Construction of High-Girth QC-LDPC Codes

Yige Wang, Jonathan Yedidia, Stark Draper

TR2008-061 September 2008

Abstract

We describe a hill-climbing algorithm that constructs high-girth quasi-cyclic low density parity check (QC-LDPC) codes. Given a desired girth, the algorithm can find QC-LDPC codes of shorter block-length in much less time compared with the previously proposed ”guess-and-test” algorithm. An analysis is also provided to explain when guess-and-test would be expected to perform well or badly.

5th International Symposium on Turbo Codes and Related Topics

Construction of High-Girth QC-LDPC Codes

Yige Wang[†‡], Jonathan S. Yedidia[†], Stark C. Draper[*]
[‡] University of Hawaii at Manoa, Honolulu, HI, yige@hawaii.edu
[†] Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA, yedidia@merl.com
[*] Dept. of ECE, University of Wisconsin, Madison, WI 53706, USA, sdraper@ece.wisc.edu

Abstract— **We describe a hill-climbing algorithm that constructs high-girth quasi-cyclic low-density parity check (QC-LDPC) codes. Given a desired girth, the algorithm can find QC-LDPC codes of shorter block-length in much less time compared with the previously proposed "guess-and-test" algorithm. An analysis is also provided to explain when guess-and-test would be expected to perform well or badly.**

I. Introduction

Two broad classes of methods have emerged for the construction of low-density parity-check (LDPC) codes [1]. One set of methods is based on random constructions, the second on algebraic constructions. Random constructions (see, e.g., [2]-[5]) can produce LDPC codes that closely approach the Shannon capacity. However, random constructions are not easy to implement in hardware as the randomly designed connections between variable and check nodes inevitably result in significant encoding and decoding complexity. On the other hand, algebraic constructions yield structures that are strongly preferred in hardware implementations.

Quasi-cyclic LDPC (QC-LDPC) codes are a particularly important class of algebraically constructed LDPC codes. They are featured in a variety of communications system standards, such as IEEE 802.16e [6], DVB-S2 [7] and 802.11 [8]. QC-LDPC codes have been constructed based on finite geometries [9] or circulant permutation matrices [10]-[14].

Depending on the application, LDPC codes are designed to optimize performance in either the "water-fall" (SNR near the code threshold) or "error-floor" (higher SNR) regime, or both. Low error floors are particularly important for applications that have extreme reliability demands, including magnetic recording and fiber-optic communication systems.

Error floor issues for LDPC codes are investigated in [15], [16], which characterize error events using "trapping sets." Trapping sets result from clusters of short cycles in the code's Tanner graph. One way to remove trapping sets that involve short cycles is to carefully design the clustering of short cycles in the code graph. An alternate, and at least conceptually simpler, approach is to design codes with larger girths – the "girth" of a code is the length of the shortest cycle in the code graph. By removing short cycles, we remove large swaths of clusters of cycles and, at one fell swoop, hopefully lower the error floor. Motivated by this idea, in this paper, we focus on the problem of optimizing the girth of QC-LDPC codes [1].

[1]Our initial experiments indeed show that girth-10 codes have a lower error floor compared with girth-6 and girth-8 codes of the same length and rate. This result, and other details about the codes found using our algorithm, will be discussed in depth in a future paper [19].

There is considerable work on optimizing girth in LDPC codes. In [17], a progressive-edge growth (PEG) algorithm is proposed for random LDPC codes. The case of QC-LDPC codes is studied in [18], where high-girth QC-LDPC codes were obtained using a random "guess-and-test" algorithm.

The trouble with guess-and-test is that it is quite time-consuming. In this paper, we propose a hill-climbing search algorithm that greedily adjusts an initial QC-LDPC code to find a code of short length that meets the specified code and girth parameters. Given a set of parameters, the algorithm finds QC-LDPC codes of shorter length and in much less time when compared to guess-and-test. The improvement is quite significant for QC-LDPC codes with large base matrices.

The rest of the paper is organized as follows. In Section II, the necessary theory for identifying cycles in a QC-LDPC code is presented, and the guess-and-test algorithm is detailed. The hill-climbing algorithm is presented in Section III. The comparison of guess-and-test and hill-climbing is presented in Section IV, and in Section V we provide an analysis explaining in more detail when the guess-and-test algorithm can be expected to succeed or fail.

II. Cycles in QC-LDPC codes

A (J, L) regular QC-LDPC code of length N is defined by a parity check matrix

$$H = \begin{bmatrix} I(0) & I(0) & \cdots & I(0) \\ I(0) & I(p_{1,1}) & \cdots & I(p_{1,L-1}) \\ \vdots & & \ddots & \vdots \\ I(0) & I(p_{J-1,1}) & \cdots & I(p_{J-1,L-1}) \end{bmatrix} \quad (1)$$

where $1 \leq j \leq J-1$, $1 \leq l \leq L-1$, and $I(p_{j,l})$ represents the $p \times p$ circulant permutation matrix obtained by cyclically right-shifting the $p \times p$ identity matrix $I(0)$ by $p_{j,l}$ positions, with $p = N/L$. For a specific QC-LDPC code we define the corresponding "base matrix" as the matrix of circulant shifts that defines the QC-LDPC code:

$$B = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & p_{1,1} & \cdots & p_{1,L-1} \\ \vdots & & \ddots & \vdots \\ 0 & p_{J-1,1} & \cdots & p_{J-1,L-1} \end{bmatrix}. \quad (2)$$

Note that we have placed zeroes in the entire first row and column of the base matrix. This form results in no loss of generality as a base matrix with non-zero entries in the first row or column can easily be converted into an equivalent code of our form without changing the Tanner graph of the underlying code.

To understand how one can find cycles in a graph given a base matrix, consider Figure 1. In the figure we show a parity check matrix H from which we focus on four 3×3 circulant permutation matrices (in black) with associated parameters p_1, p_2, p_3, and p_4. Two choices for the parameters of these four matrices are shown in the subfigures: $p_1 = 0$, $p_2 = 1$, $p_3 = 2$ and $p_4 = 1$ on the left and $p_1 = 0$, $p_2 = p_3 = p_4 = 1$ on the right. The first set of choices results in cycles of length four, while the latter results in a cycle of length 12. The cycles are also shown.

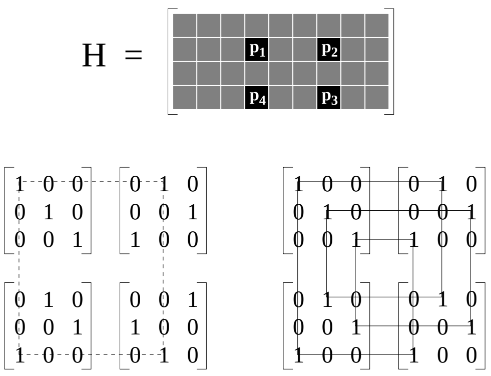

Fig. 1. A parity-check matrix and four $p \times p$ circulant permutation matrices ($I(p_1)$, $I(p_2)$, $I(p_3)$ and $I(p_4)$) selected from it. One set of parameters (lower left, $p_1 = 0$, $p_2 = 1$, $p_3 = 2$, $p_4 = 1$) results in a cycle of length four. An alternate set (lower right, $p_1 = 0$, $p_2 = p_3 = p_4 = 1$) results in a cycle of length twelve.

We now discuss the correspondence between a choice of shifts (the $p_{j,l}$) and the length of a resulting cycle. Recall that each row of a parity-check matrix corresponds to a check node and each column to a bit node. Cycles correspond to a path through nodes, alternating between check and bit nodes. In terms of the parity-check matrix a path through nodes can be visualized as rectilinear moves as depicted in Fig. 1. A horizontal move (along a row) corresponds to choosing two edges connected to the same parity-check that form part of the path. A vertical move (along a column) corresponds to choosing a second edge connected to the same bit node that will form the next step in the path.

Now, for a cycle to exist the path must end at the same bit node it started from. It is necessary (but not sufficient) for the path when viewed at the base matrix level to form a cycle (i.e., there must be a cycle in B, cf. (2)). However, since each circulant permutation matrix corresponds to p parity and p variable nodes this is not sufficient. The path could end up at a different bit node in the same circulant matrix, thus not completing a cycle. What is sufficient is if, when the path returns, it returns to the same *column* of the circulant matrix that it started from. E.g., in the left-hand example of Fig. 1, this happens for a cycle of length four. However, with the slightly different choice of circulant shifts of the right-hand example, this only happens after a cycle of length 12.

We can now specify the conditions on the $p_{j,l}$ that result in a cycle. Calculate the differences between the $p_{j,l}$ for neighboring permutation matrices along a given path, where neighbors are on the same row. E.g., in Fig. 1) these would be $p_2 - p_1$ and $p_4 - p_3$. (We could alternately and equivalently think in terms of difference along columns.) Each difference corresponds to the shift in what column (i.e., what variable node) of the permutation matrix the path passes through. Only if the differences sum to zero (mod-p) at the end of the path will the path return to the same variable node in the starting permutation matrix, thereby defining a cycle. For the example of Fig. 1 for a length-four cycle to exist the condition is:

$$p_1 - p_2 + p_3 - p_4 \mod p = 0, \tag{3}$$

which is satisfied for $p_1 = 0$, $p_2 = 1$, $p_3 = 2$, $p_4 = 1$, but is not satisfied by $p_1 = 0$, $p_2 = p_3 = p_4 = 1$. In passing we note that each cycle in a QC-LDPC code is necessarily related to $p - 1$ other cycles obtained by the $p - 1$ possible cyclic shifts in the circulant matrices, although this fact will be of no consequence in the following.

The same logic extends to longer cycles. Just as a four-cycle must pass through four elements of the base matrix arranged in a rectangle, an arbitrary cycle of length $2i$ in the Tanner graph of the code must pass through $2i$ elements of the base matrix denoted by the ordered series

$$(j_0, l_0), (j_1, l_0), (j_1, l_1), \cdots, (j_{i-1}, l_{i-1}), (j_0, l_{i-1}) \tag{4}$$

where for $1 \leq k < i$, $j_k \neq j_{k-1}$, $l_k \neq l_{k-1}$, $j_{i-1} \neq j_0$, and $l_{i-1} \neq l_0$. This ordered series can be considered a "potential" cycle of length $2i$; it will only actually correspond to a cycle if the base matrix elements traversed satisfy the generalization of equation (3). To define this generalization we use the notation introduced by Fossorier [18] who summarizes these ideas and formulates the conditions for which a length $2i$ cycle will exist for a particular base matrix. Define $\Delta_{j_x,j_y}(l) = p_{j_x,l} - p_{j_y,l}$. Theorem 2.1 of [18] shows that a necessary and sufficient condition for the code to have girth of at least $2(i + 1)$ is that

$$\sum_{k=0}^{m-1} \Delta_{j_k,j_{k+1}}(l_k) \neq 0 \mod p \tag{5}$$

for all m, $2 \leq m \leq i$, all j_k, $0 \leq j_k \leq J - 1$, all j_{k+1}, $0 \leq j_{k+1} \leq J - 1$, and all l_k, $0 \leq l_k \leq L - 1$, with $j_0 = j_m$, $j_k \neq j_{k+1}$, and $l_k \neq l_{k+1}$.

For every pair (J, L) and desired girth g (minimum-length cycle of graph) there must exist a p_{min} (or equivalently N_{min}) such that when $p < p_{min}$ or $N < N_{min}$, no parity check matrices exists that satisfies (5). It is shown in [18] that a necessary condition for girth $g \geq 6$ is $p \geq L$ if L is odd and $p \geq L + 1$ if L is even. For girth $g \geq 8$, a necessary condition is $p > (J - 1)(L - 1)$. However, these conditions give only very loose lower bounds on the actual minimal value p_{min} that can give rise to a code of a given girth.

We want an algorithm that given a pair (J, L) and a desired girth g returns a base matrix and a value of p such that the specified code has the desired girth and p was equal to p_{min}, or at least as close as possible to p_{min}. Fossorier [18] suggests a "guess-and-test" algorithm: $(J - 1)(L - 1)$ integers between 0 and $p - 1$ are chosen randomly uniform and independent identically distributed for the non-zero elements of the base matrix until a set is found such that (5) is satisfied. He also shows that for codes of girth 8 or larger, all non-zero values in the base matrix must be distinct. We consider his algorithm for those girths to be one where the $(J-1)(L-1)$ integers are chosen randomly in the range from 1 to $p-1$, such that all the integers are distinct.

The problem with guess-and-test is that it is time-consuming, especially for large base matrices and N is close to N_{min}. We have found that for largebase matrices and even given considerable searching time, the smallest N we can typically find (in a reasonable time) using this algorithm is far larger than N_{min}. We know this because for the same choice of (J, L) and girth g our algorithm typically finds a much smaller N in less time than we allow for the guess-and-test approach. We next present our algorithm. It is a hill-climbing algorithm that sequentially adjusts the base matrix in a greedy manner to rid it of short cycles.

III. Hill-climbing searching algorithm

The general idea of our hill-climbing algorithm is as follows. We start with a randomly chosen base matrix. We then iteratively change the base matrix by making a "move" changing a single element (a $p_{j,l}$) to another value. We select the move to make the greatest reduction in a cost function. The cost is a function of the number of cycles of length less than the desired girth that remain in the code graph. We further weight shorter cycles to be more costly than longer cycles. When we no longer can change any single value of the base matrix to a value that further reduces the cost (and thus the number of undesired cycles), the algorithm terminates. The algorithm is a local hill-climbing algorithm where the objective function is the weighted sum of undesired cycles, and local moves are changes in a single element of the base matrix to another value.

The main challenge in implementing this algorithm lies in book-keeping: tracking how many cycles of each length the current code contains, and what will be the resulting number of cycles if we change each possible element in the base matrix to each other possible value. The calculation becomes particularly involved when one searches for codes of girth 10 (which is the highest girth value for which we have so far implemented our algorithm), because of the many possible ways that eight-cycles can form in the graph.

We now define the cost matrix, which tracks the cost (in terms of the weighted sum of the number of cycles) of changing any element in the base matrix to have any other possible value. Thus, for any parity check matrix B defined in (2), there exists a corresponding cost matrix

$$C = \begin{bmatrix} \boldsymbol{c}_{0,0} & \boldsymbol{c}_{0,1} & \cdots & \boldsymbol{c}_{0,L-1} \\ \boldsymbol{c}_{1,0} & \boldsymbol{c}_{1,1} & \cdots & \boldsymbol{c}_{1,L-1} \\ \vdots & & \ddots & \vdots \\ \boldsymbol{c}_{J-1,0} & \boldsymbol{c}_{J-1,1} & \cdots & \boldsymbol{c}_{J-1,L-1} \end{bmatrix} \quad (6)$$

where $\boldsymbol{c}_{j,l} = [c_{j,l,0}, c_{j,l,1}, \cdots, c_{j,l,p-1}]$ and $c_{j,l,z}$ is the cost we pay for assigning element $p_{j,l}$ in B to the value z for $0 \le z \le p-1$.

Let S_i denote the set of all possible and distinct length-$2i$ potential cycles represented by the ordered series as in (4) and $|S_i|$ denote the number of all the elements in set S_i. Therefore $S_i = \{\boldsymbol{s}_{i1}, \boldsymbol{s}_{i2}, \cdots, \boldsymbol{s}_{i|S_i|}\}$ with $\boldsymbol{s}_{ik} = [(j_0^{(k)}, l_0^{(k)}), (j_1^{(k)}, l_0^{(k)}), (j_1^{(k)}, l_1^{(k)}), \cdots, (j_{i-1}^{(k)}, l_{i-1}^{(k)}), (j_0^{(k)}, l_{i-1}^{(k)})]$ for $1 \le k \le |S_i|$. Suppose the desired girth is g and the weight vector is $\boldsymbol{w} = [w_2, w_3, \cdots, w_{g/2-1}]$, where w_i is the cost weight for length-$2i$ cycles.

Given a parity check matrix H, the corresponding cost matrix C is calculated based on the following argument. For each potential cycle, we go through each of the elements of the base matrix in the potential cycle, and try to mark the "guilty" values of that element that (if we were to change to that value) would result in a cycle, assuming all other base matrix values in the potential cycle are kept unchanged. For example, for a potential six-cycle, we know that a cycle will exist if and only if $p_1 - p_2 + p_3 - p_4 + p_5 - p_6 \mod p = 0$, where p_1 through p_6 are the elements of the potential six-cycle. So if the current summed value of $p_1 - p_2 + p_3 - p_4 + p_5 - p_6 \mod p$ is 1, one knows that the guilty values for p_1, p_3, and p_5 would be one less than the current value, and the guilty values for p_2, p_4, and p_6 would be one greater than the current value.

This is relatively uncomplicated for potential cycles consisting of $2i$ *distinct* elements of the base matrix. It becomes more complicated if some elements of the base matrix in the potential cycle appear twice. This can occur in potential eight-cycles and occurs, e.g., in the second example of Fig. 1. In such cases, we must keep in mind that when a value of an element changes, the contribution to the alternating sum doubles (or triples in the length-12 cycle of Fig. 1 because the path in the base matrix cycles three times). Finding guilty value(s) therefore becomes more complicated. In fact, there can be more than one guilty value for a repeating element if p is even and the current value of the alternating sum is even. On the other hand, there may be no guilty values if p is even and the current value of the alternating sum is odd.

Formally, the we compute the cost matrix as follows.

- Step 1 : Initialize the cost matrix $c_{j,l,z} = 0$ for $0 \le j \le J-1$, $0 \le l \le L-1$, and $0 \le z \le p-1$.
- Step 2 : For $2 \le i \le g/2-1$,
  - Set $x_{j,l,z}^{(i)} = 0$ for $0 \le j \le J-1$, $0 \le l \le L-1$, and $0 \le z \le p-1$. ($x_{j,l,z}^{(i)}$ is a count of the number of cycles of length $2i$ that would result if base matrix element $p_{j,l}$ had value z.)
  - For $1 \le k \le |S_i|$, compute the alternating sum:

$$\alpha = \left[\sum_{e=0}^{2i-1} (-1)^e \cdot p_{j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}}\right] \bmod p \, .$$

For $0 \le e \le 2i-1$, if $\left(j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}\right)$ is unique in $\boldsymbol{s}_{ik}$, compute the guilty value

$$\beta = \left[p_{j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}} - (-1)^e \cdot \alpha\right] \bmod p \, .$$

If $\left(j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}\right)$ is not unique in $\boldsymbol{s}_{ik}$ and occurs for the first time and $\alpha \bmod 2 = 0$, compute the guilty value

$$\beta = \left[p_{j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}} - (-1)^e \cdot \alpha/2\right] \bmod p \, .$$

If $\left(j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}\right)$ is not unique in $\boldsymbol{s}_{ik}$ and occurs for the first time and $(p-\alpha) \bmod 2 = 0$, compute the additional guilty value

$$\beta = \left[p_{j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}} + (-1)^e \cdot (p-\alpha)/2\right] \bmod p\ .$$

In each of the above three cases, increment

$$x^{(i)}_{j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}, \beta} = x^{(i)}_{j^{(k)}_{\lfloor (e-(-1)^e+1)/2 \rfloor}, l^{(k)}_{\lfloor e/2 \rfloor}, \beta} + 1.$$

- Step 3 : For $0 \le j \le J-1$, $0 \le l \le L-1$, and $0 \le z \le p-1$, take the weighted sum

$$c_{j,l,z} = \sum_{i=2}^{g/2-1} x^{(i)}_{j,l,z} \cdot w_i.$$

Now that we have specified the sub-routine for computing the cost matrix, the actual hill-climbing algorithm is relatively straightforward to describe. We assume that we are given a desired girth g and circulant matrix size p. We start with a random base matrix, and keep choosing the move of changing a value of a single base matrix element that most reduces the cost (breaking ties randomly), until hopefully we find a base matrix which has zero costs for each of the current values of the base matrix elements, at which point we return the base matrix, or if we end in a local optimum which has positive cost, we return failure.

- Step 1 : Randomly generate a base matrix B.
- Step 2 : Calculate C based on B. For $0 \le j \le J-1$ and $0 \le l \le L-1$ let

$$\tilde{c}_{j,l} = \min_{z:\, 0 \le z \le p-1} c_{j,l,z}$$

$$\tilde{z}_{j,l} = \operatorname*{argmin}_{z:\, 0 \le z \le p-1} c_{j,l,z}$$

and

$$c'_{j,l} = c_{j,l,p_{j,l}}.$$

- Step 3 : Denote $G = [g_{j,l}]$ as the gain matrix with $g_{j,l} = c'_{j,l} - \tilde{c}_{j,l}$. Let

$$g_{max} = \max_{(j,l):\, 0 \le j \le J-1,\, 0 \le l \le L-1} g_{j,l}$$

$$(j_{max}, l_{max}) = \operatorname*{argmax}_{(j,l):\, 0 \le j \le J-1,\, 0 \le l \le L-1} g_{j,l}\ .$$

If $g_{max} > 0$, update B by setting $p_{j_{max},l_{max}} = \tilde{z}_{j_{max},l_{max}}$ and go to Step 2; otherwise, go to Step 4.
- Step 4 : If $c_{j,l,p_{j,l}} = 0$ for all $0 \le j \le J-1$ and $0 \le l \le L-1$, return the current base matrix, otherwise return 'FAILURE.'

L	4	5	6	7	8	9	10	11	12
Guess-and-test	9	14	18	21	26	33	39	46	54
Hill-climbing	9	13	18	21	25	30	35	41	47

TABLE I

UPPER BOUNDS ON p_{min} FOR $g = 8$ AND $J = 3$ QC-LDPC CODES.

IV. COMPARISON OF GUESS-AND-TEST AND HILL-CLIMBING SEARCHING ALGORITHMS

In [18] Fossorier uses guess-and-test to obtain upper bounds on p_{min} for girth-8 QC-LDPC codes with $J = 8$. We improve on these upper bounds using the hill-climbing algorithm, as shown in Table I. Notice that the guess-and-test actually works quite well up to $L = 8$.

For $g = 10$ and $J = 3$, the guess-and-test algorithm did not find base matrices that were competitive with the hill-climbing algorithm (see more details below), so we just provide the upper bounds on p_{min} found by the hill-climbing searching algorithm in Table II.

L	4	5	6	7	8	9	10	11	12
p_{min}	39	63	103	160	233	329	439	577	758

TABLE II

UPPER BOUNDS ON p_{min} FOUND BY THE HILL-CLIMBING ALGORITHM FOR $g = 10$ AND $J = 3$.

It is dangerous to compare the algorithms by the best p value that one obtains, because either algorithm could get lucky and find an unusually good base matrix. To more fairly compare the efficiency of guess-and-test with hill-climbing we introduce the "success rate". This is the percentage of times that a run of the algorithm results in a base matrix that has the desired p and g. Naturally, the success rate will be a function of the targe p and g.

Figure 2 shows the success rate of guess-and-test and hill-climbing as a function of p, with $J = 3$, $L = 9$ and $g = 8$. We observe that for the guess-and-test to find a parity check matrix with girth 8 at $p = 50$ we need to test 10^6 random matrices on average, as compared to the near certain success of hill-climbing. At $p = 30$, where hill-climbing is still able to succeed in more than one try in 10^5, it is hard to tell how many random base matrices would need to tested to expect one success using guess-and-test, but it is likely to be more than 10^{10}.

Figures 3 and 4 depict the success rate of guess-and-test and hill-climbing with $J = 3$, $L = 12$, $g = 8$ and $J = 3$, $L = 9$, $g = 10$, respectively. We observe the same tendency as in Fig. 2. As L and g increase, the success rate of hill-climbing becomes increasingly superior to that of guess-and-test.

It is important to note that each attempt of hill-climbing is much more computationally intensive than one attempt of guess-and-test. The difference will depend on the particular implementation of each algorithm. In our case, the ratio of expected simulation time of hill-climbing to guess-and-test is given in Table III. Both algorithms are implemented in Python,

L	4	5	6	7	8	9	10	11	12
g=8	61	116	194	305	460	642	833	1106	1368
g=10	136	475	1282	2972	6020	11567	20931	34922	55774

TABLE III

RATIO OF SIMULATION TIME OF THE HILL-CLIMBING SEARCHING ALGORITHM AND GUESS-AND-TEST ALGORITHM FOR $J = 3$ AND p EQUAL TO THE SMALLEST VALUES FOUND BY THE HILL-CLIMBING ALGORITHM AS GIVEN IN TABLE I AND II.

and neither implementation can be considered to be heavily optimized. Because hill-climbing is so much more likeley to be successful than guess-and-test, even with a large relative difference in the efficiency per attempt, we always find hill-climbing to be much more efficient in aggregate.

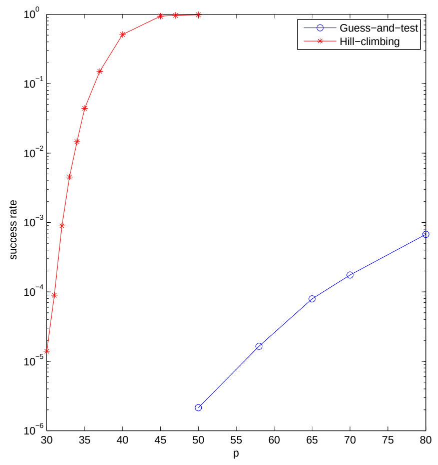

Fig. 2. Comparison of the success rate of guess-and-test and hill-climbing searching algorithms with $J = 3$, $L = 9$ and $g = 8$.

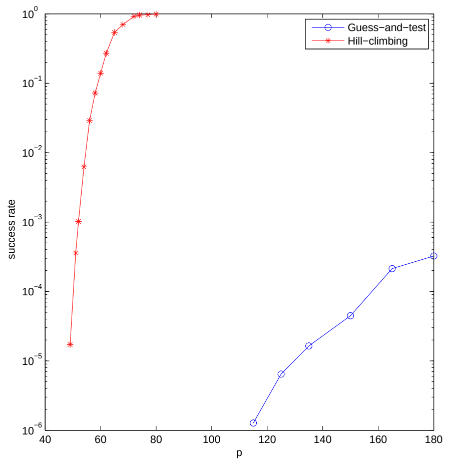

Fig. 3. Comparison of the success rate of guess-and-test and hill-climbing searching algorithms with $J = 3$, $L = 12$ and $g = 8$.

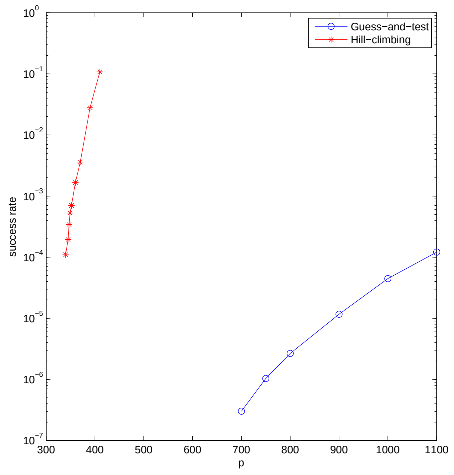

Fig. 4. Comparison of the success rate of guess-and-test and hill-climbing searching algorithms with $J = 3$, $L = 9$ and $g = 10$.

V. ANALYSIS OF THE GUESS-AND-TEST ALGORITHM

In the previous section, we saw that guess-and-test performs nearly as well as hill-climbing for $g = 8$ and $J = 3$ and L small. The performance difference increases for larger g and L. In this section we develop some insights telling us when we can expect guess-and-test to work. We focus on $g = 8$ and $J = 3$.

In a girth-8 base matrix all $p_{j,l}$'s not on the first row or column must be distinct and greater than zero. We call such base matrices "expurgated base matrices." The total number of distinct expurgated base matrices is $\mathrm{P}(p-1, 2(L-1))$, where $\mathrm{P}(n,r) = \frac{n!}{(n-r)!}$. Denote by M the total number of expurgated base matrices that contain at least one cycle with length smaller than 8, and denote by f_l, the total number of 4-cycles and 6-cycles for the l-th expurgated base matrix.

For $p < p_{min}$, since all expurgated base matrices will contain 4-cycles and 6-cycles, $M = \mathrm{P}(p-1, 2(L-1))$; otherwise, for $p \geq p_{min}$, $M < \mathrm{P}(p-1, 2(L-1))$.

Let R_i denote the set of all the potential length-$2i$ cycles in an expurgated base matrix. R_i is a subset of S_i because the expurgation condition eliminates some potential cycles. Denote $R_i = \{\boldsymbol{r}_{i1}, \boldsymbol{r}_{i2}, \cdots, \boldsymbol{r}_{i|R_i|}\}$. Denote the number of expurgated base matrices that indeed contain $\boldsymbol{r}_{ij}$ as a length-$2i$ cycle as $Z_{i,j}$. One can check that $Z_{i,j}$ is a constant for $i = 2, 3$ and all j's with $1 \leq j \leq |R_i|$; therefore set all $Z_{i,j}$ to the constant Z.

The total number of 4-cycles and 6-cycles in all expurgated base matrices is $\sum_{l=1}^{M} f_l = (|R_2| + |R_3|) \cdot Z$. Define $\overline{f}$ to be the mean number of 4-cycles and 6-cycles in the expurgated base matrices: $\overline{f} = \frac{1}{M}\sum_{l=1}^{M} f_l$. Then

$$M = (|R_2| + |R_3|) \cdot Z/\overline{f}. \tag{7}$$

For $J = 3$, straightforward calculations give $|R_2| = (L-1)(L-2)/2$ and $|R_3| = (L-1)^2(L-2)$. We can also express Z with the formula $Z = \widetilde{Z_p}(p-4)(p-5)\cdots(p-2(L-1))$ where $\widetilde{Z_p}$ is the number of distinct solutions to $A - B + C = 0 \bmod p$ with $1 \leq A, B, C \leq p-1$, $A \neq B$, $A \neq C$, and $B \neq C$. The value $\widetilde{Z_p}$ can be obtained through simulation. We can also use simulation to estimate $\overline{f}$.

Suppose we simulate D expurgated base matrices each of which contains d_k 4-cycles or 6-cycles for $k = 1, 2, \cdots, D$. Then $\overline{f} \approx \frac{1}{D}\sum_{k=1}^{D} d_k$ and M can be estimated using $|R_2|$, $|R_3|$, Z and $\overline{f}$ according to (7). We notice that since $(p-4)(p-5)\cdots(p-2(L-1))$ is a common factor in both $\mathrm{P}(p-1, 2(L-1))$ and M, the comparison between $\mathrm{P}(p-1, 2(L-1))$ and M reduces to a comparison between $U(p) = (p-1)(p-2)(p-3)$ and $V(p) = (|R_2| + |R_3|) \cdot \widetilde{Z_p}/\overline{f}$.

Fig. 5 and Fig. 6 depict the comparison of the analytical function $U(p)$ and the statistically estimated function $V(p)$ for the parameters $J = 3$, $L = 4$, $g = 8$ and $J = 3$, $L = 9$, $g = 8$, respectively. We used $D = 10000$.

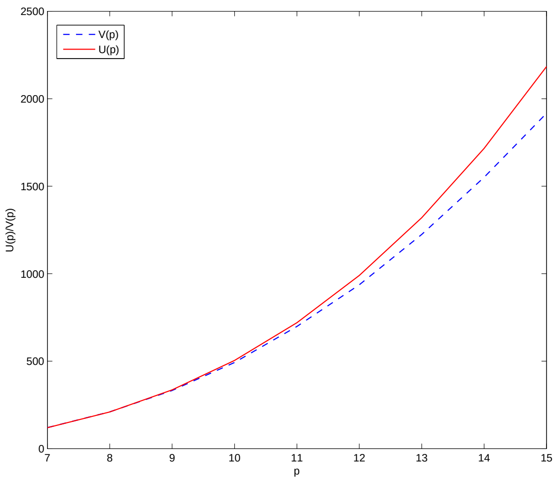

Fig. 5. Comparison of $U(p)$ and $V(p)$ for the parameters $J = 3$, $L = 4$ and $g = 8$.

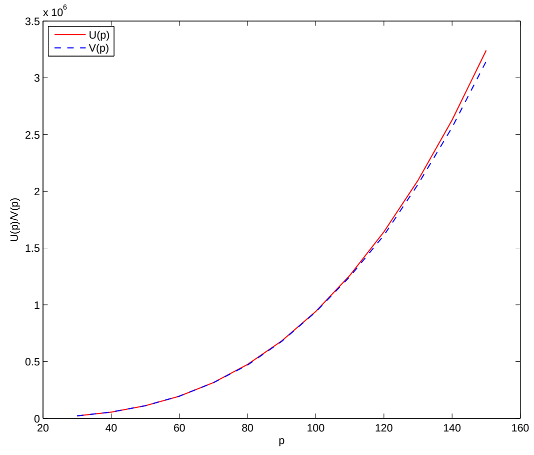

Fig. 6. Comparison of $U(p)$ and $V(p)$ for the parameters $J = 3$, $L = 9$ and $g = 8$.

We observe that for $L = 4$, the functions $U(p)$ and $V(p)$ diverge quickly after $p > p_{min}$, where p_{min} can reliably be estimated to be 9. This means that M becomes apparently smaller than $\mathrm{P}(p-1, 2(L-1))$ as soon as $p > p_{min}$, i.e., the number of expurgated base matrices without 4-cycles and 6-cycles is a significant fraction compared with the total number of expurgated base matrices. On the other hand, when $L = 9$, the functions $U(p)$ and $V(p)$ diverge only when $p >> 30$, (we know that p_{min} is at most 30). In this case, M is very close to $\mathrm{P}(p-1, 2(L-1))$ for a large range of p, i.e., the number of expurgated base matrices without 4-cycles and 6-cycles is tiny compared with the total number of expurgated base matrices.

This explains why guess-and-test fails to find minimum length QC-LDPC codes when L increases. The reason is that for large L, when p increases, although the number of expurgated base matrices increases, the number of cycles increases with almost the same rate for a large range of p. This makes the fraction of "good" base matrices very small, and leads inevitably to a very slow algorithm.

Although the above derivation is for girth-8 codes and $J = 3$, similar analyses can be readily derived for girth-10 codes and general J.

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: M.I.T. Press, 1963.
[2] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399-431, Mar. 1999.
[3] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619-637, Feb. 2001.
[4] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Improved low-density parity check codes using irregular graphs," *IEEE Trans. Inform. Theory*, vol. 47, pp. 585-598, Feb. 2001.
[5] S. Y. Chung, G. D. Forney, T. J. Richardson, and R. L. Urbanke, "On the design of low-density parity check codes within 0.0045 dB of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, pp. 58-60, Feb. 2001.
[6] IEEE Std 802.16e, "Air interface for fixed and mobile broadband wireless access systems," [Online]. Available: http://standards.ieee.org/getieee802/download/802.16e-2005.pdf.
[7] Draft DVB-S2 Standard, [Online]. Available: http://www.dvb.org.
[8] B. Bangerter *et al.*, "High-throughput wireless LAN air interface," *J. Intel Technol.*, vol. 7, pp. 47-57, Aug. 2003.
[9] Y. Kou, S. Lin, and M. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Inform. Theory*, vol. 47, pp. 2711-2736, Nov. 2001.
[10] J. L. Fan, "Array codes as low-density parity-check codes," *Proc. 2nd Int. Symp. Turbo Codes*, Brest, France, Sept. 2000, pp. 545-546.
[11] R. M. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," *Proc. Int. Symp. on Commun. Theory Applications*, Ambleside, England, Jul. 2001.
[12] D. Sridhara, T. Fuja, and R. M. Tanner, "Low density parity check codes from permutation matrices," *Proc. Conf. on Inform. Sciences and Systems*, Baltimore, MD, Mar. 2001.
[13] T. Okamura, "Designing LDPC codes using cyclic shifts," *Proc. IEEE Int. Symp. Information Theory*, Yokohama, Japan, June/July 2003, p. 151.
[14] R. M. Tanner, D. Sridhara, T. Fuja, and D. J. Costello Jr., "LDPC block and convolutional codes based on circulant matrices," *IEEE Trans. Inform. Theory*, vol. 50, pp. 2966-2984, Dec. 2004.
[15] T. J. Richardson, "Error Floors of LDPC Codes," *Proc. 41st Annual Allerton Conf. on Communications, Control and Computing*, 2003.
[16] S. K. Chilappagari, S. Sankaranarayanan and B. Vasic, "Error floors of LDPC codes on the binary symmetric channel," Proc. IEEE Int. Conf. on Commun., Jun. 2006, pp. 1089-1094.
[17] X.-Y. Hu, E. Eleftheriou, and D.-M. Arnold, "Irregular progressive-edge growth (PEG) Tanner graphs," *Proc. IEEE Int. Symp. Information Theory*, Lausanne, Switzerland, July 2002, p. 480.
[18] M. P. C. Fossorier, "Quasi-cyclic low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inform. Theory*, vol. 50, no. 8, pp. 1788-1793, Aug. 2004.
[19] Y. Wang, J.S. Yedidia, S.C. Draper, in preparation.