Design of Cages with a Randomized Progressive Edge-Growth Algorithm

Auguste Venkiah, David Declercq and Charly Poulliat
ETIS - CNRS UMR 8051 - ENSEA - University of Cergy-Pontoise

Abstract

The Progressive Edge-Growth (PEG) construction is a well known algorithm for constructing bipartite graphs with good girth properties. In this letter, we propose some improvements in the PEG algorithm which greatly improve the girth properties of the resulting graphs: given a graph size, they increase the girth g achievable by the algorithm, and when the girth cannot be increased, our modified algorithm minimizes the number of cycles of length g. As a main illustration, we focus on regular column-weight graphs, which are a class of graphs widely used for non-binary LDPC codes that can be seen as monopartite graphs: for a given target girth g_t, this new instance of the PEG algorithm allows to construct cages, *i.e.* graphs with the minimal size such that a graph of girth g exists, which is the best result one might hope for.

Index Terms

Progressive Edge-Growth (PEG), Low Density Parity Check (LDPC) codes, girth, Tanner graphs.

I. Introduction

Sparse bipartite graphs with large girths are extremely useful in coding theory and most good LDPC code constructions focus on avoiding short cycles in their associated Tanner graph. Graphs of particular interest in the recent literature are those with $d_v = 2$ edges on the variable nodes, also called "cycle graphs" [1]. Such graphs are used to design ultra sparse non-binary (NB) LDPC codes that achieve very good performance at small to moderate codeword lengths and high Galois field orders [2], and in that case it is crucial to focus on the girth properties of the underlying Tanner graph.

A construction based on a progressive edge-growth (PEG) of the graph was proposed in [3], which results in graphs that have higher girths compared to pre-existing techniques. In this letter, we propose some modifications in the PEG algorithm which further improve the girth properties of the resulting graphs: given a graph size, our method improves the girth g achievable by the PEG algorithm, and when the girth cannot be increased, our modified algorithm, that we called RandPEG for "Randomized Progressive Edge-Growth", minimizes the number of cycles of length g.

For a given graph setting and a given target girth, there exists a the minimal size for the graph such that a graph of girth g_t exists, which is often given in terms of a lower bound. In the case of cycle codes ($d_v = 2$), there exists a monopartite representation of the Tanner graph where the vectices of the monopartite graph represent check nodes, and edges represent variable nodes. When such a graph is minimal, meaning that it achieves the lower bound on the size, it is called a cage.

II. Notations and definitions

In this section, we briefly review the PEG algorithm to introduce the notations. A bipartite graph is denoted as (V, E) where V (resp. E) is the set of the vertices (resp. edges). $V = V_c \cup V_s$ where V_c is the set of check nodes and V_s the set of symbol nodes. Let $N = |V_s|$ denote the total number of symbol nodes, which we will refer to as the size of the graph. When the graph is the Tanner graph of an LDPC code, N is the codeword length. For a given graph setting, namely a 3-tuple (d_v, d_c, g), we denote by $N_g^{(d_v,d_c)}$ the lower bound on N such that a regular (d_v, d_c) graph of girth g exists. This lower bound can be easily computed by using the results of [3, lemma 3], and is known *not* to be tight when $d_v = 2$, for $g \geq 18$ [4].

Let $\mathcal{N}_{s_j}^l$ denote the set of all check nodes reached by a tree spanned from symbol node s_j within depth l, and $\bar{\mathcal{N}}_{s_j}^l$ denote the complementary set in V_s. At a given stage of the construction, only a subset of the check nodes have reached a connectivity of d_c, and we call *candidates* the check nodes in $\bar{\mathcal{N}}_{s_j}^l$ whose incident edges have not been all affected. When a particular check node is *selected* among the candidates, an edge is added in the graph between the node s_j and that check node.

The original PEG algorithm [3] is a procedure for constructing a bipartite graph in an edge by edge manner, where the selection of each new edge aims at minimizing the impact on the girth:

at each step the local girth is maximized. For each node s_j, the first edge is chosen randomly, and the other edges are chosen in the set $\bar{\mathcal{N}}_{s_j}^l$, where l is such that $\bar{\mathcal{N}}_{s_j}^l \neq \emptyset$ and $\bar{\mathcal{N}}_{s_j}^{l+1} = \emptyset$, *i.e.* among the nodes that are at the largest depth from the symbol node s_j. This maximizes the length of the cycles created through this new edge. When multiple choices are possible, the algorithm selects the candidate that has the smallest degree under the current setting.

Even though the original PEG algorithm produces only *almost* regular graphs, the construction of *strictly* regular graphs can be easily enforced by discarding all candidates where all the edges have already been assigned.

III. The Randomized-PEG Algorithm

There are basically two differences between the original PEG algorithm and the RandPEG algorithm that we propose in this paper: firstly, the way we build and use the spanning tree is different, and secondly, we introduce an objective function for the edge selection. The RandPEG algorithm is based on a randomization approach: given a target girth g_t, we consider, at each stage of the construction, the maximum number of possibilities when adding an edge in a graph, and we use the objective function to discriminate among the numerous edge candidates. Similarly to Monte Carlo approaches, the algorithm runs many times and stores the best graph.

In this section, we describe our contributions in details. Our goal is to actually reach a given target girth g_t of the bipartite graph, when *all* the edges of the graph have been assigned. Therefore, if at some point of the construction there is no possibility to add an edge without creating a short[1] cycle, then we consider that the algorithm *fails*. In the sequel, we only consider the construction of (d_v, d_c) regular graphs, in order to compare to the known bounds for regular graphs. We point out that this limitation concerns only our study, not the RandPEG algorithm itself, which can be used for the design of regular or irregular graphs.

A. Truncated spanning tree

Instead of spanning to the maximal possible depth, we span the tree only up to a maximal depth l_{max}. This technique, which defines the *nongreedy* version of the algorithm [3], is suggested for the construction of long codes where it would be computationally expensive to build the whole

[1] by short cycle, we mean cycles shorter than the target girth

tree. Here, we argue that this is not only a computational or speed-up enhancement of the algorithm, but that this technique *should* be used when one wants to construct a graph that matches the lower bound $N_g^{(d_v,d_c)}$. We justify our argument with the following three points.

1) Diameter argument: First, we give a justification on how deep the construction tree should be spanned, based on a graph argument: for a given value of the target girth g_t, if the graph has minimum size $N = N_{g_t}^{(d_v,d_c)}$ then the diameter of the graph equals $d = g_t/2$ [5]. Therefore in that case, the tree *must* be spanned up to a maximal depth $l_{max} = g_t$, so that the diameter is ensured to equal $d = g_t/2$. Indeed, if at some point the algorithm selects a node in $\bar{\mathcal{N}}_{s_j}^{l}$ with $l > g_t$, then the condition that diameter of the graph equals $g_t/2$ cannot hold, and the construction will fail.

The spanning of the tree at a given depth $l = g_t$ gives a set of candidates for which we ensure that no cycle smaller than the target girth g_t can be created if such a candidate is selected.

2) The randomization approach: We recall that our goal is to reach a given target girth g_t, when *all* the edges of the graph have been assigned. By spanning the tree less deeply, the number of candidates at each step of the algorithm becomes much larger, and each edge is selected among a very large number of candidates. Thus, the algorithm is based on a certain amount of randomness in the construction: if at some point the construction fails, then all the edges are discarded and the procedure restarts from scratch. This justifies the name of "Randomized PEG", and ensures that a wide variety of solutions are explored.

3) Reduced probability of construction failure: When spanning the tree to its maximal depth, the first cycles that are created by the algorithm are locally optimal in the sense that they are of the largest possible size. However, as the procedure progresses, the construction problem becomes too constrained and eventually fails if the target girth is relatively high compared to the graph parameters. Our extensive tests show that by spanning the tree at a lower depth, we create smaller cycles at the beginning of the procedure and thus the choice of the edge is *not* locally optimal, but nevertheless the probability that the algorithm actually terminates if much higher.

B. The objective function

We consider in this section the general case where $N \geq N_g^{(d_v,d_c)}$, *i.e.* when the graph size N is large enough such that a (d_v, d_c) graph of girth g may exist. The set of candidates can

be potentially very large, especially at the beginning of the graph construction, and it becomes possible (and necessary) to discriminate among the multiple candidates.

We describe here the objective function that we used, which minimizes the number of created cycles. We would like to point out that other objective functions could be used complementarily: the minimization of other topological structures such as the number of created stopping sets, trapping sets *etc.* or the minimization of an ACE metric [6], as done in [7] for the construction of irregular graphs.

When the construction tree is spanned up to a maximal depth l_{max}, the objective function restricts the set of candidates $\bar{\mathcal{N}}_{s_j}^{l_{max}}$, as follows:

If there are candidates at depth l_{max}, then discard all the candidates that are not exactly at the depth l_{max}. By doing so, we only create cycles of size *exactly* l_{max}, and ensure that the diameter argument is fulfilled

2- For each candidate c_j, compute $nbCycles_j$, the number of cycles that would be created if c_j is selected. Discard all candidates that would create more than $min_j(nbCycles_j)$.

3- Compute ${d_c}^{min}$, the lowest degree of all remaining candidates. Discard all candidates with current degree $d_c > {d_c}^{min}$

At this point, the algorithm randomly samples among the remaining candidates.

C. Refinement for spanning the tree

For a given target girth g_t, the diameter argument does not hold anymore for lengths N such that $N_{g_t}^{(d_v,d_c)} < N < N_{g_t+2}^{(d_v,d_c)}$. In that case, the diameter may be larger than $g/2$, and we propose an alternative strategy by introducing a gap variable: we span the tree up to a maximal depth $l_{max} = g_t + gap$.

At the beginning of the construction, cycles of size larger than $g_t + gap$ are created. Each time that it is no longer possible to add any edge, we decrease the value of gap, and therefore allow to create smaller cycles. At some point, we span the tree only up to a depth $l = g_t$, and only at this point the algorithm starts creating cycles of size g_t.

This technique, coupled with the objective function described in the previous section, allows to minimize the multiplicity of the girth, *i.e.* the number of cycles length g_t. It is not necessary for the simulations presented in the next section, but leads to a better LDPC code design when $d_v \geq 3$.

$g \backslash d_c$	3	4	5	6	7	8	9	10	$\cdots$	50
6	6^* [6]	10^* [10]	15^* [15]	21^* [21]	28^* [28]	36^* [36]	45^* [45]	55^* [55]	$\ldots^*$	1275^* [1275]
8	9^* [9]	16^*[20]	25^* [35]	36^* [48]	49^* [70]	64^* [116]	81^* [162]	100^* [230]	$\ldots^*$	2500^* [???]
10	15^* [18]	$38^{(34)}$[42]	$90^{(65)}$ [110]	$189^{(111)}$ [225]	$385^{(175)}$ [441]	$728^{(260)}$ [812]				
12	21^* [27]	52^* [104]	$105^{*\dagger}$ [380]	$186^{*\dagger}$ [966]						
14	36^* [36]	260 [292]								
16	45^* [72]	$160^{*\dagger}$ [850]								
18	$114^{(69)}$ [150]									
20	$201^{(93)}$ [285]									
22	$447^{(141)}$ [558]									

TABLE I

FOR VARIOUS VALUES OF GIRTH g AND VARIOUS VALUES OF CHECKNODE DEGREE d_c, WE REPORT THE SMALLEST GRAPH SIZE N SUCH THAT THE RANDPEG ALGORITHM COULD CONSTRUCT A REGULAR $(2, d_c)$ GRAPH OF GIRTH g.

IV. PERFORMANCE OF THE RANDPEG ALGORITHM FOR $d_v = 2$ GRAPHS

A. Design of ultra-sparse graphs

In table I we report, for different values of d_c and g, the smallest value of N such that the RandPEG algorithm could construct a regular $(2, d_c)$ graph of girth g. When this value achieves the lower bound $N_g^{(2,d_c)}$, we indicate so by super-scripting with a star ($*$), and the corresponding graph defines a $(d_c, \frac{g}{2})$-cage. Otherwise the value of the lower bound $N_g^{(2,d_c)}$ is super-scripted with parenthesis. Some values are super-scripted with a dag, which means that the RandPEG was initialized with a tree for these constructions. For comparison, the value of N such that the standard PEG algorithm could construct the corresponding graph is reported in square brackets.

For all values of d_c that we tested up to 50, the RandPEG successfully constructs cages for target girths $g = 6, 8$. Moreover, for lower values of $d_c = 3, 4$ the algorithm successfully constructs graphs of girth up to 16 that achieve the lower bound. The corresponding graphs are available on [8].

B. Application to the design of NB-LDPC codes

We now illustrate the interest of our algorithm for the design of non-binary LDPC codes. We designed two codes of rate one-half, with $(2, 4)$ graphs of size $N = 160$. For this graph setting the regular PEG algorithm constructed a graph of girth 12, whereas the RandPEG constructs a cage of girth 16. For both graphs, we optimized the non-binary coefficients in $GF(64)$ according

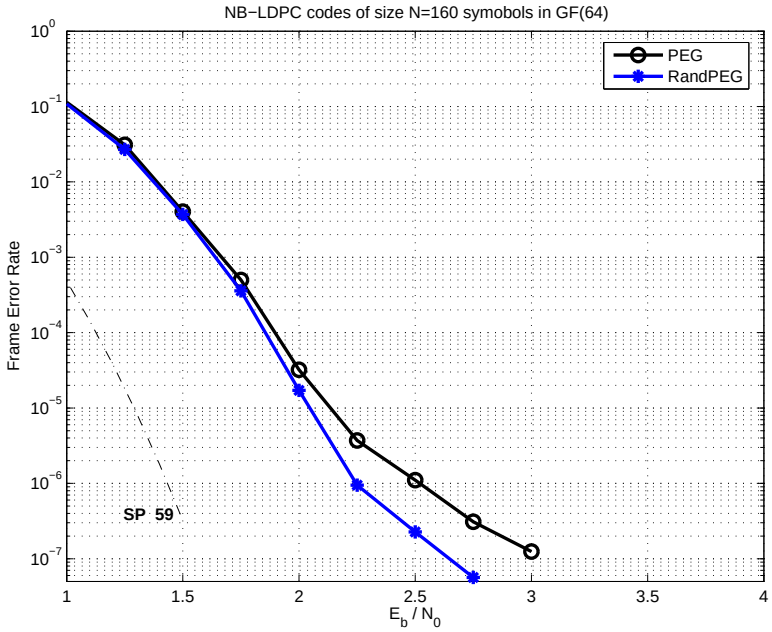

Fig. 1. Performance comparison for the design of non-binary LDPC codes: two codes whose underlying Tanner graphs were constructed with respectively the PEG and RandPEG algorithm are simulated over a BIAWGNC.

to the method described in [2], and simulated the resulting codes on a binary input additive white gaussian noise channel (BIAWGNC). The simulation results on Fig. 1 show that for ultra-sparse non-binary LDPC codes, a graph with better girth properties performs better in the error floor region, by inducing better spectrum and minimum distance properties [2].

C. Girth multiplicity

One important property that does not appear in Table I is the multiplicity of the girth, *i.e* the number of cycles with length equal to the girth. The multiplicity of the girth can be extremely important if the graph is used for designing (binary or non-binary) LDPC codes. We designed regular $(3, 6)$ binary LDPC codes of size $N = 504$ and $N = 1008$. All the codes were of girth 8, but for $N = 504$, the PEG code had a girth multiplicity of 808, whereas the RandPEG code had a multiplicity of only 452. For $N = 1008$, the PEG code had a girth multiplicity of 167, whereas the RandPEG code had a multiplicity of only 31. Simulation results show that the RandPEG codes perform better that the PEG codes.

REFERENCES

[1] H. Song, J. Liu, and B.V. Kumar, "Large girth cycle codes for partial response channels," *IEEE Trans. Magn.*, vol. 40, no. 4, pp. 3084–3086, July 2004.

[2] C. Poulliat, M. Fossorier, and D. Declercq, "Design of regular $(2, d_c)$ LDPC codes over GF(q) using their binary image," *accepted for publication in IEEE Trans. Commun.*, 2007.

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

[4] N. Biggs, "Constructions for cubic graphs with large girths," *The electronic journal of Combinatorics*, vol. 5, no. 1, 1988.

[5] R. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," *Proc. of ISTA*, 2001.

[6] T. Tian, C. Jones, J. Villasenor, and R. Wesel, "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Commun.*, vol. 52, no. 2, pp. 1242–1247, Aug. 2004.

[7] H. Xiao abd A. H. Banihashemi, "Improved progressive-edge-growth (peg) construction of irregular LDPC codes," *IEEE Commun. Lett.*, vol. 8, no. 12, pp. 715–717, Dec. 2004.

[8] "David declercq's homepage," http://perso-etis.ensea.fr/∼declercq/graphs.php.