The Performance of LDPC codes with Large Girth *

Michael E. O'Sullivan
Dept. of Mathematics and Statistics
San Diego State University
San Diego, CA 92182-7720
mosulliv@math.sdsu.edu

John Brevik
Dept. of Computer Science
University of California
Santa Barbara, CA 75275-0338
jbrevik@cs.ucsb.edu

Rich Wolski
Dept. of Computer Science
University of California
Santa Barbara, CA 75275-0338
rich@cs.ucsb.edu

Abstract

We experiment with variations of a method for constructing low-density parity-check codes to investigate the effect on decoding performance under the sum-product algorithm. The method uses circulant permutation matrices or affine permutation matrices as building blocks. We have developed an algorithm to choose these matrices so as to maximize the girth of the parity-check matrix. Substantial performance gains result from requiring large girth, but other factors also play a role.

Simulations were conducted using a wide-area computational grid program, which allowed investigation of high signal-to-noise ratios and their corresponding low output bit error rates.

1 Introduction

In this article we analyze the performance of low-density parity-check (LDPC) codes generated by three methods, all based on the same general construction. We find that, in all methods, the codes with the largest possible girth substantially outperformed those with smaller girth. In fact, for two check matrices with the same girth g, we observed a correlation between the number of cycles of length g and error correction performance. In order to achieve a fair comparison, all codes tested were $(3, 6)$-regular, had designed rate $\frac{1}{2}$ and were of one of three lengths. We used the sum-product algorithm with a maximum of 200 iterations. To determine the bit error rate for each code, we developed a wide-area computational grid program capable of decoding a large number of randomly generated input vectors simultaneously. By aggregating the computational power from several large installations, we are able to investigate high signal-to-noise ratios and their corresponding low output bit error rates.

*This work was supported, in part, by grants from the National Science Foundation numbered CAREER-0093166, ANR-0213911, CHE-0321368, and CHE–0216563, the San Diego Supercomputer Center, and by Hewlett-Packard.

We build on a standard approach for constructing low-density matrices, which is to write the matrix H as an $R \times L$ composite of submatrices:

$$H = \begin{bmatrix} A_{11} & A_{12} & \dots & \dots A_{1L} \\ A_{21} & A_{22} & \dots & \dots A_{2L} \\ \dots & \dots & \dots & \dots \\ A_{R1} & A_{R2} & \dots & \dots A_{RL} \end{bmatrix} \tag{1}$$

The parameters R and L are usually relatively small and the A_{rl} have a simple description. This leads to a compact description of H, which should be advantageous in hardware implementation.

There have been several articles using circulant permutation matrices for each A_{rl}, for example [3, 5, 12], while [1] introduced the use of affine permutation matrices. A method for achieving large girth in the bipartite graph associated to H was presented in [6] and generalized in [8]. Here we use three methods based on the general construction in [8] for choosing the submatrices. The first method takes each A_{rl} to be a circulant, the second uses either the zero matrix or a circulant. In the third method, each A_{rl} is a sum of affine permutation matrices.

2 The construction

In this section we apply the general construction and results concerning girth from [8] to the three methods. Since the girth is a property of the bipartite graph of H, the construction is described using the language of graphs (see also [11, 7]). We define a bipartite graph as a set of edges, E, two disjoint sets of vertices, L and R, and two structural maps, $\lambda : E \to L$ and $\rho : E \to R$ designating the ends of each edge e.

Construction 2.1. *Let E, L, R, λ, ρ be a finite bipartite graph. We will call it the* seed graph *for the following construction. Let $G = \{1, \dots, N\}$. For each $e \in E$ fix a permutation θ_e of G. We define a new bipartite graph with edge set $E \times G$, vertex sets $L \times G$ and $R \times G$ and structural maps $\bar{\lambda}(e, g) = (\lambda(e), g)$ and $\bar{\rho}(e, g) = (\rho(e), \theta_e(g))$. We will call the resulting graph the* covering graph.

For each $l \in L$ and $r \in R$, there is a subgraph of the covering graph consisting of two vertex sets $\{(l, g) : g \in G\}$ and $\{(r, g) : g \in G\}$, each with N elements, and having edge set $\{(e, g) \in E \times G : \lambda(e) = l \text{ and } \rho(e) = r\}$. Let A_{rl} be the $N \times N$ incidence matrix for this subgraph. It is the sum of the permutation matrices for the θ_e satifying $\rho(e) = r$ and $\lambda(e) = l$. The incidence matrix for the covering graph is then

$$\begin{bmatrix} A_{1,1} & \dots & A_{1,\#L} \\ \dots & \dots & \dots \\ A_{\#R,1} & \dots & A_{\#R,\#L} \end{bmatrix}$$

We consider first the case where the seed graph is a complete bipartite graph and each θ_e is a cyclic shift permutation given by $\theta_e(x) \equiv x + c_e \mod N$. The following result shows how to choose the c_e to construct a graph with given girth less than or equal to 12.

Proposition 2.2. *In Construction 2.1, let the seed graph be a complete bipartite graph. Let each θ_e be a translation, by c_e modulo n. Let Y be the submodule of $\mathbb{Z}^E$ defined by*

$$\sum_{e \in \rho^{-1}(r)} y_e = 0 \quad \textit{and} \quad \sum_{e \in \lambda^{-1}(l)} y_e = 0 \tag{2}$$

Suppose that $M \leq 5$, and suppose that for any $y \in Y$ of L_1-norm at most $2M$,

$$\sum_{e \in T} y_e c_e \not\equiv 0 \mod N \tag{3}$$

Then the covering graph has girth at least $2M + 2$.

Based on this proposition (actually a generalization of it), we implemented an algorithm in Magma that chooses the constants c_e recursively. The algorithm first computes the "short" elements of Y (those of L_1-norm at most $2M$), and, given an enumeration of the edges, sorts the short vectors by the maximum element of the support. For each edge e in turn, it generates a list of elements of $\mathbb{Z}/N$ that would violate the constraints imposed on c_e by (3). It then chooses c_e outside of this set.

It should be clear that a cycle in the covering graph yields a cycle in the seed graph, by projection onto the first coordinate. One may think, in Construction 2.1, of using the permutations to "break open" the small cycles in the seed graph. In fact, a sequence of edges $e_1, f_1, e_2, \ldots, e_m, f_m$ determining a cycle in the seed graph is opened if and only if the composition

$$\theta_{f_m}^{-1}\theta_{e_m} \cdots \theta_{f_1}^{-1}\theta_{e_1} \tag{4}$$

has a fixed point [1, 3]. We distinguish between two types of cycles, *balanced* and *unbalanced*. In a balanced cycle, each edge is traversed in each direction the same number of times. For example, if the seed graph has three edges between some pair of vertices, then it has a balanced 6-cycle. Similarly, a complete bipartite graph with at least 2 right vertices and 3 left vertices has a balanced 12-cycle. In Construction 2.1, balanced cycles cannot be broken if the permutations are all translations (or, more generally, if they commute), since the composition (4) is the identity. Consequently, the method presented in Proposition 2.2 can only achieve girth 12 as observed in [5] .

The second method uses a sparser seed graph that has no balanced 12-cycles. Its incidence matrix is the concatenation of two incidence matrices for Fano planes:

$$W = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1
\end{bmatrix}$$

With this seed matrix, we used the Magma algorithm outlined above to construct a girth-14 covering graph which yields a rate $\frac{1}{2}$ code of length $34,118$. The parity-check matrix may be described by giving the dimensions of the circulants ($N = 2437$) and, for each of the 42 edges in the seed graph, the corresponding row, column and the shift for the circulant. These may be written as sequences:

$$\begin{aligned}
\rho &= [1,2,3,1,4,5,1,6,7,2,4,7,2,5,6,3,4,6,3,5,7,\\
&\qquad 1,2,4,1,3,6,1,5,7,2,3,5,2,6,7,3,4,7,4,5,6]\\
\lambda &= [1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6,7,7,7,\\
&\qquad 8,8,8,9,9,9,10,10,10,11,11,11,12,12,12,13,13,13,14,14,14]\\
c &= [0,0,0,0,0,0,0,0,0,0,1,3,0,5,11,0,10,32,0,39,72,\\
&\quad 0,14,78,0,96,226,0,177,363,0,230,490,0,431,907,0,411,1001,0,774,1726]
\end{aligned}$$

We also constructed a covering graph of girth 12 from this matrix that yielded a rate $\frac{1}{2}$ code of length $6,072$, which, as we will later show, performed quite well as compared to the previous method.

The third method for code construction uses a smaller seed graph and *affine permutations* rather than circulants. Consider the affine map f on $\mathbb{Z}/N$ given by

$$f(x) = ax + b.$$

When a is coprime to N, f gives a permutation of $\mathbb{Z}/N$. The corresponding affine permutation matrix has a 1 in the i, j position if and only if $i \equiv aj + b \mod N$. Note that when $a = 1$, this permutation matrix is circulant.

For rate-$\frac{1}{2}$ codes with $(3, 6)$-regular parity-check matrices, we used the simplest possible seed graph, one with incidence matrix $[3, 3]$. The check matrix is then

$$[M_1 + M_2 + M_3 \quad M_4 + M_5 + M_6],$$

where each M_i is an $N \times N$ affine permutation matrix. Let us write the affine permutations as $x \mapsto a_e x + b_e \mod N$ for $e = 1, \ldots, 6$. Our technique for obtaining large girth is to handle balanced cycles and unbalanced cycles separately, using the Chinese Remainder Theorem. The condition for breaking balanced cycles depends on polynomials Z_π, defined for each permutation $\pi \in S_m$ as

$$Z_\pi(u_1, \ldots, u_m, v_1, \ldots, v_m) = \sum_{i=1}^{m} v_i \left((\prod_{j=i}^{i-1} u_{\pi(j)})(\prod_{j=i+1}^{m} u_j) - (\prod_{j=i}^{\pi(i)-1} u_{\pi(j)})(\prod_{j=\pi(i)+1}^{m} u_j) \right)$$

Proposition 2.3. *In Construction 2.1, suppose that p and q are coprime integers, $N = pq$, and $G = \mathbb{Z}/N$. Suppose that each θ_e is an affine permutation $\theta_e(x) \equiv a_e x + c_e \mod N$ and that $a_e \equiv 1 \mod p$. Let Y be the submodule of $\mathbb{Z}^E$ defined in Proposition 2.2 above. Also suppose that the following conditions hold:*

1. *For any $y \in Y$ of L_1-norm at most $2M$,*

$$\sum_{e \in E} y_e b_e \not\equiv 0 \mod p$$

2. *For any $m \leq M$, for any permutation π on $\{1, \ldots, m\}$ and for any $e : \{1, \ldots, m\} \to E$, such that e and $e \circ \pi$ satisfy $e(i) \neq e(\pi(i))$, $e(i) \neq e(\pi(i-1))$, $\rho(e(i)) = \rho(e(\pi(i)))$ and $\lambda(e(i)) = \lambda(e(\pi\epsilon(i)))$ we have*

$$Z_\pi(a_{e(1)}, \ldots, a_{e(m)}, b_{e(1)}, \ldots, b_{e(m)}) \not\equiv 0 \mod q$$

Then the covering graph has girth at least $2M + 2$.

The Magma program discussed earlier computes the values $b_e \mod p$. A separate Magma program recursively computes $a_e \mod q$ and $b_e \mod q$, base on item (2) of the proposition. The Chinese Remainder Theorem is then employed to get a_e and b_e modulo pq.

3 Experimental Results

To investigate the error performance of each LDPC code, we have written a grid program [4] that is capable of marshaling a large heterogeneous collection of machines. The program consists of a control infrastructure that we execute on machines located at the University of California, Santa Barbara (UCSB) and "computational engines" that can be run on any machine connected to the Internet running a variant of Unix or Linux. Each computational engine contacts the control programs via a network connection it establishes between itself and UCSB. The control programs send each engine an LDPC code to test, a signal-to-noise ratio (SNR) to test, and the number of tests to perform before checking back for further instructions.

When a computational engine receives these parameters from a control program (which it contacts immediately when it starts) it begins testing the graph by generating random error vectors according to the transmitted SNR. Each input vector is then decoded using the sum-product algorithm with a maximum of 200 iterations, and the results are checked. When a decoding failure occurs, both the input vector that failed to decode correctly and the resulting output vector are sent to the control program for logging. For each signal-to-noise ratio, test vectors were generated until 200 vectors failed to decode. The total number of bits in all tested code words as well as the number of bit failures are also sent and aggregated by the control program to compute an overall bit-error rate.

In this study, we used approximately 850 different processors from 9 sites, with as many as 450 actively computing at any one time. We combined large-scale clusters (some behind firewalls) with a pool of approximately 500 machines managed by Condor [10] at the University of Wisconsin. Condor is a cycle-harvesting system that spawns processes submitted to it on the machines it controls. When those machines are reclaimed by their respective users (by the activation of the machine console or the presence of locally-generated processes) the Condor-spawned processes are terminated. In this way it uses the "idle" cycles that are available from machines running Condor. At present, the Condor pool at Wisconsin is able to harvest cycles from approximately 2000 Linux, Solaris, and IRIX machines located on the University of Wisconsin campus, and at the National Center for Supercomputing Applications (NCSA) in Urbana-Champaign, Illinois. The 500 machines we use from Condor in this study are chosen by the Condor system itself based on the availability of machines, and the other Condor jobs with which our application must compete. Our program uses EveryWare [2] to combine the computational power provided by this opportunistic Condor access with jobs launched via different batch-queuing mechanisms on various high-performance cluster machines. The result is a nationally-distributed program for determining the bit-error rate of a graph that can use all of the processors given to it (by Condor or any other local scheduling mechanism).

The results of these experiments are shown in the performance curves below. Each shows the bit-error rate after decoding as a function of the signal to noise ratio of a symmetric memoryless Gaussian noise channel. We discuss here codes of length 282, length 1000 (approximately) and length 6000 (approximately). In these graphs, a solid line is used for codes from the first method (using circulants), a dashed line for codes constructed from the third method (affine permutations), and dot-dash line for the second method (using the seed matrix from the Fano plane). Some of the graphs include the performance of randomly generated codes, which are shown using a dotted line. A symbol

is used to indicate the girth, a triangle for girth 6, a square for girth 8, a pentagram for girth 10, and a hexagram for girth 12.

Figure 1 compares four codes of length 282 and rate $\frac{1}{2}$: a random generated code of girth 6, a randomly generated code of girth 8, and codes from the first and third method with girth 8. The difference in performance is not dramatic, but we can see that the randomly generated code of girth 6 was worst, and the first construction method was best, the difference between these two being a factor of 10 at SNR 4.5. At this length we tested altogether ten randomly generated codes of girth 6 and five generated by method one. We found that among the randomly generated codes there is a significant correlation $(.634; p < .05)$ between the number of 6-cycles in the associated graph and the logarithm of the bit error rate. Among all codes of length 282 and girth 6 tested, the correlation is also .634 ($p < .01$), and this correlation goes up to .851 ($p < .0005$) after the removal of one outlier.

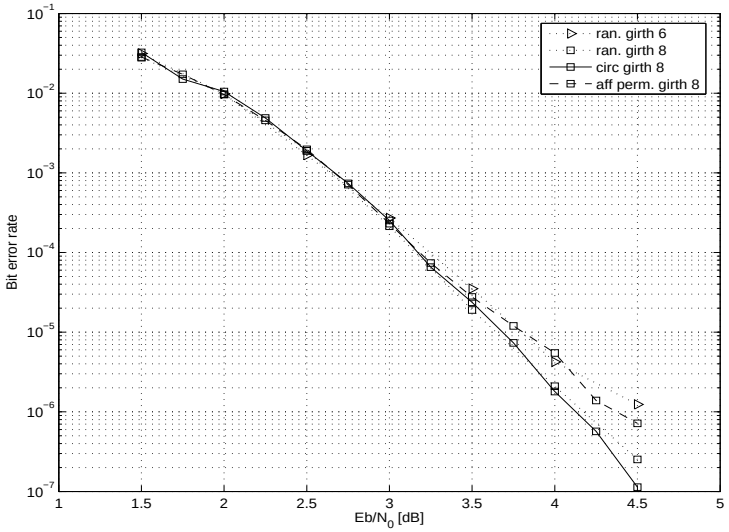

Figure 1: Comparison of rate-$\frac{1}{2}$, length-282 codes.

Figure 2 compares four codes of length roughly 1000 and rate $\frac{1}{2}$: The codes from methods one and three are from matrices with girth 10. The two methods required slightly different lengths; we used a length of 1002 for method one and 1010 for method three. Two codes from randomly generated matrices, of girths 6 and 8, are also shown, the lengths are 1008 in each case. The difference in performance is much more significant at this length. The performance for the random matrix of girth 6 is by far the worst; the random matrix of girth 8 also performs substantially worse than the codes generated using the other two methods, whose performances are very similar.

Figure 3 compares four codes of rate $\frac{1}{2}$ and length roughly $6,000$. We used the first method with length $6,078$ and a matrix with girth 12. We tried two codes with the second method each of length $6,062$ and two girths, 8 and 12. A randomly generated code of girth 6 and length $6,080$ was also tested. At SNR 2 we see that the ouptut BER for the code from method two is better than the random code by a factor of 10^3 and better than the code from method one by a factor of 10.

We observed something surprising in the course of our experiments which is illustrated in Figure 4. The Magma algorithm discussed above chooses each c_e outside of a certain excluded set B_e. We used two methods for choosing c_e: One was to choose the minimum integer not in B_e, and the other was to choose a random element not in B_e. We found

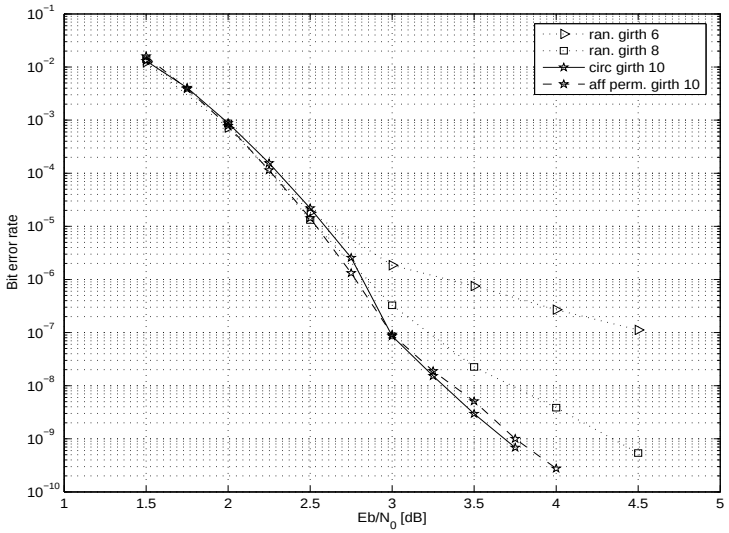

Figure 2: Comparison of rate-$\frac{1}{2}$, length 1002-1010 codes.

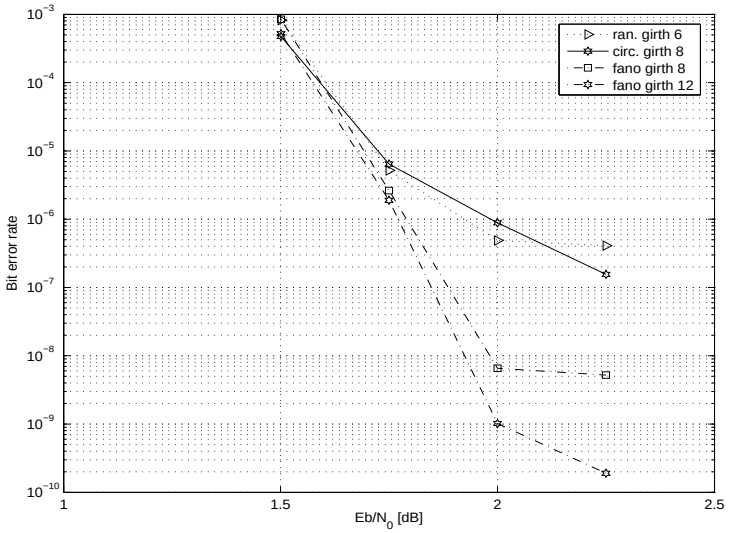

Figure 3: Comparison of rate-$\frac{1}{2}$, codes of length roughly 6000.

that the random choice was signicantly better. Figure 4 compares two girth-8 matrices of length 282, and two girth-8 matrices of length 1002, in each case one matrix used the minimimum element of B_e and the other a random element. The two matrices using the minimum choice actually have roughly the same performance, although one has length 282 and the other length 1002. In this example and in virtually all cases tested, the random choice was dramatically better than the minimum choice. The only exception was when the size N of the circulants was very close to the minimal value necessary to achieve a specified girth. In this case the choices yielded similar results. We also observed that matrices constructed using the minimum choice had a much larger number of cycles of length g than those constructed using the random choice. In the previous three figures and in Figure 5, all matrices for the curves shown were constructed using the random choice.

Figure 5 shows the effect that increasing girth has in the first construction method. All three codes tested were of length 1002 and rate $\frac{1}{2}$. The sole difference in the construction

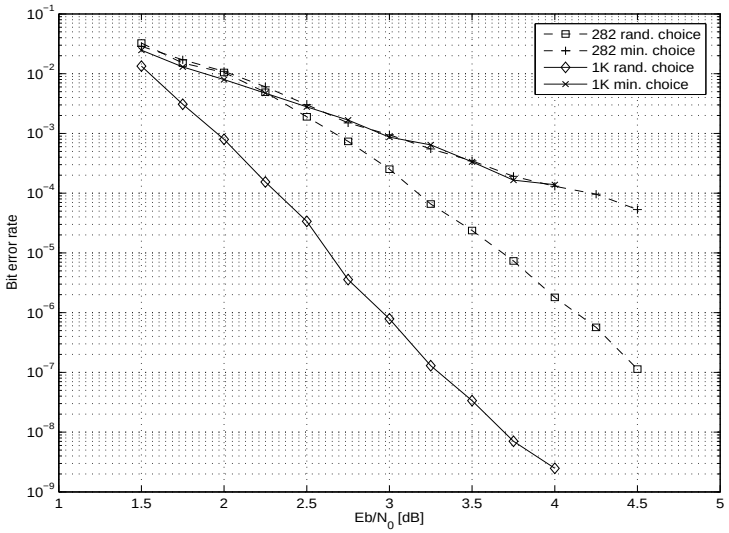

Figure 4: Comparison of two methods for choosing coefficients at lengths 282 and 1002.

was that we imposed the conditions (3) of Proposition 2.2 for $M = 2$, $M = 3$, and $M = 4$, respectively, to ensure girths 6, 8, and 10 for the graphs. We used the random choice for each c_e as opposed to the minimum choice as discussed in the previous paragraph. Each increase of 2 in the girth decreased the bit error rate by a factor of 10 at SNR 3 and above.

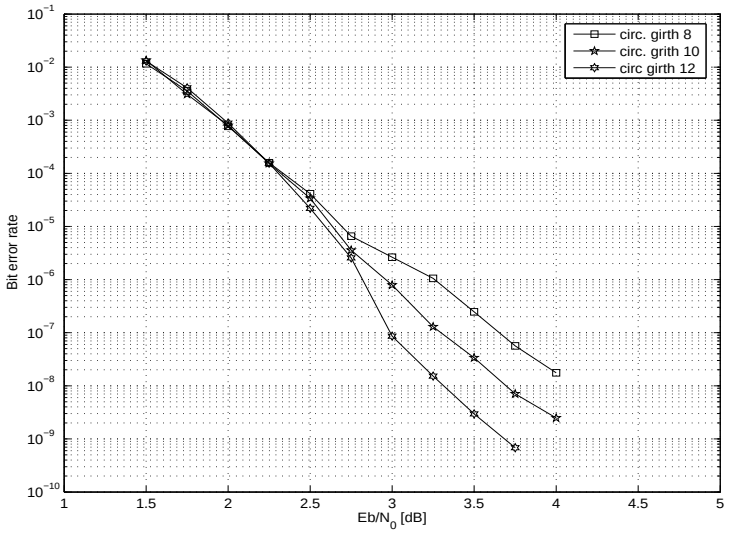

Figure 5: Comparison of rate-$\frac{1}{2}$, $(3, 6)$-regular, length-1002 codes with varying girth.

4 Conclusion

We tested the performance of LDPC codes using three methods for constructing parity-check matrices. In order to isolate the effect of small cycles in decoding, all codes tested in our experiments were $(3, 6)$-regular. For all three methods, we found that large girth was advantageous. The superior performance at length $6,000$ of the method using a

a seed matrix from two Fano planes, suggests that a sparse seed matrix may also be advantageous.

References

[1] J. Bond, S. Hui, and H. Schmidt. Linear-congruence construction of low-density check codes. In B. Marcus and J. Rosenthal, editors, *Codes, Systems and Graphical Models*, IMA Vol. 123, pages 83–100. Springer-Verlag, 2001.

[2] R. Wolski, J. Brevik, G. Obertelli, and N. Spring. Writing Programs that Run Every-Ware on the Computational Grid. IEEE Transactions on Parallel and Distributed, Vol. 12, Number 10, pp. 1066–1080, October, 2001.

[3] J. L. Fan. Array codes as low-density parity-check codes. In *Proc. 2nd Int. Symp. Turbo Codes*, pages 543–546, Brest, France, 2000.

[4] I. Foster and K. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., 1998.

[5] M. P. C. Fossorier. Quasi-cyclic, low-density parity-check codes from circulant permutation matrices. *IEEE Trans. Inform. Theory*, 50(8):1788–1793, 2004.

[6] M. Greferath, M. E. O'Sullivan, and R. Smarandache. Construction of good LDPC codes using dilation matrices. In *Proc. 2004 IEEE Int. Symp. Inform. Theory*, page 237, Chicago, Il., 2004.

[7] R. Koetter and P. O. Vontobel. Graph covers and iterative decoding of finite length codes. In *Proc. 3rd Int. Conf. on Turbo Codes and Related Topics*, pages 75–82, 2003.

[8] M. E. O'Sullivan, Algebraic construction of sparse matrices with large girth, submitted to *IEEE Trans. Inform. Theory.*

[9] Heng Tang, Jun Xu, Yu Kou, S. Lin, and K. Abdel-Ghaffar. On algebraic construction of gallager and circulant low-density parity-check codes. *IEEE Trans. Inform. Theory*, 50(6):1269–1279, June 2004.

[10] T. Tannenbaum and M. Litzkow. The Condor Distributed Processing System. Dr. Dobbs Journal, February, 1995.

[11] R. M. Tanner, D. Sridhara, and T. Fuja. A class of group-structured LDPC codes. In *Proc. of ICSTA 2001*, Ambleside England, 2001.

[12] R. Michael Tanner, Deepak Sridhara, Arvind Sridharan, Thomas E. Fuja, and Daniel J. Costello, Jr. LDPC block and convolutional codes based on circulant matrices. *IEEE Trans. Inform. Theory*, 50(12):2966–2984, 2004.