***This paper investigates the construction of random-structure LDPC (low-density parity-check) codes using Progressive Edge-Growth (PEG) algorithm and two proposed algorithms for removing short cycles (CB1 and CB2 algorithm; CB stands for Cycle Break).***

***Progressive Edge-Growth is an algorithm for computer-based design of random-structure LDPC codes, the role of which is to generate a Tanner graph (a bipartite graph, which represents a parity-check matrix of an error-correcting channel code) with as few short cycles as possible. Short cycles, especially the shortest ones with a length of 4 edges, in Tanner graphs of LDPC codes can degrade the performance of their decoding algorithm, because after certain number of decoding iterations, the information sent through its edges is no longer independent.***

***The main contribution of this paper is the unique approach to the process of removing short cycles in the form of CB2 algorithm, which erases edges from the code's parity-check matrix without decreasing the minimum Hamming distance of the code. The two cycle-removing algorithms can be used to improve the error-correcting performance of PEG-generated (or any other) LDPC codes and achieved results are provided. All these algorithms were used to create a PEG LDPC code which rivals the best-known PEG-generated LDPC code with similar parameters provided by one of the founders of LDPC codes.***

***The methods for generating the mentioned error-correcting codes are described along with simulations which compare the error-correcting performance of the original codes generated by the PEG algorithm, the PEG codes processed by either CB1 or CB2 algorithm and also external PEG code published by one of the founders of LDPC codes***

***Keywords: LDPC, low-density parity-check, PEG, progressive edge-growth, channel coding, Tanner graphs***

UDC 681
DOI: 10.15587/1729-4061.2021.225852

# INVESTIGATION OF RANDOM-STRUCTURE REGULAR LDPC CODES CONSTRUCTION BASED ON PROGRESSIVE EDGE-GROWTH AND ALGORITHMS FOR REMOVAL OF SHORT CYCLES

**Viktor Durcek**
*Corresponding author*
PhD*
E-mail: viktor.durcek@feit.uniza.sk
**Michal Kuba**
PhD*
**Milan Dado**
Professor, PhD*
*Department of Multimedia and Information-Communication Technologies
University of Zilina
Univerzitna str., 8215/1, Zilina, Slovakia, 010 26

*Received date 17.05.2021*
*Accepted date 17.07.2021*
*Published date 31.08.2021*

***How to Cite:*** *Durcek, V., Kuba, M., Dado, M. (2021). Investigation of random-structure regular LDPC codes construction based on progressive edge-growth and algorithms for removal of short cycles. Eastern-European Journal of Enterprise Technologies, 4 (9 (112)), 46–53. doi: https://doi.org/10.15587/1729-4061.2021.225852*

## 1. Introduction

Low-density parity-check codes are linear error-correcting block codes, which are characterized by their sparse parity-check matrices (usually means the number of ones in such a matrix is below 1–2 % for LDPC codes) and are able to perform close to the Shannon limit [1, 2]. An example of such matrix can be found in Fig. 1 along with its graphical representation, a Tanner graph. They were first introduced in 1960 [3]. They were however impractical to implement at the time and were forgotten until they were independently reinvented in the 1990's [1, 4, 5]. Their architecture is efficient and supports parallelism in decoding, computational simplicity, and various code rates. They can also employ several principles used in turbo codes to achieve high error-correcting performance [1].

In Table 1, a list of abbreviations used in this paper can be found.

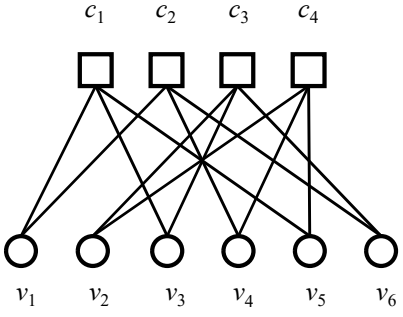
LDPC codes are used in a variety of applications, including satellite communications, Deep Space Network, Digital Video Broadcasting standards (DVB-S2, DVB-C2, DVB-T2), IEEE 802.11, IEEE 802.16e (WiMAX), LTE networks [6]. For the best error-correcting performance, LDPC codes are usually decoded by iterative soft-decision algorithms, e. g. sum-product algorithm (SPA) and its variations like min-sum algorithm (MSA). There also exist layered decoding approaches (row-layered decoding and column-layered decoding). Hard-decision or soft-decision encoding can be used. LDPC codes are divided into two categories: regular and irregular. Regular LDPC codes have constant column and row weight of their parity-check matrices (they have the same number of ones in every column and also the same number of ones in every row of the matrix). Column and row weight of an irregular LDPC code is not constant throughout the whole parity-check matrix [1, 7].

Progressive Edge-Growth (PEG) is an algorithm for computer-based design of random-structure LDPC codes. Its role is to generate a Tanner graph (a bipartite graph, which represents a parity-check matrix, as seen in Fig. 1) with as few short cycles as possible [7]. When looking at

Fig. 1, two groups of nodes can be seen: the square-shaped nodes are called check nodes (CN) and the circle-shaped nodes are called variable nodes (VN; sometimes also called symbol nodes) [8]. A cycle is a path which consists of unique edges and also starts and ends in the same node (the length of the shortest possible cycle in a Tanner graph is 4 edges). Short cycles in Tanner graphs of LDPC codes can degrade the performance of their decoding algorithm, because after a certain number of decoding iterations, the information sent through its edges is no longer independent [9].

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

*a*




*b*

Fig. 1. A parity-check matrix and its Tanner graph: *a* — parity-check matrix; *b* — Tanner graph

The proposed method of LDPC code construction consists of generating an LDPC code with the PEG algorithm and then using a loop-removing algorithm (either CB1 or CB2) to remove any excess short cycles from this LDPC code. Specific implementations of the PEG algorithm along with CB1 and CB2 loop-removing algorithms will be described below.

Table 1

List of Abbreviations

| Abbreviation | English meaning |
|---|---|
| BER | Bit Error Rate |
| BI-AWGN | Binary-Input Additive Gaussian White Noise |
| CB | Cycle Break |
| CN | Check Node |
| DVB-C | Digital Video Broadcasting (Cable) |
| DVB-S | Digital Video Broadcasting (Satellite) |
| DVB-T | Digital Video Broadcasting (Terrestrial) |
| $E_b/N_0$ | Energy per Bit to Noise Power Spectral Density Ratio |
| IEEE | Institute of Electrical and Electronics Engineers |
| LDPC | Low-Density Parity-Check |
| LTE | Long Term Evolution |
| MATLAB | Matrix Laboratory |
| MSA | Min-Sum Algorithm |
| PEG | Progressive Edge-Growth |
| QC-LDPC | Quasi-Cyclic Low-Density Parity-Check |
| SPA | Sum-Product Algorithm |
| VN | Variable Node |
| WiMAX | Worldwide Interoperability for Microwave Access |

## 2. Literature review and problem statement

LDPC codes created by the PEG algorithm are classified as random, because their parity-check matrices mostly lack structure, other than being a linear code [10]. Although LDPC codes constructed by this algorithm are among the best codes with large girth, their disadvantage is the high complexity of encoding and decoding, which is often too high for many hardware implementations [11, 12]. However, there exist LDPC codes with structured parity-check matrices. One type of these codes is called quasi-cyclic LDPC (QC-LDPC) codes. Their parity-check matrices comprise several submatrices called circulants (every row in a circulant matrix is a cyclic shift of their previous row). This structure lowers their complexity and makes them much more hardware-friendly. Although PEG LDPC codes themselves are often impractical for hardware implementations, there exist several methods for construction of QC-LDPC codes with the use of the PEG algorithm (while retaining the advantages of QC-LDPC codes) [13–18], e. g. by generating each submatrix of the final parity-check matrix using the PEG algorithm. In [13], the construction of PEG-QC-LDPC codes (type of LDPC codes with the advantage of lower memory requirements) than PEG codes is described together with modifications to also maximize its girth properties. The paper [14] describes how to introduce the quasi-cyclic property to non-binary LDPC codes based on the PEG algorithm, while retaining their good error-correcting performance (the authors of this paper state that the PEG algorithm is considered one of the most successful approaches for the construction of finite length LDPC codes), but it is concluded that this method requires further research because of short cycles which appear in smaller matrices. The paper [15] presents so-called QC-PEG-Root-Check-LDPC codes which perform well in block fading channels. The authors state that this is a unique approach because they have not found any similar publications which focus on these channel conditions. The paper [16] contains methods for constructing both regular and irregular quasi-cyclic LDPC codes with error-correcting benefits of progressive edge-growth codes. The authors state that their proposed codes exhibit BER performance comparable to random-structure LDPC codes. In [17], an optimized belief propagation based progressive edge-growth method for constructing QC-LDPC codes is proposed. Compared to PEG, this construction method improves decoding convergence by up to 11.7 % and increases success probability by up to 10 times, according to the authors of the paper. The paper [18] introduces a permutation shift determining kind of PEG algorithm used to construct QC-LDPC protograph codes. The advantage of this method is that it can construct both binary or non-binary codes. This variety of methods makes the PEG LDPC codes worthy of further research and is also a reason why generated codes mentioned in this paper were compared to other LDPC codes from external sources of the same class, e. g. with random structure of the parity-check matrix [1].

Various methods for the removal or detection of short loops in Tanner graphs were published. For example, in [19], a parity-check matrix is used to construct a so-called adjacency matrix, the purpose of which is detection of cycles. Removed edges are placed back into different places in the Tanner graph of the parity-check matrix. The paper [20] uses a transformed parity-check matrix to create a relative

matrix of its Tanner graph, which in turn is used to detect cycles in the original parity-check matrix. The validity of the method is shown on one example LDPC code in the paper. Combinational analysis of the parity-check matrix is used as a method for loop detection in [21]. In [22–24], a modified message-passing algorithm, which is meant for decoding of LDPC codes, is used to detect the short cycles. This approach has basically the same complexity as the underlying decoding algorithm. The following method in [25] was specifically designed for QC-LDPC codes: eigenvalues of a so-called directed edge matrix based on the code's Tanner graph are used to count loops in the matrix. The next method for short cycle analysis and elimination was created with convolutional LDPC codes in mind and is described in [26]. It is based on the graphical structure of short cycles and uses a polynomial syndrome former matrix.

After studying various methods of short cycle detection and elimination, we have decided to try a more direct approach: detection of short cycles directly from the binary parity-check matrix. The second realization after studying the mentioned publications was that there are no methods (at least according to our research) of short cycle elimination which take into account how the minimum Hamming distance of the code is affected by the removal of each edge from its Tanner graph.

## 3. The aim and objectives of the study

The aim of this study was to develop a new methodology of LDPC code construction based on opportunities realized after studying other published methods of random-structure LDPC code construction, namely finding out how introducing various randomizations into the Progressive Edge-Growth process will affect its output LDPC codes in combination with detection of short cycles directly from LDPC code's binary parity-check matrix and a unique approach of short-cycle elimination based on tracking minimum Hamming distance of the code after erasure of each edge from its parity-check matrix.

The following objectives of our investigation have been set to achieve the setting goals:

– select simulation parameters and properties of LDPC codes for these simulations, which are related to the code chosen as our base for comparison;

– generate LDPC codes of various densities using implemented PEG algorithms. Examine how removing short cycles of different length from these codes' parity-check matrices affects their error-correcting performance. Determine the best codes from the spectrum of generated LDPC codes and compare their error-correcting performance to each other;

– compare the error-correcting performance of PEG codes with and without their short cycles removed. Determine the best use cases for each proposed loop-removing algorithm and pinpoint differences in their output LDPC codes;

– choose one best-performing LDPC code from previous simulations and test its performance against the best known code of the same type (chosen as a base for comparisons at the start of the study). Verify the validity of the proposed methodology and competitiveness of generated codes in comparison with the best random-structure regular LDPC codes.

## 4. Materials and methods

### 4. 1. Implementations of Progressive Edge-Growth algorithm

Four versions of the PEG algorithm were used in simulations. LDPC codes generated by these algorithms (PEG 1–4) have shown no difference in their error-correcting performance, but all four algorithms were still used to generate codes for the simulations. This is because after processing these codes with the proposed loop-removing algorithms, the error-correcting performance of the resulting codes tends to be different.

Progressive Edge-Growth algorithm needs these input data:

– number of rows in the desired parity-check matrix (=number of check nodes in its Tanner graph);

– number of columns in the desired parity-check matrix (=number of variable nodes in its Tanner graph);

– number of ones in every column of the desired parity-check matrix (degree distribution; degree of every variable node in its Tanner graph);

The output of the PEG algorithm is a parity-check matrix based on the input parameters. The algorithm focuses on one variable node at a time [27]. It chooses one node, assigns all of its connections to check nodes (based on the degree distribution on input) in a way that creates the least amount of short cycles in the final parity-check matrix and then moves on to another variable node and repeats the process until each variable node has all its connections assigned. It starts with the lowest-degree VNs and works its way to the highest-degree ones.

The basic PEG algorithm works in the following way:

1. The first connections are made to the (yet) zero-degree CNs.

2. When there are no more zero-degree CNs left, the connections are made to the CNs, which are unreachable from the current VN, favoring the lowest-degree CNs.

3. If every CN can be reached from the current VN, a connection is made to the most distant CN (the one with the most edges on the path leading to it).

Lowest-degree nodes are most susceptible to decoding errors. Therefore, the PEG algorithm tries to make these nodes part of the longest cycles in the created graph. The longer the cycle, the longer the information flowing through remains independent during decoding (not being processed by the same nodes as often), which in turn makes the nodes it connects less susceptible to decoding errors.

The proposed implementation of the PEG algorithm in the MATLAB programming environment needs four input values. Three of them were described above, the fourth required input value is the weight of CN degrees in decision making (relative to the distance between starting VN and the current CN). At the point in time when every CN is accessible from the current VN, the next unassigned edge should be connected to a CN which is the furthest away possible from this VN and which also has a low degree. Because of this, the algorithm determines the length of the shortest path to each and every CN from the currently selected VN. The fourth input parameter in this implementation of the PEG algorithm determines whether the degree of a CN has the same weight in the choice of CN as its distance from the current VN.

The proposed implementation of the PEG algorithm has four versions with various degrees of added randomization

in certain operations. During the edge-growth process, there may be situations when several lowest-degree VNs with the same priority can be chosen as the next node to start distributing edges from. Likewise, it happens during the edge-growth process that there are several CNs with the same priority to connect the next edge to. When there are several VNs or CNs with the same selection priority during the process, the selection can be done on the first-come, first-serve basis or it can be randomized. By choosing different combinations of these selection methods for CNs and VNs, four versions of proposed PEG algorithm were created:

PEG 1:
– selection of a random VN with the same degree;
– selection of a random CN with the same priority;
PEG 2:
– selection of the first VN with the same degree;
– selection of a random CN with the same priority;
PEG 3:
– selection of a random VN with the same degree;
– selection of the first CN with the same priority;
PEG 4:
– selection of the first VN with the same degree;
– selection of the first CN with the same priority.

Matrices generated by the PEG 4 algorithm possess the most visible structure. Algorithms PEG 3 and PEG 2 add more randomization to the process and binary matrices created by the PEG 1 algorithm have the most random placements of ones. Although the structure of these matrices is visibly different, their error-correcting performance proved to be identical (any differences were statistically insignificant). Even though this was the case, all four PEG algorithm versions were used in simulations because after processing their codes by loop-removal algorithms (they will be described below), the resulting codes had different error-correcting capabilities in some cases.

**4. 2. Algorithms for Removal of Short Cycles in Tanner graphs**

Degradation of LDPC decoding performance led to the creation of two proposed algorithms which remove short cycles from Tanner graphs (provided in the form of a matrix). These algorithms will be referred to as CB1 and CB2 (which stands for Cycle Break). The algorithms need two inputs: a parity-check matrix (or a binary matrix in general) and a maximum length of cycles to remove – valid value is an even integer equal or larger than 4. If the value of this parameter equals 6, the algorithm will only remove cycles of length 4 and 6. If the parameter equals 8, the algorithm will only remove cycles of length 4, 6 and 8.

Compared to other loop-removing algorithms, CB1 and CB2 search for short cycles directly in the provided binary parity-check matrix. They directly scan its ones and zeroes, search for every possible path based on their patterns and regularly check, whether the current path does not create cycles of specified lengths (the input parameter: maximum length of cycles to remove).

**4. 2. 1. Description of CB1 algorithm for removal of short cycles**

At first, the CB1 algorithm chooses a check node and progressively scans every unique path (in the Tanner graph of the input matrix) that originates in the said check node while taking note of each cycle it finds along the way. After this analysis, the program has a complete list of every cycle (of length given by the "maximum length" input parameter or shorter) which intersects the currently chosen check node. The next step is to create a list of all edges found in these listed cycles, the edges being ordered by the frequency of their occurrence in the list of cycles. The goal of the CB1 algorithm is to remove all possible cycles of specified length while not making any check node isolated in the process (degree of a node cannot get lower than 2). When an edge is removed (meaning when one is removed from the parity-check matrix), all the cycles this edge was a part of are removed from the list of found cycles. The process of edge erasure continues (by investigating the next most frequent edge) until all the listed cycles are removed or until the only edges left to erase are those which would make a check node isolated by their erasure. Afterwards, the program chooses another check node and repeats the aforementioned process (again makes a list of every cycle leading through it and tries to remove them) until every check node is analyzed and left for another one.

**4. 2. 2. Description of CB2 algorithm for removal of short cycles**

Based on shortcomings of the CB1 algorithm (in some cases, the stopping criterion was too loose; the algorithm removed too many ones from parity-check matrices and severely degraded performance of the LDPC codes), we designed and implemented the CB2 algorithm – which works in similar ways to the CB1 algorithm (it does not isolate check nodes in the process of removing cycles), but before removing a cycle, it also takes into consideration the changes to the minimum Hamming distance of the current code, which would be caused by the removal. Same as CB1, CB2 also chooses an anchor check node and locates every cycle (of specified lengths) leading through this node and sorts edges of these cycles based on the most frequent occurrence in this group of cycles. But it only removes an edge in the case when it does not result in a drop of the minimum Hamming distance of the code (while also not making nodes isolated). Calculations of minimum Hamming distance makes the CB2 algorithm much more computationally expensive compared to the CB1 algorithm because this distance needs to be frequently recalculated during the whole process. After finishing work with one CN, it chooses a different one, etc.

We chose minimum Hamming distance as a criterion in the CB2 algorithm, because it is one of the basic parameters of error-correcting codes in the coding theory and reflects their error detection and correction capabilities. The higher the minimum Hamming distance, the more errors can be corrected in a given code word. In our case, the estimation of minimum Hamming distance was done by rearranging the parity-check matrix of an LDPC code to the co-called row-echelon form, from which the generator matrix of the LDPC code was calculated. In the next step, a row with the lowest Hamming weight needs to be found in this generator matrix. The Hamming weight of this row is used as an estimation of the LDPC code's minimum Hamming distance.

Both loop-removing algorithms were successfully tested with parity-check matrices of different sizes, up to 8,000×10,000.

## 5. Results of the study of LDPC construction and performance-enhancing algorithms

### 5. 1. Simulation Parameters

Several series of computer simulations were conducted taking advantage of high-performance computing and using a network of computer workstations (computer cluster) available for research purposes in the University of Žilina to test the error-correcting performance of LDPC codes created by the mentioned PEG algorithms with and without also utilizing the proposed algorithms for the removal of short cycles (CB1, CB2) with various sets of parameters. Simulations were run and algorithms were implemented using MATLAB programming environment and C programming language.

LDPC codes for these simulations were generated by all four versions of the proposed PEG1-4 algorithms with multiple densities of parity-check matrices (Each matrix generated had a constant VN degree distribution of 2, 3, 4, 5, 6 and 9, which are equal to the number of ones in every column of the matrix, also called column weight) and dimensions of 252×504 (the matrix size was chosen so the error-correcting performance of tested codes can be compared to the best known regular PEG code with these parameters according to one of the founders of LDPC codes [28]). Every LDPC code was tested in a binary-input additive white Gaussian noise channel (BI-AWGN) with a signal-to-noise ratio ($E_b/N_0$) range from –2 dB to 10 dB with a graph data point simulated each 0.5 dB (total 25 data points for each code). In the $E_b/N_0$ range of –2÷1 dB (first 7 graph data points), $10^5$ bits were encoded, exposed to BI-AWGN and decoded for each data point. In the $E_b/N_0$ range of 1.5÷10 dB (last 18 graph data points), $10^8$ random data bits were processed in the same way for each data point (an identical binary input sequence was used in each simulation).

Each of these matrices (every PEG algorithm and all densities) had their error-correcting performance tested in simulations, but each of these matrices was also processed by both CB1 and CB2 algorithms for the removal of short cycles. Each matrix was processed by both of these algorithms several times, each time with different of the parameter "maximum length of cycles to remove" (used values for both algorithms: 4, 6 and 8). So, each of the original matrices generated by the PEG algorithms spawned several new codes, all of which had their error-correcting capabilities tested with parameters described above and below.

Input data bit sequences were encoded using generator matrices retroactively calculated from each parity-check matrix. Simulated LDPC decoder uses a custom implementation of the box-plus log-SPA, the sum-product algorithm working in logarithmic domain with the box-plus approximation used in the CN processors [1]. The number of decoding iterations was set to 20.

### 5. 2. Comparison of the best CB1-processed codes with different density

In Fig. 2, the error-correcting performance of selected best codes processed by the CB1 loop-removing algorithm is compared. The LDPC codes are grouped into these categories based on their column weight (or column weight of the PEG code they originated from). In the legend of Fig. 2, the number between two slashes (e. g. /2/) represents the column weight of the original PEG matrix the specific code is related to and the number in brackets (e.g. CB1(6) ) represents the maximum length of cycles the CB1 algorithm was set up to remove. The best codes were chosen depending on how soon they were able to reach the lowest BER in the simulated ranges (some codes had better error-correcting performance in lower $E_b/N_0$ ranges but hit an error floor too early to be considered the best code in their category).

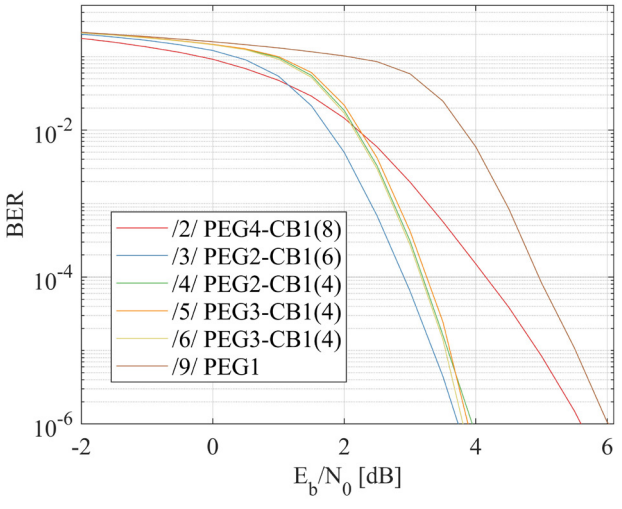



Fig. 2. Best codes processed by the CB1 algorithm per density

The best code from this graph is the PEG2-CB1 code which originated from the PEG2 code with a column weight of 3 which had its cycles of lengths 4 and 6 removed by the CB1 algorithm. In marginal cases, the PEG codes with column weight of 2 and 9 (and their related versions processed by the CB1 algorithm) have shown the worst performance in the higher $E_b/N_0$ ranges, parity-check matrices being either too sparse or too dense, respectively.

### 5. 3. Comparison of CB1 and CB2 codes to each other and to the PEG codes they originate from

The CB1 algorithm managed to improve the performance of the PEG codes which the CB2 algorithm left untouched, namely the PEG codes with sparser parity-check matrices, specifically with a constant VN degree of 2 and 3 (CB1: looser condition before edge erasure; more aggressive erasure of edges). CB2 detected the same groups of cycles as CB1 but did not erase any edges because of its stricter pre-erasure criterion. In Fig. 3, an example of error-correcting performance comparison between a code generated by the PEG 4 algorithm and its version with cycles of length 4 and 6 removed by the CB1 algorithm (marked as PEG4-CB1) can be seen. An improvement in performance is visible.

In simulations, PEG codes generated with denser parity-check matrices (constant column weight of 4, 5, 6 and 9), the CB1 algorithm removed too many edges in most cases and degraded error-correcting the performance of these codes overall. On the other hand, the CB2 algorithm managed to improve performance of codes in these cases. In Fig. 4, an example of such an improvement can be seen.

The parity-check matrix from Fig. 4 was generated by the PEG 1 algorithm with a constant VN degree of 5 (the code marked PEG). This matrix was then used as an input for both CB1 and CB2 algorithms (with a maximum length of cycles to remove set to 6, in case from Fig. 4). The CB1 algorithm significantly degraded the error-correcting ca-

pabilities of the original PEG code while CB2 managed to improve them. The denser the generated matrix (higher VN degree), the higher the improvement achieved by the removal of short cycles by the CB2 algorithm.

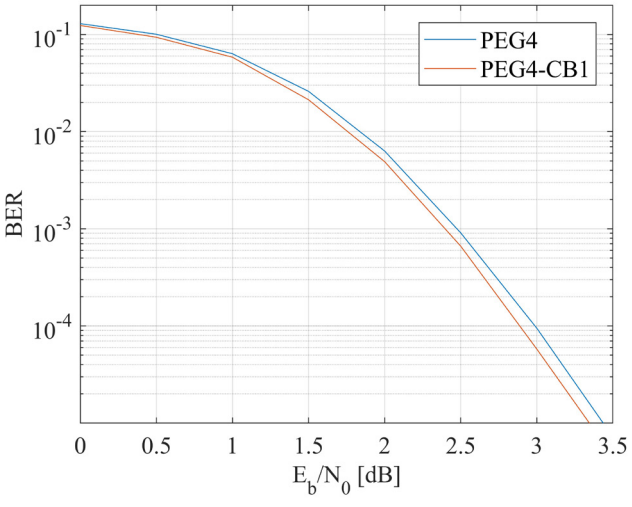



Fig. 3. Error-correcting performance of codes with column weight of 3

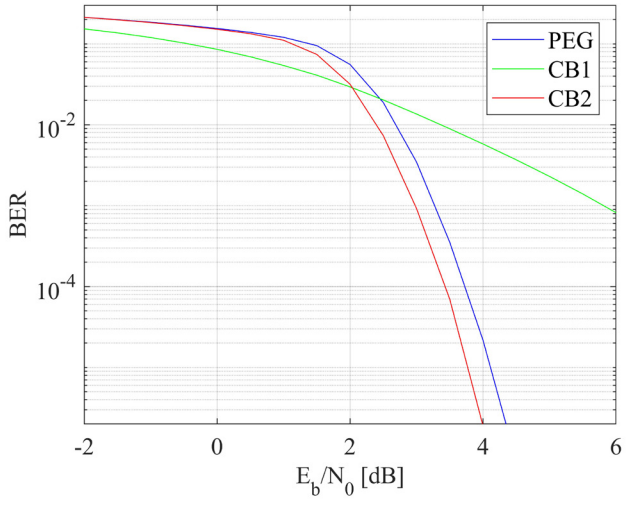



Fig. 4. Error-correcting performance comparison before and after removal of short cycles by both CB1 and CB2 algorithms

**5. 4. Comparison of the best LDPC code generated by the proposed methods to an external code with similar parameters**

The overall best code acquired from all described simulations is a 252×504 code with its parity-check matrix generated by the PEG2 algorithm with constant column weight of 3 and then processed by the CB1 loop-removing algorithm set up to remove short cycles, specifically of length 4 and 6. Length of data word (252 bits) and length of code word (504 bits) were chosen so the codes can be directly compared to codes from external sources.

A PEG-generated LDPC code declared by a founder of LDPC codes [4, 5] as being the best known regular Gallager (504, 252) code was chosen as a basis for comparison to the best code from conducted simulations (Fig. 5). The parity-check matrix was taken directly from [28] and its error-correcting performance was tested with the same software the proposed LDPC codes were tested with, under the exact same conditions.

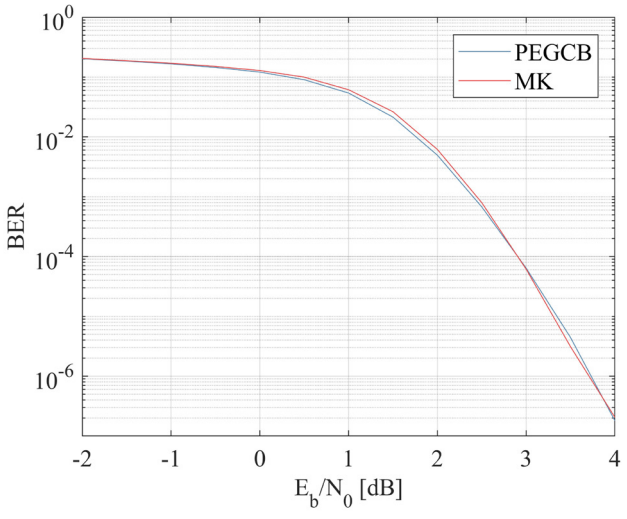



Fig. 5. Comparison to the best-known PEG-generated LDPC code with these parameters according to D. MacKay

As can be observed in Fig. 5, the performance of both compared codes is very similar: none of the codes hit any error floor within the simulated boundaries and stay very close in terms of BER, which confirms the validity of the proposed methodology and algorithms.

## 6. Discussion of the research results of LDPC construction and performance-enhancing algorithms

Two algorithms for the removal of short cycles in Tanner graphs of LDPC codes, CB1 and CB2 algorithms, were proposed in Section 4 along with four implementations of the Progressive Edge-Growth algorithm PEG1-4, which were mainly used for generating input matrices for CB1 and CB2 algorithms. The results of simulations conducted to measure the error-correcting performance of random-structure LDPC codes generated by combining the mentioned algorithms were shown and analyzed in Section 5. Each of the two loop-removing algorithms is suitable for use in specific situations, namely: the CB1 algorithm removes cycles more aggressively because of its looser pre-removal deciding criterion and managed to improve the error-correcting performance of PEG codes with sparser parity-check matrices (constant column weight: 2, 3; results can be seen in Fig. 3), but degraded the performance of PEG codes with denser parity-check matrices (constant column weight: 4÷9; an example of this can be seen in Fig. 4) in most cases. Because of its stricter pre-removal criterion, the CB2 algorithm managed to improve the performance of the codes with denser matrices (as seen in Fig. 4) in most cases but did not remove any cycles in cases of the codes with sparser matrices, making the CB1 algorithm the better choice in these particular cases. The proposed custom algorithms and described methodology were verified by creating a PEG LDPC code with similar error-correcting performance to a best-known PEG-generated LDPC code with similar parameters provided by one of the founders of LDPC codes in [28]. A comparison of these codes can be found in Fig. 5. Parameters of the code and methods for generating this code are described throughout the paper.

The mentioned simulation results and comparisons show that the proposed methodology and its unique approach to code construction can generate LDPC codes which compete with the best codes in the same category. The paper provides a set of several methods and algorithms which can be useful together as a methodology of code construction (other publications usually just focus on a single part of the whole process; related references can be found in Section 2). Or they can be used separately for partial tasks too, e.g. PEG code generation and cycle detection and/or elimination in existing LDPC codes or general Tanner graphs.

As a part of this study, a wide spectrum of codes was generated, analyzed, and compared. But parameters of each one of these codes were kept within certain constants (e.g. the dimensions of 252×504) defined by the external code chosen as a base for comparison [28]. The process of simulating operations of each of these codes took several months and a lot of processing power. Future efforts could lead to a choice of a vastly different code as a basis for comparison, e.g. an irregular LDPC code with different code rate and dimensions or to the application of methods from [13–18] to construct QC-LDPC codes using the algorithms described in this paper. These structured codes would be better suited for hardware implementation.

## 7. Conclusions

1. LDPC codes for the simulations were generated by all four versions of the PEG1-4 algorithms with multiple densities of parity-check matrices, where each matrix generated had a constant VN degree distribution of 2, 3, 4, 5, 6 and 9. The chosen matrix dimensions were 252×504 so the error-correcting performance of tested codes can be compared to the best known regular PEG code with these parameters.

2. The best code from the compared set of CB1-generated LDPC codes was the PEG2-CB1 code which originated from the PEG2 code with a column weight of 3 which had its cycles of lengths 4 and 6 removed by the CB1 algorithm.

3. The results of our simulations and comparisons have shown that each of the proposed CB1 and CB2 algorithms is suitable for different use cases. The CB1 algorithm produces better codes in cases of codes with sparser parity-check matrices and the CB2 algorithm is a better choice when improvement of a code with the denser parity-check matrix is desired.

4. The methodology of LDPC code construction using custom versions of the PEG algorithm and two cycle-removing CB algorithms described in this paper was used to create an LDPC code of specific parameters and compared to the best known code with these parameters listed in the database of codes provided in [28] by one of the founders of LDPC codes. The results of the described simulations and comparisons of these codes show that our methodology can produce error-correcting codes which rival best known codes in their class.

## Acknowledgments

This paper was written as a part of the following scientific projects: Slovak Research and Development Agency APVV-17-0631 (Co-existence of photonics sensor systems and networks in the framework of the internet of things).

References

1. Ryan, W. E., Lin, S. (2009). Channel Codes: Classical and Modern. Cambridge University Press. doi: https://doi.org/10.1017/cbo9780511803253
2. Vandendriessche, P. (2009). Some low-density parity-check codes derived from finite geometries. Designs, Codes and Cryptography, 54 (3), 287–297. doi: https://doi.org/10.1007/s10623-009-9324-9
3. Gallager, R. (1962). Low-density parity-check codes. IEEE Transactions on Information Theory, 8 (1), 21–28. doi: https://doi.org/10.1109/tit.1962.1057683
4. MacKay, D. J. C., Neal, R. M. (1996). Near Shannon limit performance of low density parity check codes. Electronics Letters, 32 (18), 1645. doi: https://doi.org/10.1049/el:19961141
5. MacKay, D. J. C. (1997). Good error-correcting codes based on very sparse matrices. Proceedings of IEEE International Symposium on Information Theory. doi: https://doi.org/10.1109/isit.1997.613028
6. Arora, K., Singh, J., Randhawa, Y. S. (2019). A survey on channel coding techniques for 5G wireless networks. Telecommunication Systems, 73 (4), 637–663. doi: https://doi.org/10.1007/s11235-019-00630-3
7. Richardson, T., Urbanke, R. (2008). Modern Coding Theory. Cambridge University Press. doi: https://doi.org/10.1017/cbo9780511791338
8. Blahut, R. E. (2003). Algebraic Codes for Data Transmission. Cambridge University Press. doi: https://doi.org/10.1017/cbo9780511800467
9. Fan, J., Xiao, Y., Kim, K. (2008). Design LDPC Codes without Cycles of Length 4 and 6. Research Letters in Communications, 2008, 1–5. doi: https://doi.org/10.1155/2008/354137
10. Liu, X., Zhang, W., Fan, Z. (2009). Construction of Quasi-Cyclic LDPC Codes and the Performance on the PR4-Equalized MRC Channel. IEEE Transactions on Magnetics, 45 (10), 3699–3702. doi: https://doi.org/10.1109/tmag.2009.2023422
11. Jiang, X.-Q., Lee, M. H., Wang, H.-M., Li, J., Wen, M. (2016). Modified PEG algorithm for large girth Quasi-cyclic protograph LDPC codes. 2016 International Conference on Computing, Networking and Communications (ICNC). doi: https://doi.org/10.1109/iccnc.2016.7440704
12. Hailes, P., Xu, L., Maunder, R. G., Al-Hashimi, B. M., Hanzo, L. (2016). A Survey of FPGA-Based LDPC Decoders. IEEE Communications Surveys & Tutorials, 18 (2), 1098–1122. doi: https://doi.org/10.1109/comst.2015.2510381
13. Prompakdee, P., Phakphisut, W., Supnithi, P. (2011). Quasi Cyclic-LDPC codes based on PEG algorithm with maximized girth property. 2011 International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS). doi: https://doi.org/10.1109/ispacs.2011.6146165
14. Huang, Y., Cheng, Y., Zhang, Y., Han, H. (2010). Construction of non-binary quasi-cyclic LDPC codes based on PEG algorithm. 2010 IEEE 12th International Conference on Communication Technology. doi: https://doi.org/10.1109/icct.2010.5689251

15. Uchoa, A. G. D., Healy, C., de Lamare, R. C., Souza, R. D. (2012). Generalised Quasi-Cyclic LDPC codes based on Progressive Edge Growth Techniques for block fading channels. 2012 International Symposium on Wireless Communication Systems (ISWCS). doi: https://doi.org/10.1109/iswcs.2012.6328513
16. Zongwang Li, Vijaya Kumar, B. V. K. (2004). A class of good quasi-cyclic low-density parity check codes based on progressive edge growth graph. Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers. doi: https://doi.org/10.1109/acssc.2004.1399513
17. Lei, Y., Dong, M. (2017). An Efficient Construction Method for Quasi-Cyclic Low Density Parity Check Codes. IEEE Access, 5, 4606–4610. doi: https://doi.org/10.1109/access.2017.2678515
18. Jiang, X.-Q., Hai, H., Wang, H.-M., Lee, M. H. (2017). Constructing Large Girth QC Protograph LDPC Codes Based on PSD-PEG Algorithm. IEEE Access, 5, 13489–13500. doi: https://doi.org/10.1109/access.2017.2688701
19. McGowan, J. A., Williamson, R. C. (2003). Loop removal from LDPC codes. Proceedings 2003 IEEE Information Theory Workshop (Cat. No.03EX674). doi: https://doi.org/10.1109/itw.2003.1216737
20. Li, B., Wang, G., Yang, H. (2009). A new method of detecting cycles in Tanner graph of LDPC codes. 2009 International Conference on Wireless Communications & Signal Processing. doi: https://doi.org/10.1109/wcsp.2009.5371660
21. Hu, P., Zhao, H. (2010). Improved method for detecting the short cycles of LDPC codes. 2010 IEEE International Conference on Information Theory and Information Security. doi: https://doi.org/10.1109/icitis.2010.5689706
22. Karimi, M., Banihashemi, A. H. (2013). Message-Passing Algorithms for Counting Short Cycles in a Graph. IEEE Transactions on Communications, 61 (2), 485–495. doi: https://doi.org/10.1109/tcomm.2012.100912.120503
23. Li, J., Lin, S., Abdel-Ghaffar, K. (2015). Improved message-passing algorithm for counting short cycles in bipartite graphs. 2015 IEEE International Symposium on Information Theory (ISIT). doi: https://doi.org/10.1109/isit.2015.7282488
24. Cho, S., Cheun, K., Yang, K. (2018). A Message-Passing Algorithm for Counting Short Cycles in Nonbinary LDPC Codes. 2018 IEEE International Symposium on Information Theory (ISIT). doi: https://doi.org/10.1109/isit.2018.8437844
25. Karimi, M., Banihashemi, A. H. (2012). Counting Short Cycles of Quasi Cyclic Protograph LDPC Codes. IEEE Communications Letters, 16 (3), 400–403. doi: https://doi.org/10.1109/lcomm.2012.020212.112311
26. Su, Z., Qiu, Q., Zhou, H. (2016). Analysis and elimination of short cycles in LDPC convolutional codes. 2016 2nd IEEE International Conference on Computer and Communications (ICCC). doi: https://doi.org/10.1109/compcomm.2016.7924880
27. Hu, X.-Y., Eleftheriou, E., Arnold, D.-M. (2001). Progressive edge-growth Tanner graphs. GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270). doi: https://doi.org/10.1109/glocom.2001.965567
28. MacKay, D. J. C. The Inference Group. Available at: http://www.inference.org.uk/is/