Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2007, Article ID 45768, 8 pages
doi:10.1155/2007/45768

Research Article

Quasi-Cyclic LDPC Codes of Column-Weight Two Using a Search Algorithm

Gabofetswe Malema and Michael Liebelt

School of Electrical and Electronic Engineering, The University of Adelaide, North Terrace, Adelaide 5005, SA, Australia

Received 16 February 2006; Revised 11 August 2006; Accepted 6 February 2007

Recommended by Richard Heusdens

This article introduces a search algorithm for constructing quasi-cyclic LDPC codes of column-weight two. To obtain a submatrix structure, rows are divided into groups of equal sizes. Rows in a group are connected in their numerical order to obtain a cyclic structure. Two rows forming a column must be at a specified distance from each other to obtain a given girth. The search for rows satisfying the distance is done sequentially or randomly. Using the proposed algorithm regular and irregular column-weight-two codes are obtained over a wide range of girths, rates, and lengths. The algorithm, which has a complexity linear with respect to the number of rows, provides an easy and fast way to construct quasi-cyclic LDPC codes. Constructed codes show good bit-error rate performance with randomly shifted codes performing better than sequentially shifted ones.

1. INTRODUCTION

Gallager has shown that column-weight-two codes have minimum distance increasing logarithmically with code length, compared to a linear increase when the column-weight is at least three [1]. Despite the low increase in minimum distance, these codes have shown potential in some applications such as partial response channels [2, 3]. These codes also have less computation because columns have only two connections.

Although LDPC code performance has been shown to be good, their hardware implementation still remains a challenge. This is mainly because of their large sizes and complex random (unstructured) row-column connections. Applications have power, area, latency, and cost constraints that LDPC encoders and decoders must meet. Structured codes have been developed to reduce hardware implementation complexity by constraining code construction. However, constraining row-column connections may reduce the maximum girth (smallest cycle) attainable [4]. It has been shown that increasing the girth or average girth of a code increases its decoding performance [5, 6]. The girth also determines the number of iterations before a message propagates back to its original node. Performance of structured codes could therefore be improved by increasing their girths.

In [7] girth 16 and 18 codes with row-weights of 4 and 3 are constructed from graphical models. The method does not provide an easy way of constructing codes for high row-weights and expanding codes. In [3] cyclic codes are constructed algebraically with girth of twelve for row-weights of k, where $k - 1$ is prime. Large-girth column-weight-two codes can also be derived from distance graphs [8]. However, most of the derived codes are not easily implementable because of their row-column interconnection structure. In this paper, we construct quasi-cyclic codes with a wide range of girths, rates, and lengths using a search algorithm. Quasi-cyclic codes have a structure that is relatively easy to implement in hardware for both encoder and decoder[9, 10].

This paper is organized as follows. Section 2 describes a non-bipartite graph representation of LDPC codes. With this representation LDPC codes can be derived from distance graphs. Bit-filling and progressive-edge growth algorithms are briefly described in Section 3. The proposed algorithm is introduced in Section 4 from which a wide range of codes is obtained. Bit-error rate (BER) performances of the obtained codes are simulated and evaluated. Hardware implementation issues of these codes are also discussed. Section 5 has concluding remarks.

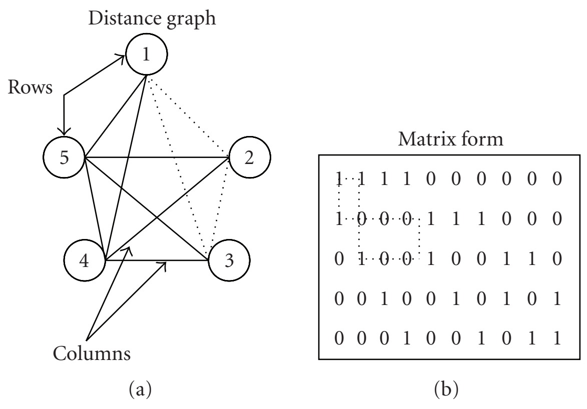

Figure 1: Graph and matrix LDPC code representation.

2. LDPC REPRESENTATION

A LDPC code matrix is usually represented by a Tanner or bipartite graph in which rows (check nodes) are one set of vertices and columns (variable nodes) are another set of vertices. Check and variable nodes are connected by an edge if the corresponding row and column have a "1" entry in the matrix. A LDPC code matrix can also be represented by a non-bipartite or distance graph in which vertices are rows and edges represent columns. A distance graph is a connected graph with M vertices, a smallest cycle length of g and average vertex degree of k. With this representation connected vertices of the graph represent rows that are connected to the same column in the matrix form. In the case of two rows per column (column-weight of two), a single edge between two vertices represents a column. Figure 1 shows a distance graph of five vertices with a minimum cycle length of three. Taking each vertex as a row and each edge as a column a corresponding matrix is formed as in the figure. The connections are represented by "1" entries in the matrix. A parity-check matrix, H, entry is equal to "1" or $H_{x,y} = 1$, if vertices v_x and v_y are connected in the graph. The number of "1" entries in a row, k, is equal to the number of edges of the corresponding vertex. The number of rows is equal to the number of vertices in the graph whereas the number of columns is equal to the number of edges. In general the size of a derived column-weight-two LDPC code matrix from a distance graph is given by $M \times Mk/2$, where M is the number of vertices in the graph and $Mk/2$ is the number of edges. The notation (N, j, k) is often used to show size and rate of a code, where N is the number of columns or length of a code and j and k are column and row-weights, respectively. The rate of a code is given by $1 - j/k$, hence for these codes the rate is $1 - 2/k$.

A cycle length of g in the graph corresponds to a cycle of length $2g$ in matrix form. In the graph we calculate the length using either vertices or edges only. In matrix form a cycle alternates between rows and columns. Therefore, the graph cycle represents half of the cycle. A cycle of three in the example graph is shown in dotted lines between vertices 1, 2, and 3. It forms a cycle of length six between rows 1, 2, 3 and columns 1, 2, 5 in matrix form. A cycle of length four in a parity-check matrix is formed if a pair of vertices are connected more than once in the corresponding graph representation. Four cycles can be broken by not connecting any two rows of a code more than once in the graph representation; a condition also known as the row-column constraint [11].

3. SEARCH ALGORITHMS

Random or pseudorandom construction algorithms such as bit-filling (BF) and progressive-edge growth (PEG) have been developed to construct a wide range of codes. The BF algorithm introduced in [12] constructs a LDPC code by connecting rows and columns of a code one at a time provided that a targeted girth is not violated. The number of connections to rows and columns is kept mostly evenly distributed by randomly selecting rows or columns with the least number of connections first. The algorithm obtains irregular codes with either a fixed row or column-weight. Although the algorithm produces high-rate and high-girth codes given a particular code size, the resulting codes are not easily implementable in hardware. This is mainly because the structure of row-column connections is not consistent enough to be an advantage in hardware implementation. The objective of the algorithm is to optimize girth or rate.

The PEG algorithm [13] is also a simple nonalgebraic algorithm that can be used to construct codes of arbitrary length and rate. It is similar to the bit-filling algorithm. In PEG, node degrees are distributed according to some performance criteria before edges are added. The algorithm builds a Tanner graph by connecting the graph's nodes edge by edge provided that the added edge has minimal impact on the girth of the graph. With this algorithm regular and irregular codes can be obtained with optimized performance. Codes obtained using this method are amongst the best performance codes at short lengths with column-weight of at least three. However, as with codes obtained using the BF algorithm, PEG codes are not easily implementable due to their pseudorandom interconnections.

4. PROPOSED ALGORITHM

There are different methods for constructing quasi-cyclic LDPC codes including algebraic and combinatorial, examples of which are found in [4, 14]. These construction methods avoid four cycles by employing the row-column constraint. Although these methods can be used to construct a wide range of codes, they have limited ability to produce codes with arbitrary girth, rate, and length.

We take advantage of the flexibility found in random search methods such as BF and PEG to construct a wide range of structured codes. We add further constraints to search algorithms such that the obtained codes are quasi-cyclic. This is achieved by dividing rows of a code into equal groups to form submatrices. Rows representing vertices are used to form a distance graph in which two vertices are connected if they are in different groups. Vertices in a group are connected in a sequential order to obtain cyclically shifted

Row connections

1.5	1.6	1.7	1.8
2.6	2.7	2.8	2.5
3.7	3.8	3.5	3.6
4.8	4.5	4.6	4.7

(a)

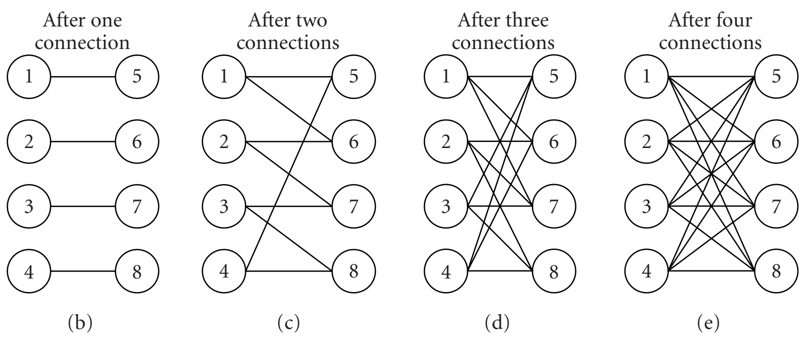

Figure 2: Graph representation of a (16, 2, 4) code with girth eight.

```
(1) Divide rows into $j'$ equal groups of size
    $p$, $(RG_1, \ldots, RG_{j'})$. If the number of rows
    is unknown or not given, start with a
    theoretical minimum number of rows if
    known otherwise start with a group size
    of $k$ (row-weight).
    $r_x$ is row $x$.
    $\bigcup_{r_x}$ is a set of rows within a distance of $g$
    from $r_x$.
(2) Pair row groups such that each group
    appears $k$ times. There are $kj'/2$ row group
    pairs, $(RGP_1, \ldots, RGP_{kj'/2})$.
(3) For $t = 1$ to $\frac{kj'}{2}$ {
            $RG_{\text{ref}} = RGP_t(1)$
            select $r_i \in RG_{\text{ref}}$
            sequentially or randomly search for
            $r_x \in RGP_t(2)$ where $r_x \notin \bigcup_{r_i}$, else
            algorithm fails
            For $z = 1$ to $p$ {
                $r_{i+z}$ is connected to $r_{x+z}$ if
                $r_{x+z} \notin \bigcup_{r_{i+z}}$, else algorithm
                fails
            }
    }
(4) Use obtained distance graph to form a
    LDPC parity-check matrix.
```

Algorithm 1

1				1				1				1			
	1				1				1				1		
		1				1				1				1	
			1				1				1				1
1							1			1			1		
	1			1							1			1	
		1			1			1							1
			1			1			1			1			

Figure 3: Matrix representation of a (16, 2, 4) code with girth eight.

identity submatrices. That is, if vertices v_x and v_y are connected, then v_{x+a} and v_{y+a} are also connected. A desired girth, g, is achieved by randomly or sequentially selecting and connecting vertices that are at a desired distance from each other. The resulting graph is then used to form an equivalent LDPC code matrix as was done in Figure 1. Algorithm 1 is described with rows representing vertices.

The algorithm constructs a distance graph code rows in step 3 of algorithm. Row r_x, which is at a distance of at least g from r_i, is searched sequentially or randomly in $RGP_t(2)$. A sequential search traverses a row group in ascending or descending order. If r_x is found, the rest of the rows are connected relative to r_i and r_x if the girth condition is not violated. Figure 2 shows row connections for a (16, 2, 4) LDPC code with girth eight constructed using the proposed algorithm. There are two groups of size 4. The first group and row 1 are always chosen as the reference group and row, respectively. A sequential search is used in group 2. The first group has rows 1 to 4 and the second group has rows 5 to 8. Since there are only two groups (group 1 and 2), the groups pairings are [1 2], [1 2], [1 2], [1 2] with each group appearing four times (desired row-weight). In the first connection row 5 is found to satisfy the distance of four (desired girth) from row 1. The rest of group 1, rows 2 to 4, are then connected to rows 6 to 8. In the second connection, row 6 is the first to satisfy the distance. It is connected to row 1 with the rest of group 1 connected to the rest of group 2. The process is repeated in connections three and four as shown in the figure. The row connections form a distance graph with the number of vertices equal to eight, a vertex degree equal to four and a girth of four. Figure 3 shows a matrix representation of the obtained code. Each set of connections forms a column group with each row group as a 4×4 submatrix. Since the first group is not searched or shifted, rows in this group are connected in their natural order in each submatrix. The top four rows contain four unshifted identity submatrices corresponding to four connections for group 1 rows. Group 2 rows are connected in their natural order only in the first connection. The bottom 4×4 submatrices represent group 2 connections.

$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}_x$	$\mathbf{I}_x$	$\mathbf{I}_x$

(a)

$\mathbf{I}$	$\mathbf{I}$	$\mathbf{O}$	$\mathbf{I}$	$\mathbf{I}$	$\mathbf{O}$
$\mathbf{I}_x$	$\mathbf{O}$	$\mathbf{I}$	$\mathbf{O}$	$\mathbf{I}_x$	$\mathbf{I}$
$\mathbf{O}$	$\mathbf{I}_x$	$\mathbf{I}_x$	$\mathbf{I}_x$	$\mathbf{O}$	$\mathbf{I}_x$

(b)

Figure 4: Structure of obtained quasi-cyclic LDPC codes.

When two row groups are used, obtained codes will have a structure as shown in Figure 4(a). $\mathbf{I}$ is a $p \times p$ identity submatrix and $\mathbf{I}_x$ is a shifted $p \times p$ identity submatrix. When more than two groups are used, codes with zero submatrices are obtained as the example in Figure 4(b) illustrates, where $\mathbf{O}$ is a $p \times p$ zero submatrix. The first row group and column-group submatrices would be shifted if the reference row (r_i) is chosen randomly.

The complexity of the algorithm is analyzed in terms of the number of rows, M, and the number of row groups as follows.

(i) If the number of row groups is j', each group is of size M/j'.

(ii) Updating neighbors of a row at a distance of g takes g cycles (or operations). For a single row group it takes gM/j' cycles. For each pair of row groups rows from the two groups are connected if they do not violate the girth condition. Checking the condition for all the connections takes another M/j' cycles. Hence, a single row group pair takes $M(g+1)/j'$ cycles.

(iii) The connection process is repeated for each row group pairing. There are $kj'/2$ group pairings for regular code with row-weight of k. Therefore, it takes $kM(g+1)/2$ cycles to complete all connections. This is assuming that the group size is large enough for the algorithm to form all connections and does not include the number of extra tries in case the connections failed the girth condition. The complexity of this algorithm is therefore $\mathbf{O}(M)$.

The actual complexity may also depend on how the algorithm is implemented. In [12] set algebra is used to eliminate rows that are too close to the current or reference row. With this approach, the complexity depends on how fast the neighbors of each row are updated. The algorithm fails if the set of rows satisfying the distance from the current row is empty. If it is not empty, the rows in the set could be chosen randomly, in sequential order or using other criteria. Sequential searching results in the same code, as the found rows will be the same assuming the reference group and rows are the same. Random searches will result in a variety of codes. Figure 5 shows row connections for two girth-eight codes obtained by sequential and random searches. When a sequential search is used, the second group is shifted by one with each connection. With random search, both groups are shifted randomly observing the condition that they are not shifted by the same amount (avoid 4-cycles). The codes obtained may have different minimum distances as shown by Fossorier in [4]. Hence, random searches would generally result in better performing codes as was confirmed by simulations shown later in this section.

1.8	1.9	1.1	1.11	1.12	1.13	1.14
2.9	2.1	2.11	2.12	2.13	2.14	2.8
3.1	3.11	3.12	3.13	3.14	3.8	3.9
4.11	4.12	4.13	4.14	4.8	4.9	4.1
5.12	2.13	3.11	7.13	3.12	1.11	4.9
6.13	3.14	4.12	8.14	4.13	2.12	5.1
7.14	4.8	5.13	2.8	5.14	3.13	6.11

(a)

1.8	5.9	6.14	3.9	6.8	4.14	7.12
2.9	6.1	7.8	4.1	7.9	5,8	1.13
3.1	7.11	1.9	5.11	1.1	6.9	2.14
4.11	1.12	2.1	6.12	2.11	7.1	3.8
5.12	2.13	3.11	7.13	3.12	1.11	4.9
6.13	3.14	4.12	8.14	4.13	2.12	5.1
7.14	4.8	5.13	2.8	5.14	3.13	6.11

(b)

Figure 5: Girth-eight row connections using (a) sequential search (b) random search.

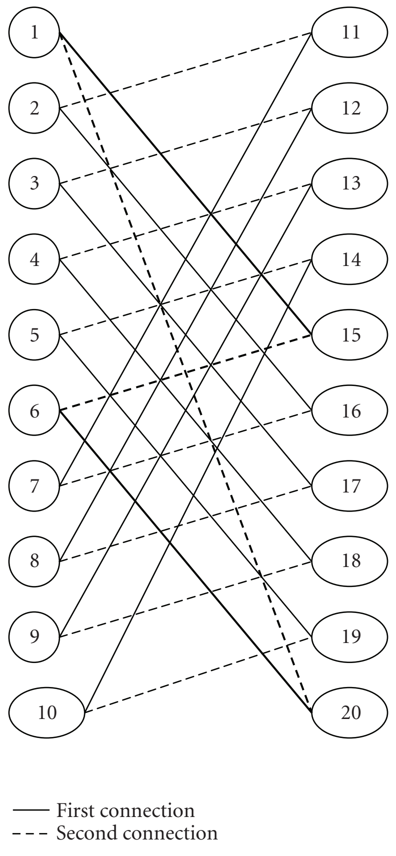

Figure 6: Formation of smaller cycles than the target girth.

The proposed algorithm does not guarantee higher girths larger than six or eight. The fact that no two rows are connected more than once guarantees a girth of six. If only two row groups are used, girth eight is guaranteed. For higher girths the algorithm checks if connecting the rest of row groups relative to the reference row does not violate the girth. Figure 6 shows how a target girth of twelve could be violated when the rest of group rows are connected. The solid and dotted lines represent the first and second connections, respectively. In the second connection try, row 20 is found to satisfy the length of at least six from reference row 1. However, connecting the rest of group 1 rows creates cycles of length four. An example of such a cycle is between rows 1, 15, 6, 20. In this case the algorithm tries another connection. From our experiments the algorithm does not take long to find connections that hold the desired girth especially for large group sizes.

4.1. Girth-eight codes

When only two row groups are used the codes obtained have a minimum girth of eight. The row groups form a bipartite graph. A bipartite graph has a minimum cycle length of four. It also has even cycle lengths. Therefore, only girths of eight and twelve can be obtained. Quasi-cyclic codes with the number of row groups equal to the column-weight have a maximum girth of twelve [4].

Obtained girth-eight codes have a minimum size of $2k \times k^2$ with a group size of k for all row-weights of k. This is the minimum size that could be obtained as it corresponds to the size of a cage graph of distance four for a given vertex degree [8]. Also, the minimum size of a row group is of size k as each row has to be connected to k different rows. Larger codes can be obtained by using larger group sizes. When the group size is larger than k, the row groups still form a bipartite graph resulting in a minimum cycle of four.

4.2. Girth-twelve codes

Girth-twelve codes are formed by a bipartite graph with a smallest cycle length of six when two row groups are used. As with girth-eight codes, girth-twelve codes could be constructed from cage graphs. For codes with $(k-1)$ as a prime, derived girth-twelve codes from cages are of size $k(k^2-k+1)$ [3, 8]. Construction of codes based on cages could only be done with known cages.

Using the proposed algorithm, girth-twelve LDPC codes could be constructed for any row-weight. Table 1 shows code and row group sizes obtained using a sequential search. Obtained codes are about twice the size of codes derived from cage graphs in some cases. Figure 7 shows row connections for two girth-twelve codes. Part (a) is a (60, 2, 4) code with a group size of 15. In part (b) a larger group size of 20 is used to construct an (80, 2, 4) code. The same amount of shifts are obtained in both cases in the second row group. From our experiments we observed that group sizes larger than those found in Table 1 hold the girth. Larger codes can therefore be constructed by using larger row groups. However, we could not prove that all larger groups maintain the girth of twelve. Table 2 shows code sizes obtained using a random search. Random searches may result in smaller codes as in Table 2. However, obtaining smaller codes generally requires many tries to get the right combination of shifts.

Table 1: Smallest girth-twelve code sizes obtained with two groups and a sequential search.

k	Min. group size	Code size
3	7	14×21
4	15	30×60
5	25	50×125
6	35	70×210
7	61	122×427
8	77	154×616
9	119	238×1071
10	134	268×1340
11	174	348×1914
12	216	432×2592
13	251	502×3263
14	304	608×4256
15	390	780×5850
16	509	1018×8144
17	615	1230×10455
18	663	1326×11934

4.3. Girths larger than twelve

Codes with higher girths were obtained by using a number of row groups larger than two. It was proved algebraically in [4] that quasi-cyclic codes formed with the number of row and column groups equal to row- and column-weights, respectively, have a maximum girth of twelve. A larger number of row groups than row-weights is used here to search for codes with girths larger than twelve. Figure 8 shows row division and row group pairing for a girth-sixteen code. There are 162 rows and three row groups. The three groups are paired as [1 2], [1 2], [1 3], [1 3], [2 3], and [2 3] with each group appearing four times. Connected rows are separated by a period. Irregular codes could be constructed by having different number of appearances of the groups. The column-weight will still be two but the row-weights will be equal to the number of times the group appears in the pairings. For example, row group pairings of [1 2], [1 2], [1 3], [1 3], and [2 3] for the example code will result in a code with an average row-weight of 10/3 and rate of 0.4. Rows in row groups 2 and 3 will have three connections only. Table 3 shows code sizes for some obtained codes with girths higher than twelve. The sizes and girths of obtained codes may differ depending on the number and combination of groups. The number of groups and group combinations used here were chosen arbitrarily.

1.16	1.17	1.19	1.23
2.17	2.18	2.2	2.24
3.18	3.19	3.21	3.25
4.19	4.2	4.22	4.26
5.2	5.21	5.23	5.27
6.21	6.22	6.24	6.28
7.22	7.23	7.25	7.29
8.23	8.24	8.26	8.3
9.24	9.25	9.27	9.16
10.25	10.26	10.28	10.17
11.26	11.27	11.29	11.18
12.27	12.28	12.3	12.19
13.28	13.29	13.16	13.2
14.29	14.3	14.17	14.21
15.3	15.16	15.18	15.22

(a)

1.21	1.22	1.24	1.28
2.22	2.23	2.25	2.29
3.23	3.24	3.26	3.3
4.24	4.25	4.27	4.31
5.25	5.26	5.28	5.32
6.26	6.27	6.29	6.33
7.27	7.28	7.3	7.34
8.28	8.29	8.31	8.35
9.29	9.3	9.32	9.36
10.3	10.31	10.33	10.37
11, 31	11.32	11.34	11.38
12.32	12.33	12.35	12.39
13.33	13.34	13.36	13.4
14.34	14.35	14.37	14.21
15.35	15.36	15.38	15.22
16.36	16.37	16.39	16.23
17.37	17.38	17.4	17.24
18.38	18.39	18.21	18.25
19.39	19.4	19.22	19.26
20.4	20.21	20.23	20.27

(b)

Figure 7: Row connections for girth-twelve LDPC codes.

Table 2: $(N, 2, k)$ girth-twelve codes using two groups and a random search.

k	Min. group size	Code size
3	7	14×21
4	13	26×52
5	21	42×105
6	31	62×186
7	53	106×371
8	67	134×536
9	105	210×945
10	125	250×1250

Group pairs $\longrightarrow$	[1 2]	[1 2]	[1 3]	[1 3]	[2 3]	[2 3]
	1.55	1.56	1.109	1.112	55.119	55.134
	2.56	2.57	2.110	2.113	56.120	56.135
Row connections $\longrightarrow$	3.57	3.58	3.111	3.114	57.121	57.136
	4.58	4.59	4.112	4.115	58.122	58.137
	⋮	⋮	⋮	⋮	⋮	⋮
	54.108	54.55	54.162	54.111	108.117	108.133

Figure 8: Group row connections forming girth-sixteen LDPC code with row-weight of 4.

4.4. Performance simulations

Bit error rate (BER) performances of constructed codes were simulated on an AWGN channel with BPSK modulation. Performance curves are shown in Figure 9. Simulated codes are all of size (2556, 2, 4). Four points are noted from these curves. Firstly, randomly shifted codes perform better than sequentially shifted codes. The two seq-(2556, 2, 4) and ran-(2556, 2, 4) codes in the figure have sequential and random shifts, respectively, from two row groups. They have a girth and average girth of twelve. However, the randomly shifted code outperforms the sequentially shifted code by about 0.4 dB at 10^{-5} BER. Secondly, multilevel or multidivision codes perform better than those with two groups. The multilevel code used here was constructed with six groups. It has a girth and average girth of twelve as the other sequentially shifted code. It, however, performs better by about 0.4 dB at 10^{-5} BER. This confirms the results obtained in [15] showing that multidivision codes have better performance compared to those with fewer divisions. However, codes in [15] are of a different structure (sub-matrix shift values and arrangement). Thirdly, performance curves show larger girth codes performing better. A girth-twenty code also from six row groups with sequential shifts outperforms the girth-twelve code by 1 dB at 10^{-5} BER. Lastly, performance curves also show a random code outperforming girth-twelve codes by about 0.7 dB. The random was constructed using a slightly modified bit-filling algorithm to obtain a regular code. It has a girth of ten and average girth of 14.6. It

Table 3: Code sizes of girths larger than twelve using a sequential search.

k	No. of groups	Min. group	Code size	Girth
3	4	9	36×54	14
3	4	16	64×96	16
3	4	17	68×102	18
3	4	18	72×108	20
4	3	35	105×210	14
4	3	54	162×324	16
4	3	69	207×414	18
4	8	390	3120×6240	24
4	6	213	1278×2556	20
5	4	65	260×650	14
5	4	112	448×1120	16
6	6	121	726×2178	14
6	12	108	1286×3888	16

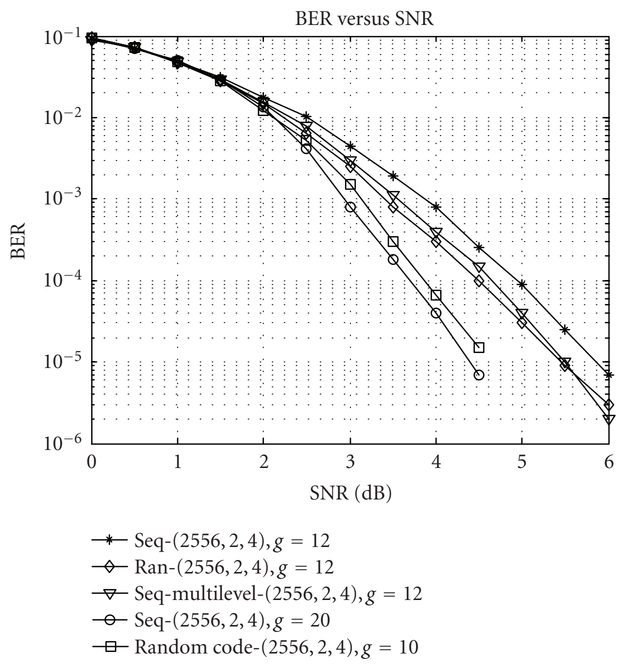

Figure 9: BER performance of obtained codes with 35 iterations.

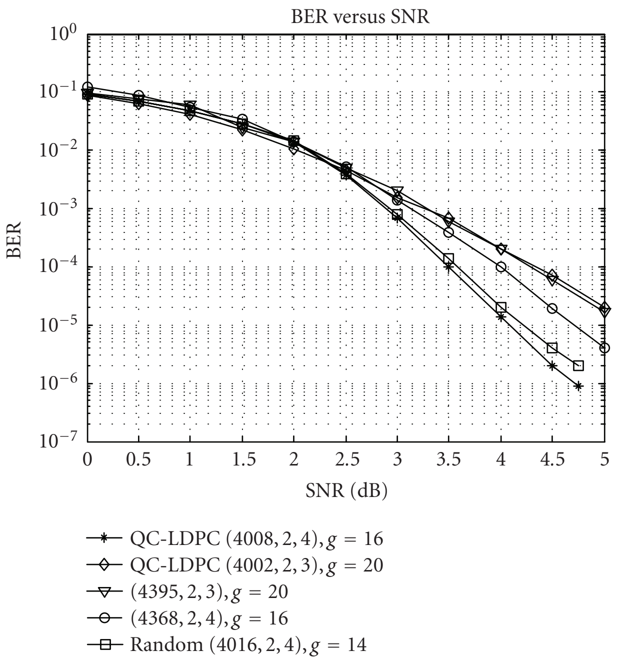

Figure 10: BER performance of larger codes compared to graphical codes with 35 iterations.

outperforms girth-twelve codes by about 0.7 dB. However, quasi-cyclic codes offer the best performance and hardware complexity tradeoff.

Figure 10 shows larger codes compared to codes obtained using graphical models in [7] and a random code with a high girth of 14. The row-weight-three codes have the same performances. The obtained QC-LDPC code with row-weight of four outperforms the graphical code by about 0.6 dB at 10^{-5} BER and outperforms the random by about 0.1 dB at 10^{-5} BER.

4.5. Hardware implementation

Codes obtained using the original BF or PEG algorithms are not easily implementable. They have an unstructured row-column connection because of the random selection of connections. Random codes are not easily implementable in hardware since there is no general rule(s) to describe row-column connections. Connections are therefore hardwired or stored in lookup tables. Hardwired interconnections are inflexible whereas lookup tables require a large amount of memory.

Codes derived from cage graphs have some structure in that the connections can be described algebraically in most graphs [16]. However, connections may vary from vertex to vertex and from graph to graph. All connection rules need to be stored in hardware. Quasi-cyclic LDPC codes are one type of codes in which a group of rows or columns has similar connections defined by shifts. These codes can be implemented by mapping a row or column group to one processing node. Addressing of messages within processing nodes is accomplished by memory shifts or offsets corresponding to the cyclic structure of the matrix. Examples of quasi-cyclic LDPC decoder architectures can be found in [10, 17]. Variable and check node computations can be overlapped for quasi-cyclic LDPC codes reducing the decoding time by up to half [17]. Encoder implementations could also be simplified by taking advantage of the quasi-cyclic structure as in [9].

Another advantage of the proposed algorithm over other methods is that it could be used to construct codes for any group configuration. The code construction is the same regardless of the submatrix arrangement. Quasi-cyclic LDPC code submatrix configuration could be optimized for BER performance or designed to map a given hardware architecture. For example, in [18] simulated annealing is used to find a configuration giving the best BER performance. A decoder architecture could then be designed based on that configuration.

5. CONCLUSIONS

A nonalgebraic search algorithm for constructing quasi-cyclic LDPC codes of column-weight two has been introduced. Rows of a code are divided into groups from which a distance graph is formed. Rows are connected in the graph if they are separated by a desired distance to obtained a desired girth. A sequential or random search could be used to find rows satisfying the distance condition. Although the algorithm does not guarantee girths larger than eight, larger girths were easily obtained from experiments. The algorithm obtains a wide range of codes in terms of girths, rate, and lengths. The algorithm is also efficient with a computational complexity linear in the number of rows. Randomly shifted codes perform better than sequentially shifted codes. Also more row groups result in better codes compared to codes from two row groups. Although the performance of obtained quasi-cyclic codes does not match that of random codes of the same size and girth, they are easier to implement in hardware.

REFERENCES

[1] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[2] H. Song, J. Liu, and B. V. K. V. Kumar, "Low complexity LDPC codes for partial response channels," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '02)*, vol. 2, pp. 1294–1299, Taipei, Taiwan, November 2002.

[3] H. Song, J. Liu, and B. V. K. V. Kumar, "Large girth cycle codes for partial response channels," *IEEE Transactions on Magnetics*, vol. 40, no. 4, part 2, pp. 3084–3086, 2004.

[4] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1788–1793, 2004.

[5] M. E. O'Sullivan, "Algebraic construction of sparse matrices with large girth," *IEEE Transactions on Information Theory*, vol. 52, no. 2, pp. 718–727, 2006.

[6] Y. Mao and A. H. Banihasherni, "A heuristic search for good low-density parity-check codes at short block lengths," in *Proceedings of IEEE International Conference on Communications (ICC'01)*, vol. 1, pp. 41–44, Helsinki, Finland, June 2001.

[7] H. Zhang and J. M. Moura, "The design of structured regular LDPC codes with large girth," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '03)*, vol. 7, pp. 4022–4027, San Francisco, Calif, USA, December 2003.

[8] G. Malema and M. Liebelt, "Low-complexity LDPC codes for magnetic recordings," in *Proceedings of International Enformatika Conference (IEC '05)*, vol. 5, pp. 269–271, Prague, Czech Republic, August 2005.

[9] H. Fujita and K. Sakaniwa, "Some classes of quasi-cyclic LDPC codes: properties and efficient encoding method," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E88-A, no. 12, pp. 3627–3635, 2005.

[10] S. Olçer, "Decoder architecture for array-code-based LDPC codes," in *Proceedings of IEEE Global Telecommunications Conference (GLOBECOM '03)*, vol. 4, pp. 2046–2050, San Francisco, Calif, USA, December 2003.

[11] J. Xu, L. Chen, L. Zeng, L. Lan, and S. Lin, "Construction of low-density parity-check codes by superposition," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 243–251, 2005.

[12] J. Campello, D. S. Modha, and S. Rajagopalan, "Designing LDPC codes using bit-filling," in *Proceedings of IEEE International Conference on Communications (ICC '01)*, vol. 1, pp. 55–59, Helsinki, Finland, June 2001.

[13] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 386–398, 2005.

[14] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Transactions on Information Theory*, vol. 47, no. 7, pp. 2711–2736, 2001.

[15] R. Bresnan, "Novel code construction and decoding techniques for LDPC codes," M.Eng.Sc. thesis, Department of Electrical and Electronic Engineering, University College Cork, Cork, Ireland, 2004.

[16] M. Meringer, "Fast generation of regular graphs and construction of cages," *Journal of Graph Theory*, vol. 30, no. 2, pp. 137–146, 1999.

[17] Y. Chen and K. K. Parhi, "Overlapped message passing for quasi-cyclic low-density parity check codes," *IEEE Transactions on Circuits and Systems*, vol. 51, no. 6, pp. 1106–1113, 2004.

[18] J. Thorpe, K. Andrews, and S. Dolinar, "Methodologies for designing LDPC codes using protographs and circulants," in *Proceedings of IEEE International Symposium on Information Theory*, pp. 238–242, Chicago, Ill, USA, June-July 2004.

Gabofetswe Malema obtained the B.S. degree in computer engineering (1997) and the M.S. degree in electrical engineering and computer science (1999) from Valparaiso University and The University of Illinois at Chicago, respectively. He worked as a Lecturer at Department of Computer Science, the University of Botswana, from 2000–2003. He is currently in his final year of Ph.D. study at School of Electrical and Electronic Engineering, the University of Adelaide. His main research interests are LDPC construction and hardware implementation and processor-in-memory architectures.

Michael Liebelt obtained B.S. degree in computer science and applied maths (1978), the B.E. degree (with honors) in electrical and electronic engineering (1979), and the M.E. degree in electrical and electronic engineering (1982) all from the University of Adelaide, Australia. His research interests are computer architecture, asynchronous digital systems and test methods and design for testability. He is currently an Associate Professor and Head of School of Electrical and Electronic Engineering at The University of Adelaide, Australia.