On Girth Conditioning for Low-Density Parity-Check Codes

Samuele Bandi, Velio Tralli, Andrea Conti, and Maddalena Nonato

***Abstract*—Low-density parity-check (LDPC) codes are gaining interest for high data rate applications in both terrestrial and spatial communications. They can be designed and studied through a bipartite graph whose characteristics affect the performance. This paper proposes a low-complexity method to improve the performance of LDPC codes by selectively removing some cycles from the associated bipartite graph. The method is based on a modified version of the breadth first search (BFS) algorithm that we call modified BFS (MBFS), which is applied to find cycles, and a greedy procedure to eliminate them. Throughout the paper we will give a detailed description of the algorithm proposed and analytically study its complexity. Simulation results show that this girth conditioning method applied to some classes of codes, whose structure allows further optimization, can lead to a significative complexity reduction and a performance improvements with respect to other methods.**

***Index Terms*—Low-density parity-check codes, girth conditioning, breadth first search algorithm, performance evaluation.**

I. Introduction

LOW-DENSITY parity-check (LDPC) codes are linear block-codes with very sparse associated parity-check matrix, that is, it contains only a low-density of non-zero elements [1]–[3]. They have been recently considered as a near Shannon limit channel coding technique for terrestrial and spatial communications with requirements in terms of high data rate and spectral efficiency [4]. A parity check matrix $\mathbf{H}$, with dimension $M \times N$ and elements in the Galois Field of order q (i.e., GF(q)) is associated to a bipartite graph $\mathcal{G}$ with N left nodes and M right nodes (an example is reported in Fig. 1a). This graph is usually called *Tanner graph* (TG), where left and right nodes are named *variable* nodes and *check* nodes, respectively. While *regular* LDPC codes are those for which all nodes of the same type have the same degree, *irregular* LDPC codes have a suitably designed random distribution of node degrees. This distribution affects both complexity and performance, thus efficient encoding and error floor reduction techniques[1] have been investigated in the literature (see, e.g., [2], [6], [7]).

The performance of an LDPC code with iterative decoding depends on some structural properties of the associated TG; of particular importance is the *girth*, that is the length of the shortest cycle in the graph (e.g., in terms of number of edges or nodes). A code is said with girth g when its TG is free of cycles of length lower than g. In the presence of cycles in the TG, the *belief-propagation algorithm* (BPA) does not converge to maximum likelihood performance [1], [8]: a message sent by a node along a cycle will propagate back to the node itself after some iterations causing loss of independence in the messages sent thereafter. However, cycle-free TGs may have small minimum distance [1], leading to poor bit error rate (BER). Several methods have been proposed to generate parity-check matrices whose associate TGs are free of short cycles (see, e.g., [9]–[14]). Progressive edge-growth (PEG) method is proposed in [14] to construct TGs, given the graph parameters, in an edge-by-edge greedy manner by optimizing the local girth at variable nodes. It can be used to generate codes of any rate and length, and for short-length codes shows significant improvement over randomly constructed codes. However, LDPC codes are often designed without explicit constraints on the girth.

In this paper we propose an efficient method to increase the girth of a given LDPC code instead of constructing a parity-check matrix of given girth. A key issue for the application of girth conditioning methods is the memory requirement. This approach has been addressed rarely in the literature; as an example, in [15] the increasing of the girth is obtained by removing edges from the TG. The potential drawback of this methodology is that edge deletion affects the degree distribution of both variable and check-nodes. We explicitely address the issue of removing as few edges as possible, to enforce a target girth. Since the problem is NP-hard [16], we propose a greedy approach based on a modified version of the well known *breadth first search* (BFS) algorithm [17] for cycle searching, called MBFS hereafter. We analyze the complexity showing that MBFS has less stringent memory requirements than the algorithm proposed in [15], and therefore can be applied to a larger family of codes.

II. The Issue of Girth Conditioning for LDPC Codes

Let us consider a bipartite graph $\mathcal{G}$, where V is the set of variable nodes and $\mathcal{T}_s$ the support tree rooted in the variable node $s \in V$. At the level i of the tree $\mathcal{T}_s$ each node j is connected to a set $\delta(j)$ of neighbors which includes one predecessor Pred(j) at level $i-1$ and one or more nodes belonging to the set $\Gamma(j) = \{k \in \delta(j), k \neq \text{Pred}(j)\}$ at level $i+1$. If $\mathcal{G}$ contains a cycle of length ℓ which includes the node s, then there will be nodes in $\mathcal{T}_s$ that occur at least twice at level $\ell/2$. As an example, in the TG of Fig. 1b two occurrences of node d at level 4 of the support tree $\mathcal{T}_a$ rooted in the variable node a indicate the existence of a cycle of

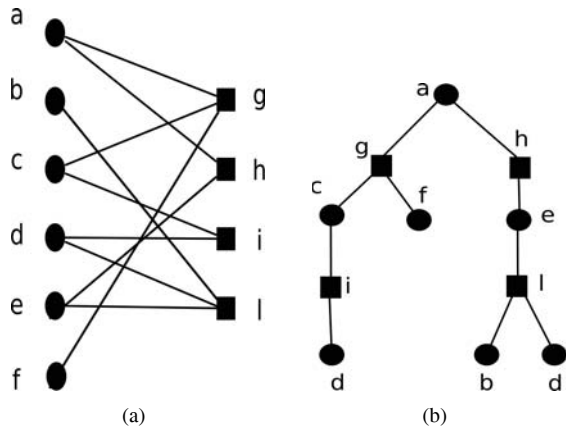
Paper approved by O. Milenkovic, the Editor for Coding Theory and Applications of the IEEE Communications Society. Manuscript received February 10, 2009; revised December 1, 2009 and June 14, 2010.

The authors are with ENDIF at University of Ferrara, via Saragat 1, 44100 Ferrara, Italy. V. Tralli and A. Conti are also with CNIT Consortium, Italy (e-mail: samuele.bandi@gmail.com, velio.tralli@unife.it, a.conti@ieee.org, nntmdl@unife.it).

This work was supported in part by the FP7 European project OPTIMIX (Grant Agreement 214625) and presented in part at Softcom 2007, Dubrovnik, Croatia.

Digital Object Identifier 10.1109/TCOMM.2011.121410.090017

[1]In the absence of floor the error probability behavior would be log-concave (see [5]).



(a) (b)

Fig. 1. (a) Example of Tanner graph containing a cycle of length 8 $\{d, l, e, h, a, g, c, i\}$; (b) A length ℓ cycle can be found exploring the support tree up to level $\ell/2$: e.g., node d and $\ell = 8$.

length 8 involving node a. To find the edges of the cycle, we follow the path from node d to a on the left branch of $\mathcal{T}_a$, and the path from a to d on the right branch.

The girth conditioning algorithm we propose consists of the following two steps, that will be further developed in Sec. II-A and II-B:

1) *find cycles* with length up to a given value. To find cycles including the node s we have to build the support tree $\mathcal{T}_s$ and explore it using a suitable search algorithm. Instead of using the the classic BFS algorithm [17], we introduce the MBFS algorithm.

2) *break cycles* by deleting as few edges as possible (since removing too many edges can destroy the structure of the graph and degrade the performance) and by avoiding the removal of edges which make the graph no longer connected.[2] After showing that this minimization problem belongs to a well-known class of NP-complete problems we will propose a greedy approach for its solution.

A. MBFS Algorithm

We propose a novel algorithm based on a FIFO queue Q which, unlike standard BFS, will not contain only nodes, but also other elements (i.e., integer numbers and asterisks, the meaning of which will be clarified later) useful to find the predecessor of each node by a simple inspection of Q. At each step, every node in the graph is labeled as either *explored*, *untouched*, or *pending*. A node is explored if it is in Q together with its neighbors, untouched if it is not in Q, and pending if it is in Q but without its neighbors. Besides the standard queue operator PUSH and POP, we introduce the function SELECT that returns the first pending element of Q from the head. Each element of Q, after being selected, either node, number, or asterisk, is marked as explored.

The construction of Q proposed here allows the exploration of the bipartite structure of the graph to find cycles without the need of storing the predecessor of each node, which impacts on the space complexity of the algorithm. A formal description of the algorithm is given in Table Algorithm 1 and illustrated hereafter. It consists of two parts:

[2]This constraint is automatically respected if only one edge is deleted in each cycle, since any node which is part of a cycle is connected to at least two edges.

Algorithm 1 MBFS algorithm

for all nodes $s \in V$ **do**
  procedure EXPLORE $\mathcal{T}_s$ UP TO LEVEL $g/2$
    $\ell = 1$; PUSH(s); PUSH(ℓ);
    while $\ell \leq g/2$ **do**
      u=SELECT (Q)
      if u is a node **then**
        PUSH($\Gamma(u)$); PUSH($*$);
      end if
      if u is a number and $u = \ell$ **then**
        ℓ++; POP; PUSH(ℓ);
      end if
      if u is an asterisk **then**
        do nothing
      end if
    end while
  end procedure

  procedure FIND NODES BELONGING TO A g-CYCLE INCLUDING NODE s
    k=0;
    for all node u appearing at least twice as pending in Q **do**
      $m = g/2$; k++; $\mathcal{C}_u^{(0)} = \{u\}$
      while $m \neq 1$ **do**
        move to the head of Q until occurence of number m
        determine the number of asterisks j met
        go to number $m-1$ in Q
        move $j+1$ positions towards the tail, skipping asterisks
        let v be the node in this position; $\mathcal{C}_u^{(k)} = \mathcal{C}_u^{(k)} \bigcup \{v\}$
        $m--$
      end while
    end for
    compose a path by joining all possible pairs of $\mathcal{C}_u^{(g/2)}$
  end procedure
end for

A) For each node $s \in V$ a queue Q_s representing the support tree $\mathcal{T}_s$ is constructed and used to find cycles of length g including the node s. To find a cycle of length $\ell \leq g$ including the variable node s we visit $\mathcal{T}_s$ up to level $\ell/2$ and assume that the TG does not contain cycles of length less than or equal to ℓ. Let us consider the example of Fig. 1 and initialize the queue Q_a with two elements: the root node a and the level number 1 (i.e., PUSH(a,1)), with both of them marked as pending. At each iteration we start from the head of Q_a and select the first pending element (say u) of Q_a (i.e., SELECT), by marking it as explored. If u is a node, we scan the set of neighbors $\delta(u)$ and append to the tail of the queue all elements in $\Gamma(u)$ and an asterisk (i.e., PUSH($\Gamma(u)$, *)) marking them as pending. If u is a number, then we append the number $u+1$ to the tail of Q_a and remove the asterisk

at end of the queue (i.e., POP). There is a subtle point in the MBFS algorithm, which makes it different from standard BFS: when u is selected, then only neighbors in $\Gamma(u)$ are added to Q_a, including those already visited. Note that the predecessor is not added to Q_a to avoid fake cycles of length 2. Note also that the exponential growth due to the insertion of replicated nodes is avoided, since we apply the search to graphs which are free of cycles up to length $\ell - 2$ and stop the search at level ℓ.

As an example, let us consider the support tree $\mathcal{T}_a$ of Fig. 1b. The queue Q_a up to the depth 4 becomes[3]: $Q_a = \{a, 1, g_a, h_a, 2, c_g, f_g, *, e_h, 3, i_c, *, *, l_e, 4, d_i, *, b_l, d_l, 5\}$. If the graph contains cycles of length g, some nodes among those marked as pending will appear more than once in Q_a. Observe that, by construction, the pending nodes are those between the last two numbers occurring in Q_a. In our example, for instance, the pending nodes are d_i, b_l, d_l and the double appearance of node d after number 4 in Q, indicates the presence of a cycle of length 8 including a.

B) To find the nodes that compose a cycle including s, we have to find the paths connecting each multiple occurrence of pending nodes in Q_s to the root node s. In the example of Fig. 1b the procedure to find the path from the second occurrence of node d at level 4 (that is d_l) to a (the root of the support tree $\mathcal{T}_a$) is as follows:

- take node d_l appearing in the list Q_a after number 4;
- move to the left and count the number j of asterisks before the occurrence of number 4;
- go to element 3 and move $j + 1$ positions (excluding asterisks) to the right, the letter in this position marks a node which is in the cycle (in our example this letter is l_e);
- iterate the procedure. The path from d_l to a is therefore $\{d, l, e, h, a\}$.

In the same way we can find the path from d_i to a: the path is $\{d, i, c, g, a\}$. By combining the two paths together we obtain the cycles of length 8 involving node a: $\{d, l, e, h, a, g, c, i, d\}$.

This algorithm applied to each variable node, that is N times, enables the construction of a simple table containing all the cycles of a given length in the TG. We will use this table to properly delete some edges from the TG in order eliminate cycles of a given length.

B. Edge deletion algorithm

The problem of selecting the minimum number of edges to be removed from an undirected graph to break all cycles of length at most ℓ is known as the *partial feedback edge set problem* (PFESP) [18]. The PFESP is also related to other classes of problems: it is a special case of the *hitting cycle problem*, where the subset of cycles to be broken can be arbitrarily selected with reference to all cycles of the graph. In a seminal work by Yannakakis [16], it was demonstrated by reduction from *vertex cover* that the PFESP is NP-hard even when restricted to bipartite graphs. However, we formulate our problem in the class of the *edge deletion problems* and the approximation result given in [19] applies.

[3]For the sake of clarity the subscript of each node indicates its predecessor.

A heuristic approach seems a viable solution method for tackling the problem of removing cycles with minimum number of deleted edges. Our proposed approach is inspired by the mixed integer linear programming formulation of the PFESP as a set covering problem (SCP): let Γ^ℓ denote the set of cycles of even length less than or equal to ℓ in $\mathcal{G}$, let E_γ be the set of edges of cycle γ, for all $\gamma \in \Gamma^\ell$, and finally let x_e be the binary variable associated to the edge e (i.e., $x_e = 1$ if the edge e is deleted). Then the problem can be formulated as:

$$\textit{find: } \min \left\{ \sum_{e \in \bigcup_{\gamma \in \Gamma^\ell} E_\gamma} x_e \right\},$$

$$\textit{subject to: } x_e \in \{0,1\} \, \forall e \in \bigcup_{\gamma \in \Gamma^\ell} E_\gamma$$

$$\text{and } \sum_{e \in E_\gamma} x_e \geq 1, \ \forall \gamma \in \Gamma^\ell.$$

Several greedy approaches have been described for SCP (see [20] for a comparison in terms of complexity). A straightforward one, algorithm **Gr** in [20], when applied to PFESP, consists of scoring edges with reference to the covered rows and ranking them in a non increasing order. Then, according to the greedy framework, the highest rank edge is iteratively selected and inserted in the partial solution, and the scores are updated, until all edges with positive score have been processed. This procedure requires the explicit enumeration of all cycles in Γ^ℓ, which is a time comsuming process (although not exponential, since ℓ is bounded). Therefore, we propose to proceed incrementally with respect to ℓ by solving one such problem for a given cycle length[4]. In this way, only a subset of cycles has to be handled at each iteration. Furthermore, cycles whose edges belong to smaller cycles might have already been broken. The heuristic procedure which we propose exploits the fact that cycles of length ℓ are searched in a graph which is cycle-free with reference to cycles of length $\ell' < \ell$.

In particular, to remove the cycles, in the proposed approach we process the table of the cycles in the set Γ_ℓ starting from the first row. For each edge $e_{i,j}$, where i and j are variable and check nodes indexes, respectively, we increase a counter $K_{i,j}$ (initialized to zero), every time we find $e_{i,j}$ in the table. As a memorization space for the different counters, the parity-check matrix $\mathbf{H}$ itself, with elements $K_{i,j}$, can be used. If $K_{i,j}$ exceeds 1, at any time, it means that the corresponding $e_{i,j}$ is involved in more than one cycle. We first eliminate the edges corresponding to the largest value of the counter. After eliminating an edge $e_{h,k}$ and the related cycles form the table, we decrease all the counters $K_{i,j}$ corresponding to each edge $e_{i,j}$ belonging to the cycles where $e_{h,k}$ is involved. The procedure can be repeated until all cycles are removed. Note that this approach does not guarantee that we remove the smallest number of edges, but it can be considered as a greedy solution to a seemingly NP-hard problem.

III. Analysis of MBFS Complexity

We now examine the complexity of the proposed algorithm. Since girth-conditioning through MBFS is performed off-chip

[4]Note that PFESP remains *NP-hard* when the cycles to brake are those with exactly ℓ edges. See Theorem 8 (ii) in [16]

TABLE I
SPATIAL COMPLEXITY OF MBFS ALGORITHM AND ALGORITHM IN [15] APPLIED TO QC CODES FROM [15] AND MNC CODES FROM [3].

QC(510,255): $d_c = 8$, $d_v = 16$, $N = 510$		
g_{max}	$\mathcal{C}_\ell$	$\mathcal{C}_\ell$ of [15]
6	1.6×10^4	8.4×10^6
8	2.1×10^6	1.1×10^9
10	2.7×10^8	1.4×10^{11}
MNC(30000,2000): $d_c = 3$, $d_v = 4$, $N = 30000$		
g_{max}	$\mathcal{C}_\ell$	$\mathcal{C}_\ell$ of [15]
8	$1.7\,10^3$	5.2×10^7
10	2.1×10^4	6.2×10^8
12	2.5×10^5	7.5×10^9

before using an LDPC code, temporal complexity is not really an issue. On the contrary, spatial complexity (i.e., memory occupation) can be the major concern. When working with long LDPC codes (with more than 10^5 nodes) a non-efficient algorithm in memory utilization can be infeasible. Hence, in this section we will treat spatial complexity. The complexity of girth conditioning mostly depends on the complexity of MBFS algorithm, that is on the number of elements in $\mathcal{T}_s$.

To analyze complexity, let us consider as a general case the irregular LDPC code. By using the same notation as in [21], the degree distribution is a polynomial $\gamma(x)$ in the form $\gamma(x) = \sum_{i\geq 2}^{d} \gamma_i x^{i-1}$ where the coefficients γ_i are non-negative, $\gamma(1) = 1$ and d is the maximum degree. In the study of LDPC codes one always refers to the performance of an *ensemble* specified by a couple of *degree distributions* $\lambda(x)$ and $\rho(x)$ [21]. More precisely, the coefficients $\lambda_i(\rho_i)$ of the polynomial $\lambda(x)$ ($\rho(x)$) represents the fraction of edges emanating from variable (check) nodes of degree i. The maximum degrees for variable and check nodes are denoted by d_v and d_c, respectively. The number of variable nodes of degree i, V_i, is then given by $V_i = N\Lambda_i$, where Λ_i is the fraction of variable nodes of degree i, that is $\Lambda_i = (\lambda_i/i)/\sum_{j\geq 2} \lambda_j/j$. The number of edges is $E = \sum_{i\geq 2} V_i$. The distributions (λ, ρ) are edge-perspective distributions. For our purpose, we are more interested in node-perspective distributions, that we will denote by (Λ, R). If we denote with A_i (B_i) the number of check (variable) nodes connected to variable (check) node i, we obtain the complexity for cycles of length ℓ as shown in the Appendix,

$$\mathcal{C}_\ell = \mathcal{O}(2\mathbb{E}\{W_\ell\} - 1) \leq \mathcal{O}(\eta^{\frac{\ell}{2}}) \qquad (1)$$

where $\eta = \sum_{i=1}^{d_v} i\Lambda_i \sum_{j=1}^{d_c} jR_j$.[5]

In case of regular (α, β)-LDPC code where every variable and check nodes have degrees α and β, respectively, (1) applies with $\eta = \alpha\beta$. Note that, since every list L used to explore the generic support tree $\mathcal{T}_s$, can be reused after finding all cycles involving s, we do not need to multiply by N the previous expression.

Note that the complexity of the method proposed in [15] is in the order of $\mathcal{O}(N\,\eta^{\frac{\ell}{2}})$. The factor N, which represents the gain in complexity between our method and the one in [15], can be exploited to remove cycles of higher orders, in particular for long codes with small d_v and d_c. In Table I we show the spatial complexity bound of the method proposed here and the one in [15] when applied to some codes appeared in literature, in particular QC codes from [15] and MNC codes from [3]. As an example, for QC codes with the same complexity required by the method in [15] to have a girth equal to 8, it is possible to obtain conditioned codes with girth 10. The benefits are even more evident with MNC codes, in fact, when the method in [15] provides girth 8, our approach gives girth at least 12.

[5] R_i is the fraction of check nodes of degree i. It is obtained, similarly to Λ_i as $R_i = \frac{\rho_i/i}{\int_0^1 \rho(x)dx}$.

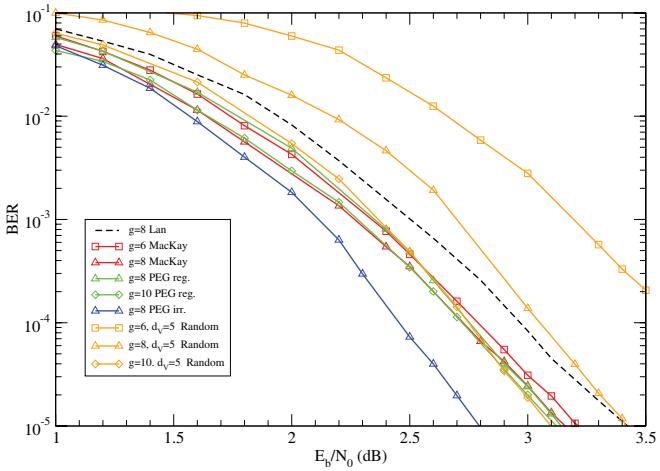


Fig. 2. BER as a function of SNR before and after girth conditioning for the MacKay's regular (504,252) code [22], the regular and irregular PEG (504,252), and the random regular ($d_v = 5$) code. For comparison, BER of QC-LDPC$(255, 510)$ after girth conditioning, as in [15], is reported.

IV. NUMERICAL RESULTS

The proposed MBFS-based girth conditioning technique can be used either to improve the performance of a given LDPC code or to construct a new LDPC code. If we start with a bipartite graph $\mathcal{G}_0$ with girth g_0 and we apply the girth conditioning technique once, we can obtain a new graph $\mathcal{G}_1$ with girth $g_1 = g_0 + 2$. Applying this technique k times we can obtain a new bipartite graph $\mathcal{G}_k$ with girth $g_k = g_0 + 2k$. One would continue to apply girth conditioning as long as the performance continues to improve. There is a limited number of girth conditioning iterations allowed due to the fact that the progressive removal of edges can eventually destroy the graph structure of the code. This behavior can be easily explained by looking at the degrees distribution of variable and check nodes. Subsequent applications of girth conditioning generate lower columns and rows degrees, moving the weight distributions towards the left. This can be interpreted as a progressive destruction of the structural properties of the code, which eventually leads to a performance degradation.

The BER as a function of the SNR is reported in Fig. 2 before and after girth conditioning for the MacKay's regular (504,252) code [22], the regular and irregular PEG (504,252), and a random regular ($d_v = 5$) code with length 504. It is shown that a marginal improvement can be obtained for MacKay's regular codes when increasing the girth from 6 to 8, and for the PEG regular codes when increasing the girth from 8 to 10. We verified that irregular PEG codes cannot be improved (not shown in the figures). All these codes have a

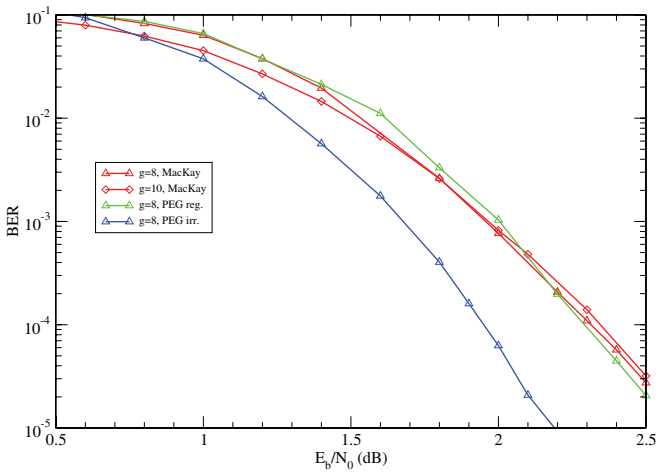


Fig. 3. BER as a function of SNR before and after girth conditioning for some codes with length 1008: MacKay's regular (1008,504) rate-1/2 code [22], regular PEG (1008,504) code, and irregular PEG (1008,504) code.

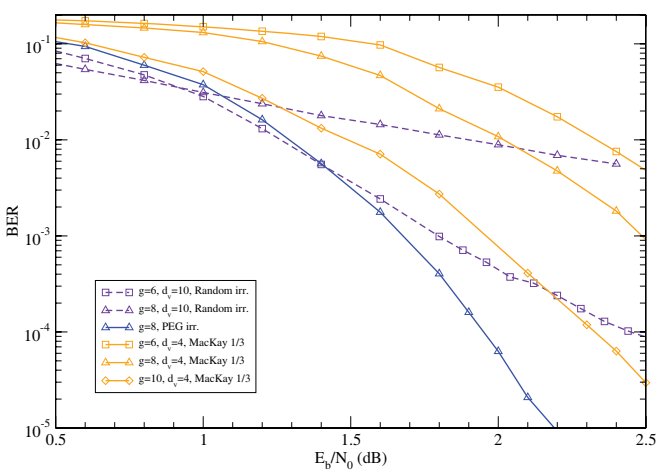


Fig. 4. BER as a function of SNR for some codes before and after girth conditioning: MacKay's regular (816,272) rate-1/3 code ($d_v = 4$) [22] and irregular ($d_v = 10$) random code. For comparison the performance of irregular PEG (1008,504) code is reported.

maximum degree of variable nodes $d_v = 3$. It is interesting to note, on the other hand, that significant performance improvement is obtained through girth conditioning (from 6 to 10) when random regular codes with $d_v = 5$ are considered. With girth equal to 10 random codes obtain performance close to that of MacKay's and PEG codes. This means that when the degree of variable nodes in regular codes is not small there is space to increase the girth through edge removal with performance improvement.

A girth conditioning technique that removes edges following a completely different Trellis-based algorithm was proposed in [15]; to compare the two strategies in terms of BER we condition the same QC-LDPC code proposed in *Example 1* of [15] which has a 255×510 parity check matrix $\mathbf{H_0}$ consisting of two 255×255 circulants M_1 and M_2.[6] By comparing our results of Fig. 2 with the ones obtained in [15] we can observe that our algorithm gives the same performance despite the lower complexity. This means that performance degradation introduced by girth conditioning (beyond a given value of g_k) does not depend on the efficiency of the algorithms. The lower complexity of our algorithm, however, makes it feasible for larger classes of codes, for which algorithm proposed in [15] may not be applicable.

This allows us to investigate the effects of girth conditioning on other classes of LDPC codes. As an example, we consider here the attempt of further optimizing other codes of length around 1000, that is the MacKay's regular (1008,504) rate-1/2 code ($d_v = 3$) and (816,272) rate-1/3 code ($d_v = 4$) [22], the regular PEG (1008,504) code [14] and irregular ($d_v = 10$) random codes (the latter constructed according to some degree distributions optimized with the well known density-evolution technique [21]). Random codes are constructed without cycles of length 4. Looking at BER as a function of SNR reported in Figs. 3 and 4, we note that we cannot further improve the performance of these LDPC codes, with the exception of rate-1/3 code (girth 6 and $d_v = 4$) which allows significant improvements, up to 0.8 dB. This means that the tool of girth conditioning, for the reasons outlined in the paper, is useful for codes constructed or designed with a small girth and weight distribution with not small minimum degree (i.e., a structure that allows further optimization). This is the case of the regular rate-1/3 code in the figure, whereas this is not the case of random irregular code with optimized degree distribution. Well designed codes do not need girth conditioning. Since codes with large block length are usually obtained with random construction and optimized irregular degree distribution [21], girth conditioning looses the relevance for this application.

[6]A $n \times n$ circulant is characterized by a polynomial $\mathbf{g}(X)$ over $GF(2)$ whose coefficients, with their n cyclic shifts, determine the matrices. In particular, the comparison is made with codes having $\mathbf{g}_1(X) = 1 + X^{47} + x^{72} + X^{104} + X^{106} + X^{191} + X^{212} + X^{225}$ and $\mathbf{g}_2(X) = 1 + X^8 + x^{37} + X^{87} + X^{125} + X^{137} + X^{149} + X^{161}$.

V. Conclusions

In this paper we proposed an algorithm based on MBFS for removing cycles of TG and we investigated its performance and complexity. The MBFS-based girth conditioning method can be used either to improve the performance of and existing LDPC code or to construct a new code. The reduced complexity of MBFS algorithm makes this method feasible even for long block length codes, without the need to memorize big trellis structures as proposed by other methods in literature. The results show significant performance improvement for some classes of codes, whereas in some others cases is not effective. This motivates further analysis on efficiency of the girth conditioning techniques in relation with the construction method of the LDPC code.

Appendix: Complexity of MBFS Algorthm Applied to Irregular LDPC Codes

Let us denote with A_i (B_i) the number of check (variable) nodes connected to variable (check) node i, and with X_m (Y_m) the variable (check) nodes at level m of a support tree $\mathcal{T}_s$. The following inequality holds

$$X_{m+2} \leq \sum_{i=1}^{\sum_{j=1}^{X_m} A_j} B_i \quad \text{with } m \text{ even}. \tag{2}$$

This is quite evident since the number of check nodes at level $m+1$ (odd) is $\sum_{i=1}^{X_m} A_i$ and each check node has degree B_i. In a perfectly symmetric fashion, the following inequality holds for check nodes at level $m+2$ (odd)

$$Y_{m+2} \leq \sum_{i=1}^{\sum_{j=1}^{Y_m} B_j} A_i \quad \text{with } m \text{ odd}. \tag{3}$$

If X_i and S are independent, non-negative, and integer-valued random variables, then $\mathbb{E}\{\sum_{i=1}^{S} X_i\} = \mathbb{E}\{X\}\mathbb{E}\{S\}$ (see [23]). By considering the expected values we obtain therefore

$$\mathbb{E}\{X_{m+2}\} \leq \mathbb{E}\left\{\sum_{i=1}^{X_m} A_i\right\}\mathbb{E}\{B\} = \mathbb{E}\{X_m\}\eta \tag{4}$$

where $\eta = \mathbb{E}\{A\}\mathbb{E}\{B\}$. Since $X_0 = 1$ (at level zero we have only the root node), the average number of variable nodes at level m of the support tree is given by

$$\mathbb{E}\{X_m\} \leq \eta^{m/2} \quad \text{with } m \text{ even}. \tag{5}$$

Note that $\mathbb{E}\{A\} = \sum_{i=1}^{d_v} i\Lambda_i$ and $\mathbb{E}\{B\} = \sum_{i=1}^{d_c} iR_i$. By using a similar procedure we obtain the following result for Y_m

$$\mathbb{E}\{Y_m\} \leq \mathbb{E}\{A\}\eta^{\frac{m-1}{2}} \quad \text{with } m \text{ odd}. \tag{6}$$

Let us now define $Z_m = X_m + Y_{m+1}$ with m even; we obtain $\mathbb{E}\{Z_m\} \leq (\mathbb{E}\{A\}+1)\eta^{m/2}$. Hence, the average number of nodes of a support tree rooted in s, $\mathcal{T}_s$ of ℓ levels, with ℓ even, can be obtained by summing $\mathbb{E}\{Z_m\}$ over the even values of m, as

$$\begin{aligned}\mathbb{E}\{W_\ell\} &\leq \mathbb{E}\left\{\sum_{i=0}^{\ell/2} Z_{2i}\right\} = (\mathbb{E}\{A\}+1)\sum_{i=0}^{\ell/2}\eta \\ &= (\mathbb{E}\{A\}+1)\frac{\eta^{\frac{\ell}{2}+1}-1}{\eta-1}. \end{aligned} \tag{7}$$

Since we visit $\mathcal{T}_s$ with BFS, the complexity of the search operation on the single support tree results in (1).

VI. Acknowledgments

Authors wish to thank the associate editor and anonymous reviewers for helpful comments and suggestions. Authors would like to thank Prof. Chiani and Dr. Paolini for helpful discussions.

References

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. MIT Press, 1963.
[2] T. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 638-656, Feb. 2001.
[3] M. Kay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399-431, Feb. 1999.
[4] G. P. Calzolari, M. Chiani, F. Chiaraluce, R. Garello, and E. Paolini, "Channel coding for future space missions: new requirements and trends," *Proc. IEEE*, vol. 95, no. 11, pp. 2157-2170, Nov. 2007.
[5] A. Conti, D. Panchenko, S. Sidenko, and V. Tralli, "Local bounds based on log-concavity property of the error probability in wireless communication systems," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2766-2775, June 2009.
[6] G. Liva, E. Paolini, and M. Chiani, "Quasi-cyclic generalized LDPC codes with low error floor," *IEEE Trans. Commun.*, vol. 56, no. 1, pp. 49-57, Jan. 2008.
[7] M. Yang, Y. Li, and W. E. Ryan, "Design of efficiently encodable moderate-length high-rate irregular LDPC codes," *IEEE Trans. Commun.*, vol. 52, no. 4, pp. 564-571, Apr. 2004.
[8] T. Richardson and R. Urbanke, "The capacity of low-density parity-check codes under message passing decodiong," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599-619, Feb. 2001.
[9] Y. Mao and A. H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," in *Proc. IEEE ICC 2001*, June 2001, Helsinki, Finland, pp. 41-44.
[10] J. Rosenthal and P. O. Vontobel, "Constructions of regular and irregular LDPC codes using Ramanujan graphs and ideas from Margulis," in *Proc. IEEE ISIT 2001*, June 2001, Washington DC, p. 4.
[11] D. Leyba, O. Milenkovic, Bennet and N. Kashyap, "New partition-regular sequences and array codes of large girth," in *Proc 42nd Annual Allerton Conf. Commun., Control Comput.*, Monticello, IL, 2004, pp. 240-249.
[12] O. Sullivan, "Algebraic construction of sparse matrices with large girth," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 718-727, Feb. 2006.
[13] M. P. C. Fossorier, "Quasicyclic, low-density parity-check codes from circulant permutation matrices," *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788-1793, Aug. 2004.
[14] X.-Y. Hu, E. Eleftheriou, and D. M. Arnold, "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386-398, Jan. 2005.
[15] L. Lan, Y. Y. Tai, L. Chen, S. Lin, and K. Abdel-Ghaffar, "A trellis-based method for removing cycles from bipartite graphs and construction of low density parity check codes," *IEEE Commun. Lett.*, vol 8, no. 7, July 2004.
[16] M. Yannakakis, "Edge-deletion problems," *SIAM J. Comput.*, vol. 10 no. 2, pp. 297-309, 1981.
[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd edition. MIT Press and McGraw-Hill, 2001.
[18] M. R. Garey and D. S. Johnson, *Computers and Intractability.* W. H. Freeman and company, 1979.
[19] A. Natanzon, R. Shamir, and R. Sharan, "Complexity classification of some edge modification problems," *Discrete Applied Mathematics*, vol. 113, pp. 109-128, 2001.
[20] T. Grossmana and A. Wool, "Computational experience with approximation algorithms for the set covering problem," *European J. Operational Research*, vol. 101, no. 1, pp. 81-92, 1997.
[21] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619-637, Feb. 2001.
[22] D. J. C. MacKay, *Encyclopedia of Sparse Graph Codes*. [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.
[23] S. M. Ross, *Applied Probability Models with Optimization Application*. Dover Publications, 1970.