# Progressive Edge-Growth in IRA Codes

Mauro Q. Lustosa e Weiler A. Finamore

***Resumo*— Este artigo aborda o uso do algoritmo Progressive Edge-Growth (PEG) para a construção de códigos IRA. Os códigos IRA (Irregular Repeat-Accumulate) são códigos irregulares baseados em matrizes esparsas que permitem codificação em tempo linear para canal AWGN (ruído aditivo gaussiano branco) ou canais com apagamento. O algoritmo Progressive Edge-Growth, elaborado originalmente para códigos LDPC foi aplicado para construção de grafos de códigos IRA visando a redução do número de ciclos e melhores propriedades de distância mínima. Uma avaliação da versão aprimorada do algoritmo PEG é apresentada juntamente a uma proposta de método alternativo e comparação dos resultados.**

***Palavras-Chave*— IRA, PEG, LDPC, códigos em grafos, condicionamento de ciclos.**

***Abstract*— This article addresses the use of the Progressive Edge-Growth (PEG) algorithm for use with IRA codes. IRA codes are channel codes on sparse-graphs that allow linear-time encoding, with near capacity performances on the AWGN, and Binary Erasure channels. Originally conceived for LDPC codes, the PEG algorithm was applied to the construction of IRA graphs with the goal of reducing the occurrences of short cycles and improve the distance properties. An evaluation of the *enhanced look-ahead* variation of the PEG algorithm is presented along with an alternative method and a comparison of practical results.**

***Keywords*— IRA, PEG, LDPC, Sparse Graph Codes, girth conditioning.**

## I. Introduction

Irregular Repeat-Accumulate were introduced by Hui Jin, Khandekar & McEliece [1] in 2000, providing a family of codes that use linear-time encoding and iterative decoding while communicating reliably at rates close to channel capacity. The authors proved these codes can achieve channel capacity on the binary erasure channel with remarkably good performance on the AWGN channel.

IRA codes are a particular case of the LDPC codes, as they work by adding parity bits to very large blocks. The code is represented by a matrix containing relatively few non-zero elements, i.e. a *sparse matrix*. A simple manner for describing such a matrix is by keeping track only of the positions (and values, which are not implicit in the non-binary case) of the few non null elements. While classic LDPC parity check matrices don't necessarily follow any visible pattern, the non-systematic part of IRA matrices is composed of the main diagonal and the subdiagonal filled with ones, as shown in (1), while all other elements are null. For this characteristic format, IRA codes are also called *Staircase Codes*.

Eng. Mauro Lustosa¸ Prof. Weiler A. Finamore CETUC, PUC–Rio, Rio de Janeiro, Brasil, E-mails: lustosa@cetuc.puc-rio.br, weiler@cetuc.puc-rio.br.
Este trabalho contou com apoio do CNPQ.

$$\mathbf{H}_{\text{IRA}} = \left( \overbrace{\begin{matrix} 1 & & 1 \\ & 0 & 1 \\ 0 & 1 & \\ 1 & & 0 \end{matrix}}^{\text{systematic}} \quad \overbrace{\begin{matrix} 1 & & \\ 1 & 1 & 0 \\ & 1 & \ddots \\ 0 & 0 & \ddots \; 1 \end{matrix}}^{\text{non-systematic}} \right) \tag{1}$$

The parity-check matrices represent a bipartite graph, which we call the Tanner Graph (after R. Michael Tanner). A bipartite graph is composed of

- two disjoint sets of nodes;
- edges that join only nodes in distinct sets;

The two disjoint sets are labeled **variable-nodes**, whose values are given by the symbols that compose a codeword; and **check-nodes** that impose parity constraints on the variable-nodes at the opposite ends of its incident edges. Variable nodes either belong to the class of **information-nodes** ($u_\kappa \in \mathcal{U}$, $\kappa \in \{1, 2 \ldots, k\}$), that carry the original message; or **parity nodes** ($w_\mu \in \mathcal{W}$, $\mu \in \{1, 2 \ldots, m\}$) that carry the redundant bits. Variable nodes may also be labeled $c_j \in \mathcal{N}$, $j \in \{1, 2 \ldots, n\}$ (where $\mathcal{N} \triangleq \mathcal{U} \cup \mathcal{W}$ and $n = m + k$) in contexts where no distinction needs to be made between parity and information variable-nodes. Check-nodes are denoted $z_i \in \mathcal{M}$, $i \in \{1, 2 \ldots, m\}$. See Figure (1) for an example of the Tanner graph of an IRA code (in the example, the graph does not include nodes of degree three or four).

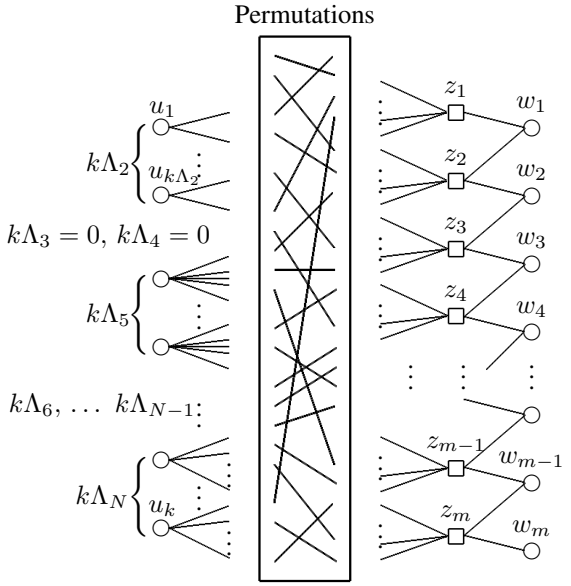


Fig. 1. A sketch of the Tanner graph of an IRA code: $\Lambda_i$ is the fraction of information nodes with degree $i$ in $\mathcal{U}$; ◯ are variable-nodes; □ are check-nodes

The degree of a node is given by its number of adjacent nodes (or incident edges). Graphs where all nodes in each class have the same degree are called *regular graphs*. During decoding, variable-nodes and check-nodes exchange *messages* through their edges. While a variable-node with a high degree receives more messages from neighboring check-nodes, these messages are more useful when they come from a check-node with a lower degree. The trade-off between these interests resulted in the use irregular degree distributions (i.e. a pre-defined statistical distribution of degrees in the graph's nodes) which yield graphs that perform better than regular ones [2]. IRA codes are described in [1] as right regular (i.e. all check-nodes have the same degree).[1]

The iterative decoding of IRA (and LDPC) codes uses the *message passing algorithm* [4][5]. The structure of the graph determines how efficiently the decoder will perform, and in order to optimize the decoding capabilities of a graph we should optimize three properties of the graph: the degree distribution, the edge permutation and the block length.

The work presented in this document does not involve optimizing degree distributions, focusing on graph construction methods for relatively short block-lengths ($k = 500, 1000, 2000$ bits). Figure 1 depicts a graph of message block-length $k$, with $m$ additional parity nodes. The box between the variable-nodes and parity nodes represents the random permutations of the edges in the graph. The edges joining the parity nodes $w_j$ and check-nodes $z_j$ are pre-defined. Parity nodes ensure that all the nodes directly connected to the check-nodes will sum to zero, and their value is given by

$$w_j = w_{j-1} \oplus \sum_{l \in \mathcal{U}_{z_j}} u_l, \tag{2}$$

where $\mathcal{U}_{z_j}$ denotes the information-nodes joined with check-node $z_j$.

The message passing algorithm relies on products of individual probabilities to factor joint probabilities. These joint probabilities are then marginalized into probabilities of single variables and the process repeats in a recursive manner. The values estimated by this algorithm become inaccurate when probability products of dependent variables are used as joint probabilities. This dependency is very strong when two sets of variables nodes $\mathcal{N}_{z_k}$ and $\mathcal{N}_{z_l} \in \mathcal{N}$ — each set composed of the neighborhoods of check-nodes $z_k$ and $z_l$, respectively — contain more than one element in common. In a Tanner Graph, this coincidence is called a length-four cycle. In fact, these are cycles that may occur in all even numbered lengths [2] depending on how the edges are drawn, and are inevitable for graphs without degree-one nodes. However, these distorted probabilities might not prevent successful decoding if we manage to make these cycles long enough. Therefore we want to maximize the graph's *girth*, it's shortest cycle length.

[1] see [3] for exceptions to this rule

[2] a length-two cycle would mean one check-node being joined with one variable node by two edges, which cancels the variable-node's influence on this check

## II. Progressive Edge-Growth Algorithm

The Progressive Edge-Growth algorithm, first presented in [6] for construction of sparse graphs for LDPC codes, was used in this work to design IRA codes following the degree distributions presented in [1]. We also modified the look-ahead enhanced version of the algorithm to verify its effects on the decoder's performance.

Progressive Edge-Growth is a method for the construction of Tanner Graphs proposed in [6]. It attempts to make the graph's girth to be made as long as possible by maximizing the local girth (the shortest cycle including the current variable-node) at each new edge that is placed. The PEG algorithm proceeds along the variable-nodes, in order of increasing degree, in a manner that each new edge has as little impact on the graph's girth as possible.

We can observe all the paths (sequence of adjacent edges) that include one particular node by expanding it as the root of a tree using the current edges as in Figure 2. The check-nodes that are joined with the root are said to be in depth 0, the other variable-nodes adjacent to the check-nodes in depth 0 are roots to the subsequent subtree. Evidently, every element that is present in a tree reaches all others through the current edges, and a cycle of length $2(l+1)$ is completed if an additional edge is drawn between the root and a check-node that appears for the first time in depth $l$. The set of check-nodes reached by node $u_j$ at depth $l$ is denoted $\mathcal{M}^l_{u_j}$, its complementary check-nodes being in $\bar{\mathcal{M}}^l_{u_j}$.

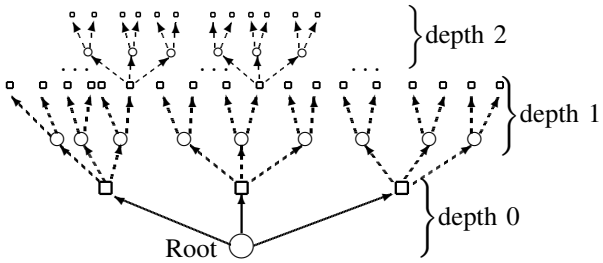


Fig. 2. An example of a tree drawn to depth 2.

IRA codes are peculiar in the sense that all check-nodes are already connected before we start drawing the edges, meaning there are no two disjoint subgraphs in an IRA graph. Once a variable-node enters the graph by receiving its first edge, every additional edge incident on this same node shall complete a cycle.

The PEG algorithm inserts information-nodes ($u_j$) into the graph, by joining it with other nodes, in order of increasing degree ($d_{u_j}$). $E$ represents the complete set of edges that composes the graph, $E_{u_j}$ is the sub-graph of depth zero rooted in $u_j$ and $E^{\kappa}_{u_j}$ is the $\kappa^{th}$ edge in $E_{u_j}$.

### A. Look-Ahead enhanced version

While the PEG algorithm clearly maximizes each information-node's local girth at each stage, there is no certainty on whether the graph will retain its girth in later stages of the algorithm. As an enhancement of PEG, [6] proposes the Look-Ahead enhanced version that verifies the impact that each new edge would cause on the graph.

**Algorithm 1** Progressive Edge-Growth Algorithm for IRA Codes

**for** $j = 1$ to $k$ **do**
  **for** $\kappa = 0$ to $d_{u_j} - 1$ **do**
    **if** $\kappa = 0$ **then**
      $E^0_{u_j} \longleftarrow \mathbf{edge}(z_i, u_j)$, where $E^0_{u_j}$ is the first edge incident to $u_j$ and $z_i$ is a randomly picked check-node among those with the lowest degree under the current graph setting $\bigcup_{i=0}^{j-1} E_{u_i}$.
    **else**
      expand a tree from information-node $u_j$ up to depth $l$ under the current graph setting such that $\bar{\mathcal{M}}^l_{u_j} \neq \emptyset$ but $\bar{\mathcal{M}}^{l+1}_{u_j} = \emptyset$, then $E^\kappa_{u_j} \longleftarrow \mathbf{edge}(z_i, u_j)$, where $E^\kappa_{u_j}$ is the $\kappa^{th}$ edge incident to $u_j$ and $z_i$ is a check-node randomly picked from a subset of $\bar{\mathcal{M}}^l_{u_j}$ containing only check-nodes with the lowest degree under the current graph setting.
    **end if**
  **end for**
**end for**

Instead of choosing randomly from the eligible candidates as defined in algorithm 1, the Look-Ahead enhanced algorithm considers hypothetical partial graphs for the cases where each candidate receives the new edge. The hypothetical partial graphs are then compared in terms of the maximum depth to which the local tree would expand before reaching all check-nodes. The suggested strategy is to choose the candidate check-node for which the tree expands to the highest depth.

## III. Experimental Results

We build IRA codes using the five distributions provided by [1] with $k = \{500, 1000, 2000\}$ and tested them on the additive white gaussian noise (AWGN) channel for performance evaluation. With the goal of identifying the possible causes for unexpected results we also count the number of length-four cycles in each graph, and generate codewords from low-weight input messages to have a sample of low weight codewords. The low weight codewords give an insight into the code's distance properties.

### A. Rate 1/3

We compare the three approximately distributions from [1] with $k = 1000$ and rate approximately $1/3$ using the standard PEG algorithm (`PEG-ST`). The table I gives the regular check-node degree, the mode in the edge degree distribution,[3] the $E_b/\mathcal{N}_0$ threshold, and in the bottom rows are the number of length four cycles and minimum distance of the codes we generated using the PEG algorithm in its standard version.

We estimate the performance of these codes through simulation, using ten block error events per point to find the approximated bit error probabilities. At very low bit error probabilities approaching $10^{-7}$ accurate estimations become more time-consuming and fewer samples are collected (resulting in more

[3]This is not the same as the node degree distribution, refer to [7] for the unique relation between the two.

TABLE I
PEG-ST: K=1000, $rate \approx \frac{1}{3}$

| Distr. | #1 | #2 | #3 |
|---|---|---|---|
| $a$ | 2 | 3 | 4 |
| mode ($\lambda_i$) | $\lambda_6 \approx 0.64$ | $\lambda_{13} \approx 0.49$ | $\lambda_{27} \approx 0.45$ |
| max. degree | $\lambda_6 \approx 0.64$ | $\lambda_{13} \approx 0.49$ | $\lambda_{28} \approx 0.45$ |
| Rate | 0.333364 | 0.333223 | 0.333218 |
| $\left(\frac{E_b}{\mathcal{N}_0}\right)_{thr.}$ [dB] | 0.190 | -0.25 | -0.371 |
| # 4-cycles | 0 | 0 | 6 |
| $d_{min}$ | 29 | 31 | 31 |

significant error-margins). In repeated experiments, however, distributions #2 and #3 show consistently tied performance, ranking better than distribution #1 as predicted by density evolution in [1]. The crosses in Figure 3 mark the upper bound to the actual bit-error probability with confidence interval 0.05.

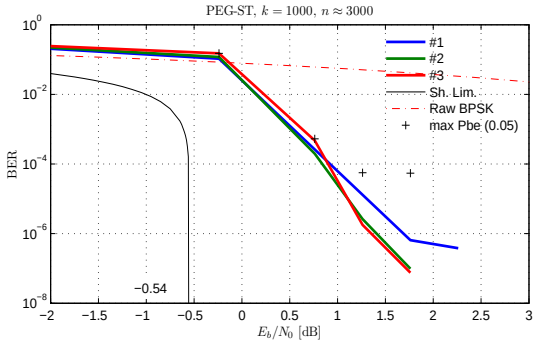


Fig. 3. Code performance using degree distributions #1, #2 and #3

Table I shows that the codes built from distributions #2 and #3 were found to have the same minimum distances, but distribution #3 is more prone to forming cycles due to higher node-degrees. Fortunately these cycles tend to involve only the variable nodes of higher degrees, which rely on messages from many additional nodes for decoding. Nevertheless, length-four cycles can be catastrophic when they include nodes of degree two.

It is possible to observe the distance properties of a code by encoding low-weight messages and plotting the hamming weight of the resulting codewords. Figure 4 shows one of such plots, where we can observe the minimum weight (in logarithmic scale, polar plot) and the minimum weight of the densest 99% and 95% of these codewords.

The space between the two innermost circles depends on how many low weighted error patterns could take a received vector to the nearest valid codeword, causing undetected errors. Codes obtained from distributions #2 and #1 showed very similar properties to the ones in Figure 4.

The degree distributions provided for codes with rate approximately $\frac{1}{2}$ are briefly described in table II. These distributions include higher degrees than the previous three, although the highest degrees are not the mode, and the distribution #4 does not have information-nodes with degree two. Although the decoding thresholds obtained through density evolution favor distribution #5 by 0.078dB, the alleged better distance properties of distribution #4 keep the decoder from deciding

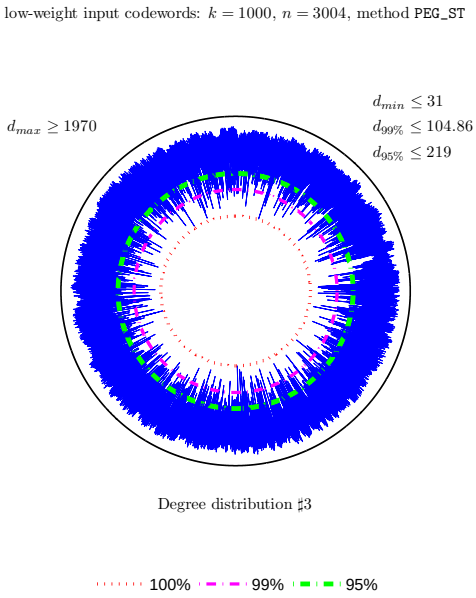


Fig. 4. Low weight codewords, distribution #3 $n = 2000$

towards neighboring codewords, thus achieving lower error-floors [4] than distribution #5. Figures 5 and 6 show the undetected errors that affect the performance of one distribution but not the other.

TABLE II
PEG-ST K=1000,$rate \approx \frac{1}{2}$

| Distr. | #4 | #5 |
|---|---|---|
| $a$ | 8 | 8 |
| mode ($\lambda_i$) | $\lambda_{12} \approx 0.33$ | $\lambda_7 \approx 0.22$ |
| Rate | 0.50227 | 0.497946 |
| $\left(\frac{E_b}{N_0}\right)_{thr.}$ [dB] | 0.344 | 0.266 |
| # 4-cycles | 2895 | 3969 |
| $d_{min}$ | 12 | 18 |

Figures 7 and 8 show histograms with the weight of randomly generated valid codewords for codes obtained from distributions #4 and #5 and $k = 500$. The histograms resemble a Normal probability mass distribution but do not provide a good basis to determine with a good degree of certainty which code has the higher minimum distance. The random encoding of low-weight messages is, therefore, not an effective method for establishing the distance properties of LDPC codes, even for block lengths as short as 500.

### B. Enhanced Versions

The large number of length-4 cycles in the graphs generated from distributions #4 and #5 motivate the use of the look-ahead enhanced version of the PEG algorithm (PEG−LA) for better results. The complexity of the look-ahead enhanced version increases quadratically with the block-length, since the algorithm expands a new local graph for each check-node that is eligible for receiving an edge. For practical codes, this imposes a very high cost in computation time.

[4] see [8], for more on distance properties of LDPC codes

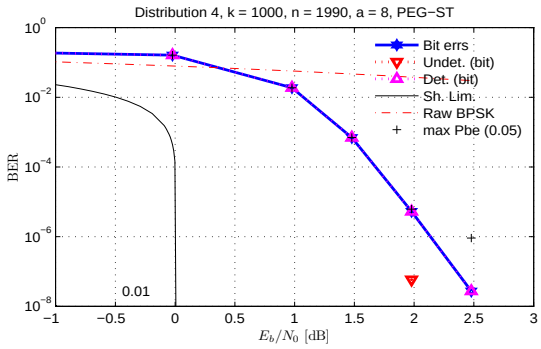


Fig. 5. Code performance using degree distribution #4

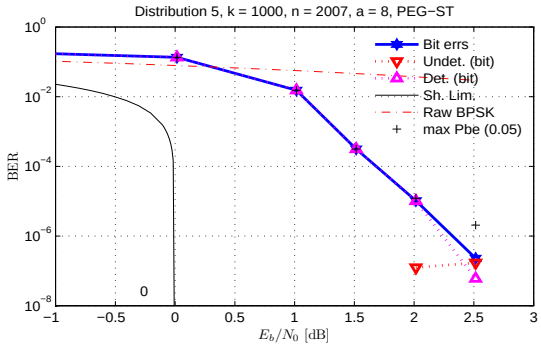


Fig. 6. Code performance using degree distribution #5

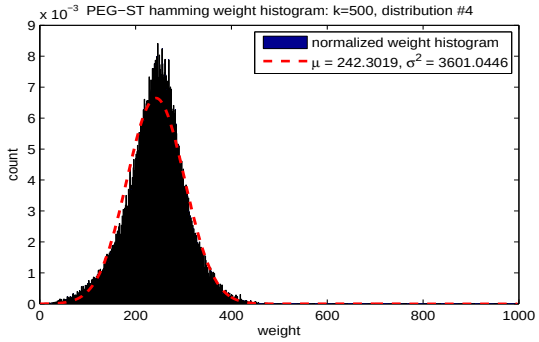


Fig. 7. histogram of low-weight codewords for distribution #4, $k = 500$

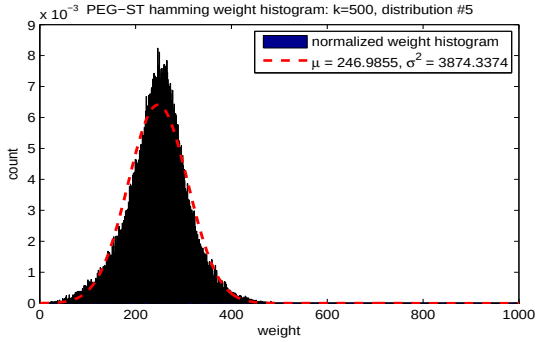


Fig. 8. histogram of low-weight codewords for distribution #5, $k = 500$

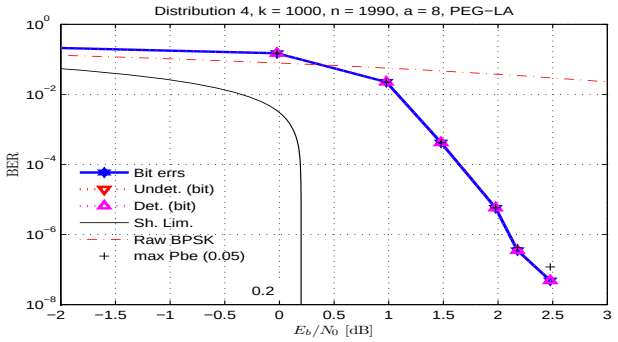


Fig. 9. Code performance using degree distribution #4 with the PEG look-ahead enhanced version

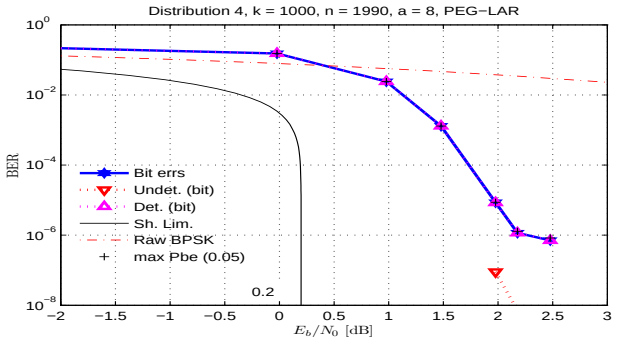


Fig. 10. Code performance using degree distribution #4 with the PEG look-ahead reversed version

The bit-error performance does not differ drastically between the standard PEG and the Look-Ahead enhanced method, but the latter does show an improved performance at high SNR. The criterium for deciding which check-node makes the best candidate for receiving a new edge in the `PEG-LA` method is noteworthy. Choosing the hypothesis that leads to the deepest tree may be a misleading strategy since a graph that expands as a tree with many depth levels is a graph with a large maximum cycle-length, but not necessarily a large girth. In fact, graphs with short girth will expand slowly as less new check nodes are reached at each subsequent depth level. Graphs with large girth will add more new check-nodes to the tree at each step during the tree expansion, reaching all check-nodes in fewer steps.

As an alternative strategy, the reverse criterium was adopted for a new version which we label `PEG-LAR` (PEG look-ahead reverse). The check node that reaches all others in *less* expansion steps is joined with the current variable node. The results of the simulated transmissions using the code generated with the `PEG-LAR` algorithm with distribution #4 and $k = 1000$ are in Figure 10. Since the complexity of the `PEG-LAR` algorithm is the same as in the `PEG-LA` and the results are clearly inferior, we conjecture that it is not a good alternative for the two other competitors.[5]

[5]In the article originally submitted in may 2009, the results were slightly favorable to the PEG-LAR algorithm. With additional simulations, it was established that the code built by the `PEG-LAR` method did not perform as well as the code that used the `PEG-LA`.

## IV. Conclusion

We applied the Progressive Edge-Growth algorithm for construction of IRA graphs using different degree distributions. For relatively short block-lengths we could obtain graphs without cycles for distributions with low degrees. As the maximum degree and the mode of the distribution approach values in the order of magnitude of the number of check-nodes it becomes impossible for the algorithm to generate a cycle-free graph. Codes with cycles can still achieve satisfactory results, although far from the theoretical bounds.

We investigated the effectiveness of the *look-ahead* enhanced version of the PEG algorithm, with poor results when considered the added complexity of this method. We adapted the *PEG look-ahead* algorithm to an original alternative version by reversing the decision criterium for the placement of edges in the enhanced algorithm. This version showed a decline in performance, while maintaining a high computational cost. Yet, there is no proof, to the author's knowledge, that the look-ahead enhanced PEG algorithm shall increase the graph's girth more efficiently than the standard version.

## Acknowledgements

The authors thank CNPQ for providing scholarship funds and Igor Kozintzev for making his LDPC soft-decision decoder available.

## References

[1] H. Jin, A. Khandekar, and R. McEliece, "Irregular repeat-accumulate codes," *International Conference on Turbo Codes*, 2000.

[2] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," *Proceedings of the 29th annual ACM Symposium on Theory of Computing*, 1997.

[3] S. J. Johnson and S. R. Weller, "Constructions for irregular repeat-accumulate codes," in *IEEE International Symposium on Information Theory*, pp. 179–183, IEEE, September 2005.

[4] D. MacKay, "Good error-correcting codes based on very sparse matrices," *Information Theory, IEEE Transactions on*, vol. 45, pp. 399–431, Mar 1999.

[5] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *Information Theory, IEEE Transactions on*, vol. 47, pp. 498–519, Feb 2001.

[6] X. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth tanner graphs," *IEEE Transactions On Information Theory*, 2005.

[7] T. J. Richardson, A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Transactions on Information Theory*, no. 2, pp. 619–637, 2001.

[8] T. Tian, C. Jones, J. D. Villasenor, and R. Wesel, "Construction of irregular ldpc codes with low error floors," in *IEEE International Conference on Communications*, 2003.