Randomized Progressive Edge-Growth (RandPEG)

Auguste Venkiah, David Declercq, Charly Poulliat
ETIS, CNRS, ENSEA, Univ Cergy-Pontoise
F-95000 Cergy-Pontoise
email:{venkiah,declercq,poulliat}@ensea.fr

***Abstract*— The Progressive Edge-Growth (PEG) construction is a well known algorithm for constructing bipartite graphs with good girth properties. In this paper, 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. First, we consider regular column-weight two graphs, which are a class of graphs used for the design of non-binary LDPC codes: for a given target girth g_t, this new instance of the PEG algorithm allows to construct graphs with the minimal size such that a graph of girth g_t exists, which is the best result one might hope for. We illustrate the interest of such minimal constructions with simulation results. Then, we illustrate the usefulness of our algorithm for constructing regular binary LDPC codes that perform well in the error floor region.**

I. Introduction

In the limit of very large block lengths, the concentration theorem [1] ensures that under iterative decoding, the performance of randomly constructed Low Density Parity Check (LDPC) codes is very close to their asymptotic behavior. However, for small to moderate block lengths, random constructions perform rather poorly, because of shorts cycles in the associated Tanner graph. In that case, the performance of a code is closely related to the girth of the graph, which is defined as the size of the shortest cycle. The motivation behind having high girth LDPC codes is twofold. Firstly, the iterative belief propagation (BP) decoder tends to the Maximum A Posteriori (MAP) decoder and thus behaves better in the waterfall region. Secondly, an LDPC code with few short cycles has less chances to exhibit decoding failures caused by pseudo-codewords than an LDPC code with many short cycles. Therefore, a high girth tends to improve the performance in the error floor region.

The importance of constructing graphs with high girth is particularly important for non-binary (NB) LDPC codes. The BP decoder and its simplified versions have shown to achieve very good performance on non-binary GF(q)-LDPC codes with $d_v = 2$ and with high order Galois fields [2], [3]. For these ultra-sparse codes, it is crucial to focus on the girth properties of the underlying Tanner graph [4].

Large girths are obtained by suitably constructing the underlying Tanner graph, and a construction based on a progressive edge-growth (PEG) of the graph was proposed in [5], which results in LDPC codes that have higher girths compared to pre-existing random LDPC code construction techniques. The PEG algorithm has been extended to improve the decoding performance of the LDPC codes [6], but this method, based on the use of the ACE metric [7], only concerns irregular LDPC codes.

In this paper, 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.

The remainder of this paper is organized as follows: we first briefly review the PEG algorithm to introduce the notations in section II, and then present in section III our contributions in detail. In section IV, we illustrate the usefulness of our algorithm with simulation results.

II. Notations and definitions

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 (d_v, d_c) regular graph of girth g exists. This lower bound can be easily computed by using the results of [5, lemma 3], and is known *not* to be tight when $d_v = 2$, for $g \geq 18$ [8].

The original PEG algorithm [5] 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. 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_c. We use the same definition as in [5] for the depth. 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 assigned. 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.

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

[0]This work was supported by the ANR under Grant ANR 05-RNRT-019-04

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. Selecting a candidate that is at a depth l from node s_j creates a cycle of length $2l+2$.

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. In the sequel, we only consider the construction of regular (d_v, d_c) graphs, in order to compare to the known bounds for regular graphs. We emphasize the fact that this limitation concerns only our study, *not* the RandPEG algorithm itself.

III. The Randomized-PEG Algorithm

In this section, we describe our contributions in details. 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 that we use 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 an objective function to discriminate among the numerous edge candidates. 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 that case, all the edges in the graph are discarded and the algorithm starts from scratch. Similarly to Monte Carlo approaches, the algorithm runs many times and stores the best graph.

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 [5], is suggested for the construction of long codes where it would be computationally expensive to build the whole 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 $d = g_t/2$ [9]. Therefore in that case, the tree *must* be spanned up to a maximal depth $l_{max} = (g_t-2)/2$, 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 $l = (g_t-2)/2$ 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 is 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, as done in [6] 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:

1- 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 $d_c >$

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

$d_c{}^{min}$

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 $l_{max} = (g_t + gap - 2)/2$.

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 $gap = 0$, then we span the tree only up to a depth $l = (g_t - 2)/2$, 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. A typical value of $gap = 2$ is used for most constructions of regular codes.

Algorithm 1: RandPEG algorithm

Data: g_t, N, $\{(d_{s_j})_{j=1:N}\}$, w_{max}, gap
Result: $G = (V, E)$
$w \leftarrow 0$; /* number of trials for the graph construction */
BEGIN $E \longleftarrow \emptyset$; $gap \leftarrow 2$
for $j \leftarrow 1$ **to** N **do**
  for $k \leftarrow 1$ **to** d_v **do**
TRY SpanTree within maximal depth $(g_t + gap - 2)/2$
  if $\bar{\mathcal{N}}_{s_j}^{g_t+gap} = \emptyset$ **then**
    if $gap > 0$ **then**
      $gap = gap - 2$
      goto TRY
    else
      if $w < w_{max}$ **then**
        w++
        goto BEGIN
      else
        Algorithm FAILS
  else
    Apply Objective Function (section III-B)
    Randomly select a candidate: $E_{s_j}^k \leftarrow$ edge (s_j, c_j)
StoreGraph

IV. Performance of the RandPEG algorithm

A. Design of ultra-sparse graphs (d_v=2)

For a given graph setting and a given target girth, there exists a minimal size for the graph such that a graph of girth g_t exists, which is often given in terms of a lower bound. We call *minimal graph* a graph that has a size that achieves the lower bound. In table I we report the smallest value of N such that the RandPEG algorithm could construct a regular $(2, d_c)$ graph of girth g, for different values of d_c and g. When this value achieves the lower bound $N_g^{(2,d_c)}$, we indicate so by superscripting with a star $(*)$, thus indicating that the graph is minimal. Otherwise the value of the lower bound $N_g^{(2,d_c)}$ is superscripted 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 minimal graphs 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 can be found on [10].

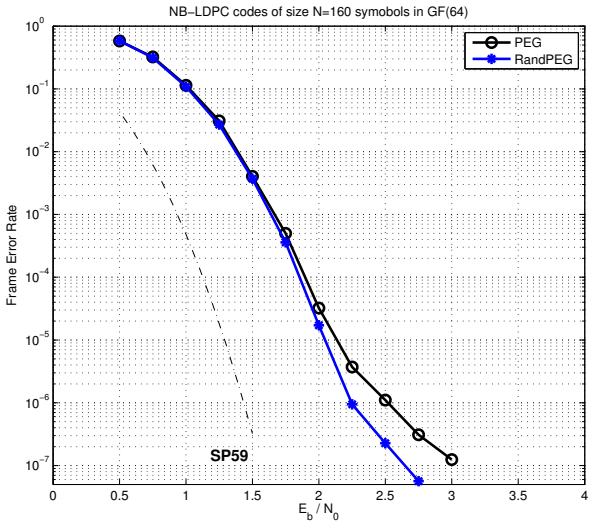

Fig. 1. Performance comparison for two non-binary LDPC codes: two codes whose underlying Tanner graphs were constructed respectively with the PEG and RandPEG algorithm are simulated over a BIAWGNC. The 'SP59' label denotes the Sphere Packing bound of 1959 [11].

B. Ultra-sparse graphs for NB-LDPC codes

We now illustrate the interest of our algorithm for the design of ultra-sparse non-binary LDPC codes, also called cycle Tanner-graph codes 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 standard PEG algorithm constructed a graph of girth 12, whereas the RandPEG constructs a minimal graph of girth 16. For both graphs, the non-binary coefficients in GF(64) were optimized according to the method described in [4], and the resulting codes were simulated over a binary-input additive white Gaussian noise channel (BIAWGNC) with BP decoding. Frame error rates are estimated with 100 frames in error. The simulation results reported in figure 1 show that

$g \backslash d_c$	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 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. (SEE SECTION IV-A FOR EXPLANATIONS)

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 [4]

C. Regular (3,6) codes

By minimizing the girth multiplicity, the RandPEG algorithm indirectly reduces the number of stopping/trapping sets, and therefore, we expect the resulting LDPC codes to perform better in the error floor region. We used the RandPEG algorithm to construct regular (3,6) LDPC codes, of sizes $N = 504$ and $N = 1008$. All graphs where of girth 8, but the graphs designed with the RandPEG had less cycles of length 8. The graph of size $N = 504$ constructed with the standard PEG algorithm had 808 cycles of length 8, whereas the graph constructed with the RandPEG had only 452 cycles of length 8. The graph size $N = 1008$ constructed with the standard PEG algorithm had 167 cycles of length 8, whereas the graph constructed with the RandPEG had only 31 cycles of length 8. This illustrates the importance of the objective function.

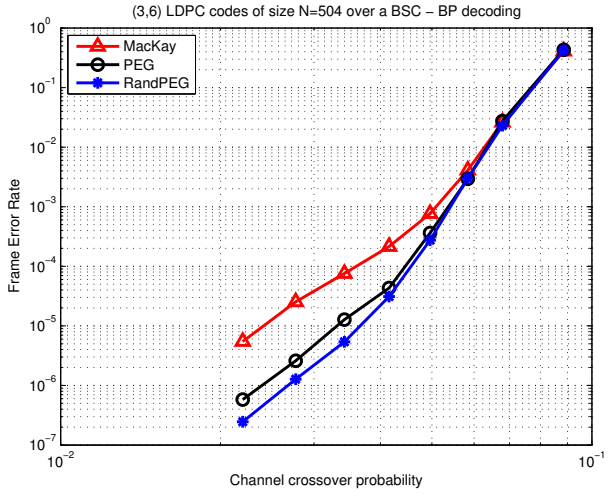

Fig. 2. Regular (3,6) LDPC codes of size $N = 504$ simulated on a BSC channel, and decoded with BP decoding

The resulting codes were simulated on a Binary Symmetric Channel (BSC) and decoded with BP decoding. We compared the PEG and RandPEG constructions to the codes optimized by MacKay [12]. The results are reported in figures 2 and 3. It

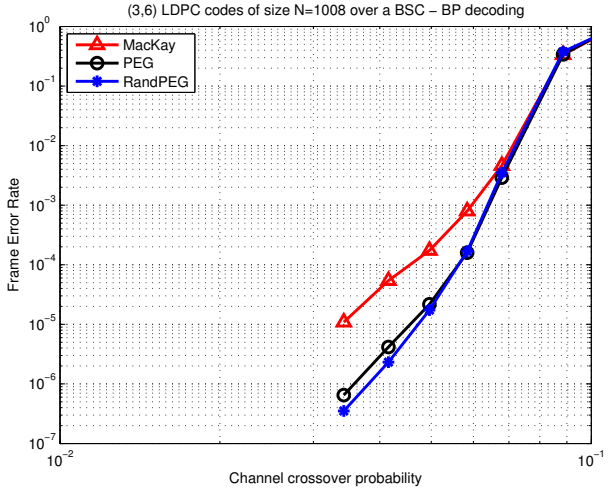

Fig. 3. Regular (3,6) LDPC codes of size $N = 1008$ simulated on a BSC channel, and decoded with BP decoding

appears that indeed the minimization of the girth multiplicity is important for improving the performance of LDPC codes in the error floor region.

V. CONCLUSION

We have presented some modifications to the PEG algorithm based on 1) the introduction of a larger amount of randomness in the construction, 2) a different manner for spanning and using the construction tree of the original PEG algorithm, and 3) the introduction of an objective function that minimizes the girth multiplicity. This instance of the PEG algorithm that we called RandPEG greatly improves the achievable girth for given graph parameters, especially for column-weight two graphs ($d_v = 2$) where it allows to construct minimal graphs. With simulation results, we illustrated the interest of these minimal constructions for the design of ultra-sparse non-binary LDPC codes, and showed that the objective function also allows to construct binary regular LDPC codes that perform better in the error floor region.

REFERENCES

[1] T. J. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message-passing decoding," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[2] X.-Yu Hu and E. Eleftheriou, "Cycle tanner-graph codes over GF(2^b)," in *Proc. of IEEE ISIT 2003, Japan, Yokohama*, 2003, p. 87.
[3] D. Declercq and M. Fossorier, "Decoding algorithms for non binary LDPC codes over GF(q)," *IEEE Trans. Commun.*, vol. 55, no. 4, pp. 633–643, Apr. 2005.
[4] 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.
[5] X.-Yu 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.
[6] 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.
[7] 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.
[8] N. Biggs, "Constructions for cubic graphs with large girths," *The electronic journal of Combinatorics*, vol. 5, no. 1, 1988.
[9] R. Tanner, D. Sridhara, and T. Fuja, "A class of group-structured LDPC codes," in *Proc. of ISTA*, 2001.
[10] "David declercq's homepage," http://perso-etis.ensea.fr/~declercq/graphs.php.
[11] C.E. Shannon, "Probability of error for optimal codes in a gaussian channel," *Bell System Technical Journal*, vol. 38, pp. 611–656, May 1959.
[12] "Encyclopedia of sparse graph codes," http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.