 4 & 0 & 1 & 3 & 4 & 0 & 1 & 2 & 4 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 4 & 2 & 3 & 4 & 0 & 1 & 4 & 0 & 1 & 2 & 3 & 1 & 2 & 3 & 4 & 0 & 3 & 4 & 0 & 1 & 2 \end{pmatrix}$$

Figure 1: parity check matrix $H_A(5,3)$

2. For each i with $3 \le i \le w$ and $z_i \in \{2, \dots, j\}$, find the column vector, say $\boldsymbol{s}_i$, in $\phi(H_A(p,j))$ such that the first component of $\boldsymbol{s}_i$ is $e_{\lfloor i/2 \rfloor}$ and the z_i-th component of $\boldsymbol{s}_i$ is 0. By the construction of array LDPC codes, such a column vector exists uniquely. The vector $\boldsymbol{s}_i$ is used as the i-th column vector of S.

3. For the rest of column positions, perform an exhaustive search for column vectors. During the search, we can avoid assignment of column vectors whose first component do not coincide with the e_i-value which has been determined in the first step. And during the exhaustive search, it is possible to use some criteria to abandon hopeless computation. For example, assume that $w = 6$ and the procedure has determined four column vectors $\boldsymbol{s}_1, \dots, \boldsymbol{s}_4$ of S. If, for example, $s_{2,1} = 0$, $s_{2,2} = 1$, $s_{2,3} = 2$ and $s_{2,4} = 3$, then the procedure can abandon the current search trial because it is impossible to choose two column vectors and make the first row of S satisfy the cancel-out condition. (at least four more vectors are needed to make the first row satisfy the cancel-out condition). By making use of this kind of criteria, we can skip hopeless computation and make the procedure much more efficient than straightforward search algorithms.

For each of constructed support matrix, check if the cancel-out condition holds or not. If the condition holds, then the code $C_A(p,j)$ contains a codeword of weight w.

4 Example

Consider $C_A(5,3)$ code whose parity check matrix is given in Fig. 1. First consider if $C_A(5,3)$ contains a codeword of weight four. If there exists, then the support matrix of the minimum weight codeword must be written as

$$S = \begin{pmatrix} 0 & 0 & e_1 & e_1 \\ 0 & s_{2,2} & 0 & s_{2,4} \\ 0 & s_{3,2} & s_{3,3} & 0 \end{pmatrix}. \tag{2}$$

Vectors whose support matrices are in this form belong to $V(z)$ where z is a zero locater defined as $z = (\triangle, 1, 2, 3)$. If e_1 in (2) is chosen to be 1, then the third and the fourth columns of S are determined uniquely and S must be written as

$$S = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & s_{2,2} & 0 & 3 \\ 0 & s_{3,2} & 4 & 0 \end{pmatrix}.$$

To make this matrix satisfy the cancel-out condition, the second column must be $(0, 3, 4)^T$, but such a column is not in $H_A(5, 3)$. Thus the choice of $e_1 = 1$ is faulty. We can see easily that even if we choose e_1 different from 1, it is not possible to make S satisfy the cancel-out condition. Therefore, $C_A(5, 3)$ does not contain codewords of weight four.

Next consider if $C_A(5, 3)$ contains a codeword of weight six. In this case, we need to consider two zero locaters $z_1 = (\triangle, 1, 2, 3, \perp, \perp)$ and $z_2 = (\triangle, 1, 2, \perp, 3, \perp)$ which intuitively correspond to

$$S_1 = \begin{pmatrix} 0 & 0 & e_1 & e_1 & e_2 & e_2 \\ 0 & s_{2,2} & 0 & s_{2,4} & s_{2,5} & s_{2,6} \\ 0 & s_{3,2} & s_{3,3} & 0 & s_{3,5} & s_{3,6} \end{pmatrix}, \tag{3}$$

and

$$S_2 = \begin{pmatrix} 0 & 0 & e_1 & e_1 & e_2 & e_2 \\ 0 & s_{2,2} & 0 & s_{2,4} & s_{2,5} & s_{2,6} \\ 0 & s_{3,2} & s_{3,3} & s_{3,4} & 0 & s_{3,6} \end{pmatrix}, \tag{4}$$

respectively. Choosing $e_1 = 1$ in S_1 results in

$$S_1 = \begin{pmatrix} 0 & 0 & 1 & 1 & e_2 & e_2 \\ 0 & s_{2,2} & 0 & 3 & s_{2,5} & s_{2,6} \\ 0 & s_{3,2} & 4 & 0 & s_{3,5} & s_{3,6} \end{pmatrix},$$

We have four choices for the second column vector because $H_A(5, 3)$ have four unused column vectors whose first component is zero. If we choose $e_2 = 0$ or $e_2 = 1$, then four out of six components in the second row of S_1 must have different values, and S_1 cannot satisfy the cancel-out condition. Thus e_1 must be either of 2 or 3 or 4. For each choice of e_i, we have ${}_5C_2$ choices for the third and the fourth columns in S_1. Thus, there are $4 \times 3 \times {}_5 C_2 = 120$ patterns for the choice of column vectors for S_1 above. Computer search showed that none of the 120 patterns satisfy the cancel-out condition, and therefore the choice of $e_1 = 1$ is faulty. In a similar way, we can verify that there is no support matrix which is in the pattern of (3) and satisfy the cancel-out condition.

For support matrices in the pattern of (4), choose $e_1 = 1$ and $e_2 = 3$ for example. This uniquely determines the third and the fifth columns of S_2, and allows 4 choices each for the fourth and the sixth columns. Now consider to choose the two columns so that

$$S_2 = \begin{pmatrix} 0 & 0 & 1 & 1 & 3 & 3 \\ 0 & s_{2,2} & 0 & 4 & 4 & 1 \\ 0 & s_{3,2} & 4 & 2 & 0 & 4 \end{pmatrix}.$$

If we can choose so that $s_{2,2} = 1$ and $s_{3,2} = 2$, then S_2 satisfies the cancel-out condition. Indeed, the sixth column in $\phi(H_A(5,3))$ is exactly $(0,1,2)^T$. Thus we found a support matrix which satisfies the cancel-out condition. The codeword which corresponds to the support matrix

$$S_2 = \begin{pmatrix} 0 & 0 & 1 & 1 & 3 & 3 \\ 0 & 1 & 0 & 4 & 4 & 1 \\ 0 & 2 & 4 & 2 & 0 & 4 \end{pmatrix}$$

is 10000 10010 00000 01010 01000.

5 Minimum Weights Found by the Procedure

In the table 1, values marked with * indicate newly revealed minimum distance which are computed by the proposed procedure. The procedure also reconfirmed known results which are associated with bibliography numbers in the table. Yang showed that the lower bound of $d(p,4)$ is 10 for any prime number $p > 7$. Our procedure confirmed that $d(p,4)$ is exactly 10 for $p = 11, 13, 17, 19, 23, 29, 31, 37, 41, 43$ and 47, though results above $p = 23$ is not shown in the table 1.

6 Concluding Remarks

We proposed a procedure to test if a complete array LDPC code contains codewords of specified weight. The problem might be solvable by a naive procedure which tests all the combinations of column vectors in a check matrix, but such an approach will suffer for huge computational complexity caused by the number of combinations of column vectors. On the other hand, it is also possible to take a purely algebraic approach as in [13]. The problem of this approach is that the theoretical analysis becomes too complicated if the code parameter is not small. The procedure considered in this paper can be regarded as a hybrid approach of the algebraic and the computer-oriented approaches. Some algebraic analysis in [13] is replaced by a computer search in the proposed study, which allows us to use the basic idea of [13] to codes with bigger parameters. For example, by using the proposed technique together with the help by computer search, we found some exact minimum weight of $C_A(p, j)$, which were not shown in previous works.

MacKay investigates an upper bound of the minimum distance of general regular Quasi-Cyclic LDPC codes[8], and Fossorier discusses the minimum distance of Quasi-Cyclic LDPC codes from circulant permutation matrices[5]. Those results suggest that if the column weight (parameter j) is fixed to a certain constant, then extending the code length (i.e. increasing the parameter p in the C-ALDPC code) will not contribute to increase the minimum distance. Interestingly, our results well agree with the observation. As shown in Table 1, the minimum weight depends on the column weight j, but not on the parameter p.

The proposed procedure still have some problems. For example, the classification of

Table 1: discovered minimum weights

p	codelength	$j=4$	$j=5$	$j=6$
5	25	8[13]		
7	49	8[13]	12[11]	12[10]
11	121	10*	10*	14 $\leq$*
13	169	10*	12*	14*
17	289	10*	12*	
19	361	10*	12*	

the zero locaters is still very rough, and we may have much overlap on the set of vectors covered by different zero locaters. We also remark that the algorithm for the computer search should make use of some more algebraic conditions to narrow the search space.

References

[1] M. Blaum and R. M. Roth, *New Array Codes for Multiple Phased Burst Correction*, IEEE Trans. Inform. Theory, Vol. 39, pp.66–77, January, 1993

[2] J. L. Fan, *Array Codes as Low-density Parity-check Codes*, Proc. of 2nd Int. Symp. on Turbo Codes, pp.543–546, 2000

[3] J. L. Fan, *Constrained Coding and Soft Iterative Decoding*, Kluwer Academic Publishers, 2001

[4] P. G. Farrell, *A Survey of Array Error Control Codes*, European Trans. on Telecommunications, Vol. 3 No. 5, pp.441–454, 1992

[5] M. P. C. Fossorier, *Quasi-Cyclic Low Density Parity Check Codes from Circulant Permutation Matrices*, IEEE Transactions on Information Theory, pp.1788–1793, August, 2004

[6] M. Hirotomo and M. Mohri, and M. Morii, *A Probabilistic Computation Method for the Weight Distribution of Low-Density Parity-Check Codes*, Proc. IEEE Intl. Sympl. Information Theory, Adelaide, Australia, pp.2166–2170, September, 2005

[7] X. Y. Hu and M. P. C. Fossorier, *On the Computation of the Minimum Distance of Low-Density Parity-Check Codes*, Proc. of the IEEE International Conference on Communications, Paris, June, 2004

[8] D. J. C. MacKay and M. Davey, *Evaluation of Gallager Codes for Short Block Length and High Rate Applications*, IMA Workshop on Codes, Systems and Graphical Models, 1999

[9] T. Mittelholzer, *Efficient Encoding and Minimum Distance Bounds of Reed-Solomon-type Array Codes*, Proc. of 2002 IEEE Int. Symp. on Information Theory, p.282, 2002

[10] K. Sugiyama and Y. Kaji, *A minimum weight test for a certain subclass of array LDPC codes*, IEICE Technical Report IT2006-2(2006-5), pp.7–11, 2006

[11] K. Sugiyama and T. Kohnosu, *On the Minimum Distance of Array-type LDPC Codes*, Proc. on SITA2004, December, 2004

[12] R. M. Tanner, *Minimum-distance Bounds by Graph Analysis*, IEEE Trans. Inform. Theory, Vol.47, No.2, pp.808–821, December, 2003.

[13] K. Yang and T. Helleseth, *On the Minimum Distance of Array Codes as LDPC Codes*, IEEE Trans. Inform. Theory, Vol. 49, No.12, pp.3268–3271, December, 2003.