LDPC Codes of Arbitrary Girth

Tsvetan Asamov and Nuh Aydin
Department of Mathematics, Kenyon College, Gambier, OH 43022 USA

***Abstract*— For regular degree two LDPC codes the MAP and BP thresholds coincide. In that case there is a strong relationship between high girth and performance. This article presents a greedy algorithm, called successive level growth (SLG), for the construction of LDPC codes with arbitrarily specified girth. The simulation results show that our codes exhibit significant coding gains over randomly constructed LDPC codes and in some cases outperform PEG codes in the additive white Gaussian noise channel.**

***Index Terms*— LDPC codes, large girth, Successive Level Growth (SLG).**

I. Introduction

Low density parity-check (LDPC) codes have been a subject of intensive research for the past few years because of their outstanding error-correcting capabilities. LDPC codes are specified by a sparse parity check matrix [1] and its corresponding Tanner graph [2]. The girth g of the code is defined as the length of the shortest cycle of the graph. Tanner [2] determined a lower bound on the minimum distance that grows exponentially with the girth of the code. Moreover, Gallager [1] showed that under the sum-product algorithm [3] the number of independent decoding iterations is proportional to g. A number of graph-algorithmic approaches such as bit-filling [4], [5] and progressive edge-growth [6] have been suggested in the past few years. In this correspondence we present a new technique to construct LDPC codes of any desired girth. It is organized as follows: in section II we introduce the notation that we are using. The algorithm is discussed in section III. Finally, construction parameters and simulation results are presented in section IV.

II. Notation

LDPC codes are defined by sparse parity-check matrices. There exists a one-to-one correspondence between the set of all parity-check matrices and the set of all bipartite graphs. Hence we can use the two terms interchangeably.

Typically, a graph is denoted by (V,E) where V is the set of vertices, also known as nodes, and E is the set of edges. In the case of a bipartite graph, $V = V_s \cup V_c$, where V_s denotes the set of symbol nodes and V_c is the set of check nodes. We split V_s and V_c even further into k disjoint subsets which we refer to as levels or tiers. Thus, $V_s = V_s^0 \cup V_s^1 \cup .. \cup V_s^{k-1}$ and $V_c = V_c^0 \cup V_c^1 \cup .. \cup V_c^{k-1}$. Moreover, $V_s^p = \{s_0^p, s_1^p, s_2^p, ..\}$ and $V_c^p = \{c_0^p, c_1^p, c_2^p, ..\}$, where s_q^p denotes the q^{th} symbol node from the p^{th} level and c_q^p denotes the q^{th} check node from the p^{th} level. Hence, $V_s = \bigcup_{p=0}^{k-1} V_s^p$ and $V_c = \bigcup_{p=0}^{k-1} V_c^p$, while $V_s^{p_1} \cap V_s^{p_2} = \emptyset$ and $V_c^{p_1} \cap V_c^{p_2} = \emptyset$, for all $0 \le p_1 < p_2 \le k-1$. We denote the degree of a node i as d_i. A graph is called (d_s, d_c)-regular if every symbol vertex has a degree of d_s and every check vertex has a degree of d_c. Moreover, an edge between node i and node j is referred to as $Edge(i,j)$. Finally, we denote a breadth-first search starting from vertex i with $BFS(i)$.

III. Successive Level Growth

A. An Observation on Regular Tanner Graphs

Consider the parity-check matrix H given below. H specifies a $(2,4)$-regular LDPC code of girth 8. Notice that $H[0,0] = 1$, so there is an edge connecting s_0^0 and c_0^0. However, if that edge is removed, then $BFS(s_0^0)$ and $BFS(c_0^0)$ would not have any common nodes after 3 levels of each search have been performed.

$$\begin{array}{cccccccccccccccc|c}
S_0^0 & S_0^1 & S_1^1 & S_2^1 & S_0^2 & S_1^2 & S_2^2 & S_0^3 & S_1^3 & S_2^3 & S_3^3 & S_4^3 & S_5^3 & S_6^3 & S_7^3 & S_8^3 & \\
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 & C_0^0\\
1&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0 & C_0^1\\
0&1&0&0&0&0&0&1&1&1&0&0&0&0&0&0 & C_0^2\\
0&0&1&0&0&0&0&0&0&0&1&1&1&0&0&0 & C_1^2\\
0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&1 & C_2^2\\
0&0&0&0&1&0&0&1&0&0&1&0&0&1&0&0 & C_0^3\\
0&0&0&0&0&1&0&0&1&0&0&1&0&0&1&0 & C_1^3\\
0&0&0&0&0&0&1&0&0&1&0&0&1&0&0&1 & C_2^3
\end{array}$$

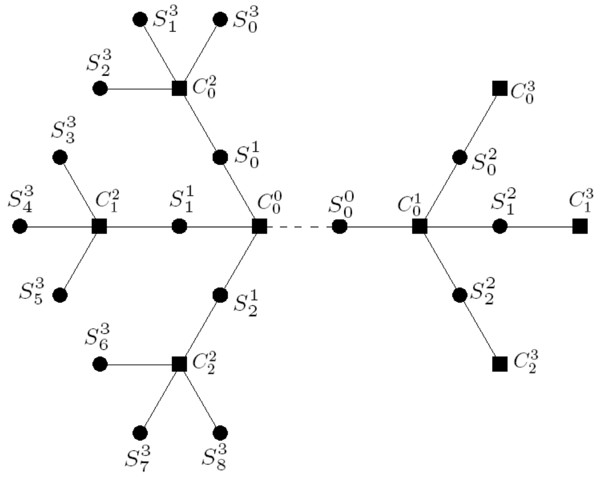

Fig. 1. If $H[0,0] = 0$, $BFS(C_0^0)$ and $BFS(S_0^0)$ would not share any common nodes after 3 levels of each search have been performed.

The Gallager bound on the length of a (d_s, d_c)-regular code can be computed in the following manner :

Algorithm 1 Compute the minimum possible length for a (d_s, d_c)-regular LDPC code of girth g

```
$min\_length = 1$; $add = 1$;
for $t = 1$ to $\frac{g}{2}$ do
 if ($t$ mod 2) = 0 then
  $add = add \cdot (d_c - 1)$
 else
  $add = add \cdot (d_v - 1)$
 end if
 $min\_length = min\_length + add$
end for
return $min\_length$
```

B. Successive Level Growth Construction

Here we describe the details of our algorithm, Successive Level Growth (SLG). The input parameters comprise the desired girth g, symbol nodes degree d_s, check nodes degree d_c and the maximum acceptable length max_length. The output is a possibly regular but often irregular Tanner graph of girth at least g and length smaller than or equal to max_length. The rate of the code can be adjusted via $R = 1 - \frac{d_s}{d_c}$. The main idea of the SLG algorithm is to start with a graph consisting of a single symbol node, single check node and a single edge connecting them. Then the graph is expanded in a level by level fashion. When a suitable level is reached, edges are being added in such a manner that the resulting graph contains no cycles of length less than g. In our implementation only odd-indexed levels satisfied the selection criterion but in general other rules can be used as well. The whole process terminates when it becomes impossible to add a new level. This could be caused either by all symbol nodes having a degree equal to d_s or the impossibility of adding a new level without the total number of symbol nodes exceeding max_length. Finally, we call the Add Appropriate Edges routine if the very last level does not satisfy the selection criterion.

General SLG Algorithm

```
Algorithm 1
if($max\_length \geq min\_length$) then
  $V_s = \{s_0^0\}$; $V_c = \{c_0^0\}$; $E = \{(s_0^0, c_0^0)\}$;
  $level = 0$;
 do
  Add New Level
  $flag = true$;
  if (level selection criterion is satisfied) then
   Add Appropriate Edges
   Output Intermediate Graph
   $flag = false$;
  end if
 while (a new level can be added)
 if ($flag$) then
  Add Appropriate Edges
  Output Final Graph
 end if
end if
```

The Add New Level routine is quite straightforward. Let n and m denote the number of symbol and check vertices respectively at the current level. For each symbol node i of degree $d_i < d_s$, $d_s - d_i$ new check nodes from the next level are created and linked to i. In a similar fashion, for each check node j of degree $d_j < d_c$, $d_c - d_j$ new symbol nodes are created and linked to j.

Add New Level

```
$t = 0$;
 for $i = s_0^{level}$ to $s_{n-1}^{level}$ do
  while $(d_i < d_s)$ do
   $V_c \leftarrow s_t^{level+1}$; $E \leftarrow Edge(i, s_t^{level+1})$; $t = t + 1$;
  end do
 end for
$t = 0$;
 for $j = c_0^{level}$ to $c_{m-1}^{level}$ do
  while $(d_j < d_c)$ do
   $V_j \leftarrow s_t^{level+1}$; $E \leftarrow Edge(s_t^{level+1}, j)$; $t = t + 1$;
  end do
 end for
$level = level + 1$;
```

Adding edges to the graph inevitably leads to the formation of cycles. One possible selection process would be the formation of cycles of the largest possible length. However we adopt a different strategy, namely the formation of cycles which are as small as possible but not smaller than g. This allows the addition of a larger number of edges, while not affecting the girth. In addition, we also try to make the check node degree distribution more uniform by giving preference to check nodes of smaller degrees.

Add Appropriate Edges - General Version

```
do
 $new\_edge = false$; $temp = \infty$; $d_c\_temp = d_c$;
 for $i = s_0^{level}$ to $s_{n-1}^{level}$ do
  if $(d_i < d_s)$ then
   for $j = c_0^{level}$ to $c_{m-1}^{level}$ do
    if $(d_j < d_c\_temp)$ then
     if $(ShortestPath(i, j) \leq temp)$ then
      if $(ShortestPath(i, j) \geq (g - 1))$ then
       $temp = ShortestPath(i, j)$;
       $new\_edge = true$; $d_c\_temp = d_j$;
       $s\_temp = i$; $c\_temp = j$;
      end if
     end if
    end if
   end for
  end if
 end for
 if($new\_edge$) then
  $E \leftarrow Edge(s\_temp, c\_temp)$;
 end if
while($new\_edge$)
```

Finding the shortest path from node i to node j involves either $BFS(i)$ or $BFS(j)$, or another procedure of equivalent complexity. Moreover, a single breadth-first search finds the distance from the starting node, to every other node in the graph. Hence, if the check node selection rules are implemented into the breadth-first search, an equivalent, up to node indexing, version with lower computational complexity can be constructed for the Add Appropriate Edges routine. This is the approach that we employed for our implementation.

```
Add Appropriate Edges - Fast Version
do
  $new\_edge = false$; $temp = \infty$; $d_c\_temp = d_c$;
  for $i = s_0^{level}$ to $s_{n-1}^{level}$ do
    if $(d_i < d_s)$ then
      $ModifiedBFS(i)$;
    end if
  if($new\_edge$) then
    $E \leftarrow Edge(s\_temp, c\_temp)$;
  end if
while($new\_edge$)
```

As a final remark in this section, we would like to mention that the given method can be generalized to the construction of k-partite graphs of arbitrary girth.

IV. Simulation Results

In this section we compare the bit-error and word error rates of two codes constructed by the proposed algorithms with the error rates of random LDPC codes and progressive edge growth codes in the additive white Gaussian noise channel. The sum-product algorithm is used for decoding and the maximum number of iterations is set to 100. Moreover, we employ the rate-adjusted signal to noise ratio suggested in [7] $SNR_1 = 10\log_{10}[E_b/(2R\sigma^2)]$, where $SNR_2 = 10\log_{10}[E_b/(2\sigma^2)]$ is the usual signal to noise ratio.
The relationship between girth and performance has been known to be relatively weak for codes of symbol degree greater than two. However, when $d_s = 2$ the MAP and BP thresholds coincide. In that case the girth of the code is critical. Table I contains the input and output parameters of two codes we constructed using SLG. Notice that as both codes are regular any value of max_length that is not smaller than respectively 425 or 105 could have been used. Error rates comparisons with symbol degree two PEG and random codes are plotted below.

TABLE I - SLG Parameters

	Input			Output					
	g	d_s	d_c	g	Avg.d_s	Avg.d_c	V_s	V_c	Min.Dist.
Fig. 2	16	2	5	16	2.00	5.00	425	170	8
Fig. 3	12	2	5	12	2.00	5.00	105	42	6

In Figure 2 all the codes yield identical results in the low SNR region. However the SLG code outperforms the PEG and the randomly generated code in the high SNR range. At a BER of 10^{-6} we register a coding gain of approximately 0.3 dB over the PEG code.
In this case all symbol nodes of the SLG code have local girth of 16, while the PEG code has minimum distance of 6 and local cycle distribution $423x^{12} + 2x^{14}$.
Similarly, in Figure 3 the SLG LDPC code outperforms the random LDPC code in the high SNR region. At a BER of 10^{-6} our graph indicates a coding gain of 0.35 dB over the PEG code. While the SLG has a local cycle distribution of $105x^{12}$ and minimum distance of 6, the PEG code exhibits minimum distance of 4 and local cycle distribution of $15x^8 + 90x^{10}$.

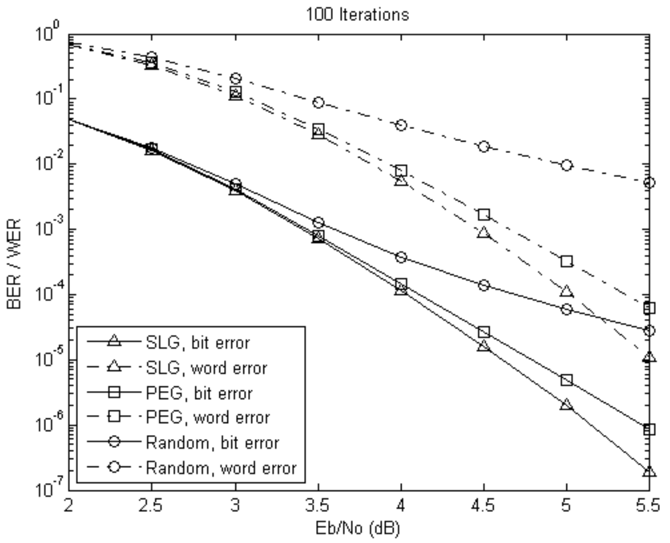

Fig. 2. BER and WER comparison for a girth 16 SLG code, PEG code and random LDPC code.

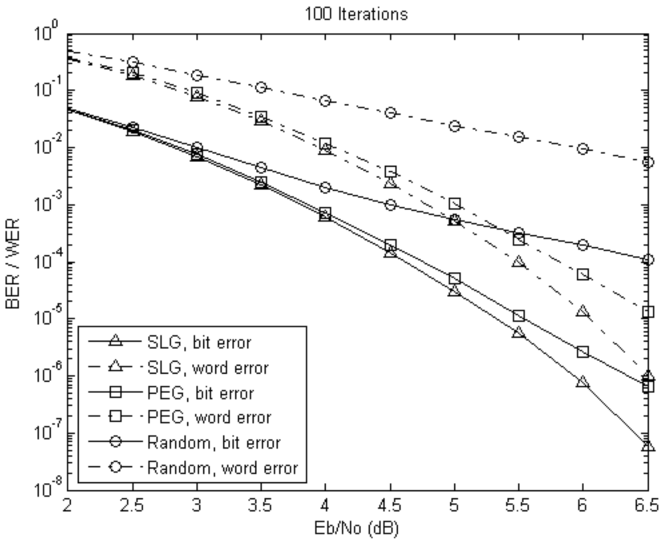

Fig. 3. BER and WER comparison for a SLG LDPC code of girth 12, PEG code and random LDPC code.

V. Conclusion

We have discussed a new method for the construction of LDPC codes of high girth. Simulation results show that our codes exhibit significant coding gain over random LDPC codes. Implementing a strategy for the avoidance of degree one symbol nodes and structuring the codes would be the main objective of our future work.

Acknowledgment

This research was funded by the Summer Science Scholars Program at Kenyon College. The authors would also like to thank the anonymous reviewers for their useful feedback.

References

[1] R. G. Gallager, *Low-Density Parity Check Codes.* Cambridge, MA: MIT Press, 1963.

[2] R. M. Tanner, A recursive approach to low complexity codes, *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533547, Sep. 1981.

[3] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, Factor graphs and the sum-product algorithm, *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498519, Feb. 2001.

[4] J. Campello, D. S. Modha, and S. Rajagopalan, Designing LDPC codes using bit-filling, *Proc. of Intl. Conf. Communi. (ICC)*, Helsinki, Finland, June, 2001.

[5] J. Campello and D.S. Modha. Extended bit-filling and LDPC code design. In *Proc. 2001 IEEE Globecom Conference*, 2001.

[6] Xiao-Yu Hu, E. Eleftheriou, and D.M. Arnold. Regular and irregular progressive edge-growth Tanner graphs *Information Theory, IEEE Transactions* vol. 51, no. 1, pp. 386 - 398, Jan. 2005

[7] D. J. C. Mackay, Good error-correcting codes based on very sparse matrices, *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399431, Mar. 1999.

[8] H. Zhang and H. M. F. Moura, Large-girth LDPC codes based on graphical models, *4th IEEE Workshop on Signal Processing Advances in Wireless Communications*, 2003

[9] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, Source code for progressive edge growth parity-check matrix construction [online] Available: *http://www.inference.phy.cam.ac.uk/mackay/codes/PEG/PEG.tar.gz*