Improved Trellis-Based Algorithm for Locating and Breaking Cycles in Bipartite Graphs with Applications to LDPC Codes

Juane Li
AppoTech USA
Sunnyvale, CA 94085, USA
E-mail: jueli@ucdavis.edu

Shu Lin and Khaled Abdel-Ghaffar
University of California, Davis
Davis, CA 95616, USA
E-mail: {shulin, ghaffar}@ucdavis.edu

Abstract—LDPC codes are typically decoded using belief propagation decoding algorithms applied to their Tanner graphs. Cycles of short lengths in these graphs, which are bipartite graphs representing parity-check equations of the codes, can negatively impact the performance of the decoder. In this paper, a trellis-based algorithm is presented to locate cycles in bipartite graphs. This algorithm is an improvement in complexity and run-time over the one developed by Lan et al. in 2004. When applied to Tanner graphs of quasi-cyclic LDPC codes, the newly proposed algorithm can exploit the structure of their parity-check matrices over the one developed by Lan et al. in 2004. When applied to the Tanner graphs of LDPC codes and broken them in order to improve performance. The paper is concluded in Section IV.

I. INTRODUCTION

Iterative belief propagation techniques, such as the sum-product algorithm (SPA) and the min-sum algorithm (MSA), are effective in decoding LDPC codes. These techniques are applied to the Tanner graphs representing the parity-check matrices of the codes. Studies have shown that the performance of these decoders depends on the girth and the number of short cycles in the Tanner graphs used for decoding. Short cycles cause messages exchanged in iterative decoding to be correlated after a few iterations, which prohibits the decoding from converging or makes it converge slowly, see e.g., [1],[2] and references therein. We propose an algorithm to find cycles in the Tanner graphs of LDPC codes and break them in order to improve performance.

Throughout this paper, $\mathcal{G} = (\mathcal{U} \cup \mathcal{W}, \mathcal{E})$ denotes a bipartite graph which is the Tanner graph of an LDPC code where $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ is the set of $n$ variable nodes (VNs), $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ is the set of $m$ check nodes (CNs), and $\mathcal{E}$ is the set of $E$ edges, each connecting a VN to a CN. A walk of length $l$ in $\mathcal{G}$ is a sequence of nodes $(v_1, v_2, \ldots, v_{l+1})$ in $\mathcal{U} \cup \mathcal{W}$ such that there is an edge in $\mathcal{E}$ connecting $v_i$ and $v_{i+1}$ for $1 \leq i \leq l$. The walk is said to be closed if $v_1 = v_{l+1}$. The walk is called a path if $v_1, v_2, \ldots, v_l$ are distinct and a cycle if $v_1, v_2, \ldots, v_l$ are distinct and $v_1 = v_{l+1}$. The girth, $g$, of $\mathcal{G}$ is the length of a shortest cycle, which is even for a bipartite graph. The bipartite graph is said to be $(d_u, d_w)$-regular if each VN has degree $d_u$ and each CN has degree $d_w$.

Numerous researchers have proposed techniques to count cycles, see e.g. [3]–[7]. However, these techniques can count cycles of lengths up to $2g - 2$, where $g$ is the girth of the graph since they fail to distinguish between cycles and closed walks consisting of more than one cycle. The lengths of these cycles are at least twice the girth. By restricting the algorithms to count cycles of length less than $2g$, all these closed walks which are not cycles are eliminated. The most efficient one of these algorithms is the technique presented in [7] which has computational complexity $O(gE^2)$.

By breaking cycles in the Tanner graph of an LDPC code, we may be able to improve its performance. To be able to accomplish this, cycles need to be found and not only counted. In [8], Lan et al. presented a trellis-based algorithm (TBA) to find cycles of length equal to the girth $g$ based on a trellis expansion of the graph. Their algorithm allows them to break cycles to obtain a new graph with larger girth. To find cycles of length $g$, the complexity is $O(n^d u_{g/2}^2 d_w^2)$ for a $(d_u, d_w)$-regular bipartite graph. More recently, Bandi et al. [9] developed a method based on the Breadth First Search algorithm which reduces the memory requirement by a factor of $n$. They also proposed a greedy algorithm to eliminate cycles by removing as few edges as possible.

In this paper, we propose an improved trellis-based algorithm (ITBA) which is more efficient than the TBA in [8]. Unlike the methods in [8] and [9] which are devised to only find cycles of length $g$, the ITBA can be used to find cycles of any length. In particular, it can be used to count cycles of lengths equal to or greater than $2g$ which cannot be counted by other methods. Further, starting with a quasi-cyclic (QC) LDPC code, then unlike [9], we develop a technique to break cycles that maintains the QC structure which simplifies both the encoding and decoding.

This paper is organized as follows. Section II starts with a brief review of the TBA presented in [8] before developing the ITBA and determining its complexity. Section III applies the ITBA to QC-LDPC codes and describes how to remove cycles in order to improve performance. The paper is concluded in Section IV.
II. TRELLIS-BASED ALGORITHMS

A. The TBA

Consider a bipartite graph \( G = (U \cup W, E) \) with \( n = |U| \) VN nodes and \( m = |W| \) CN nodes that has girth \( g \). The TBA in [8] forms a trellis composed of a number, \( t \), of sections each representing the graph \( G \). Such a trellis has \( t+1 \) levels of nodes numbered from 0 to \( t \) [10]. Nodes in even levels are labeled with the VN nodes in \( U \) and nodes in odd levels are labeled with the CN nodes in \( W \). Let \( V_l = \{v_{1,l}, v_{2,l}, \ldots, v_{n,l}\} \) be the set of nodes at the \( l \)-th level. If \( l \) is even, \( q_l = n \) and \( v_{i,l} \) is labeled with \( u_i \) for \( 1 \leq i \leq n \). If \( l \) is odd, \( q_l = m \) and \( v_{i,l} \) is labeled with \( w_i \) for \( 1 \leq i \leq m \). Two nodes \( v_{i,l} \) and \( v_{j,l+1} \), \( 0 \leq l \leq t \), in two consecutive levels are connected by a branch if and only if their labels in \( G \) are connected by an edge. Therefore, every section of the trellis is a representation of the graph \( G \) and every section is the mirror image of its preceding one. As an example, consider the bipartite graph \( G \) in Fig. 1(a), where the nodes in \( U \) and \( W \) are represented by hollow and full circles, respectively. Fig. 1(b) shows the trellis composed of six sections of the graph \( G \).

The TBA locates cycles of length \( 2k \leq 2g - 2 \) in \( G \) using a trellis composed of \( t \geq 2k \) sections. This is done by tracing paths of length \( 2k \) in a trellis composed of \( t \geq 2k \) sections starting at a node \( v_{i,0} \), for some \( 1 \leq i \leq n \), at the 0-th level and ending at a node \( v_{i,2k} \) at the \((2k)\)-th level such that no two branches in a path correspond to the same edge in \( G \). Notice that both the starting node \( v_{i,0} \) and the ending node \( v_{i,2k} \) of the path are labeled with the same VN \( u_i \). Each such path identifies a cycle of length \( 2k \) in the graph and, conversely, each cycle of length \( 2k \) corresponds to such a path. For example, in Fig. 1(b), \((u_1, u_{1}, u_2, w_4, u_3, w_2, u_1)\) is a path that corresponds to a cycle of length six in the bipartite graph \( G \).

The power of the TBA comes from having an efficient way to find paths that correspond to cycles by working on the trellis recursively section by section. As explained in [8], the algorithm can be used to eliminate cycles of short lengths in order to increase the girth of the graph.

B. The ITBA

The ITBA is based on the fact that a path in a trellis corresponding to a cycle in \( G \) of length \( 2k \) is a concatenation of two paths, each of length \( k \): one starting at node \( v_{i,0} \) labeled with \( u_i \) at the 0-th level and ending at some node \( v_{j,k} \) at the \( k \)-th level and the other starting at node \( v_{j,k} \) at the \( k \)-th level and ending at node \( v_{i,2k} \) labeled with \( u_i \) at the \((2k)\)-th level. This second path, if traced backward, is the mirror image of a path of length \( k \) starting at node \( v_{i,0} \) labeled by \( u_i \) at the 0-th level and ending at node \( v_{j,k} \) labeled by \( u_j \) if \( k \) is even and \( w_j \) if \( k \) is odd at the \( k \)-th level. Hence, it suffices to determine cycles of length \( 2k \) based on certain pairs of paths of length \( k \), rather than \( 2k \) as in the TBA. For example, in Fig. 1(b), the paths labeled by \((u_1, u_1, u_2, w_4)\) and \((u_1, u_2, u_3, w_4)\) of length three, define the same cycle \((u_1, u_1, u_2, u_3, w_2, u_1)\) in \( G \) of length six as the one obtained using the TBA.

To characterize the cycles of length \( 2k \) passing through node \( u_i \), we define a partial elementary path of length \( l \leq k \), starting at node \( v_{i,0} = v_{e_0,0} \) at the 0-th level labeled with \( u_i \) and ending at node \( v_{j,l} = v_{e_l,t} \), \( 1 \leq j \leq q_l \), at the \( l \)-th level, to be the labels of a sequence of \( l+1 \) nodes \((v_{e_0,0}, v_{e_1,t}, v_{e_2,t}, \ldots, v_{e_l,t})\) with distinct labels such that \( v_{e_l,t} \) and \( v_{e_{l+1},t+1} \), \( 0 \leq t \leq l-1 \), are connected by a branch in the trellis. Such a partial elementary path corresponds to a path of length \( l \) in the graph \( G \) starting at node \( u_i \), which is the label of \( v_{e_0,0} \), and ending at node \( v_j \), which is the label of \( v_{e_l,t} \) if \( l \) is even, or \( w_j \), which is the label of \( v_{e_l,t} \) if \( l \) is odd. The set of such partial elementary paths is denoted by \( \text{PEP}(i,j,l) \).

Notice that all paths in \( \text{PEP}(i,j,l) \) have the same starting node and the same ending node. Two paths in \( \text{PEP}(i,j,k) \) that do not have other nodes in common correspond to a cycle of length \( 2k \) in \( G \). Conversely, any cycle of length \( 2k \) passing through node \( u_i \) correspond to a path of pairs in \( \text{PEP}(i,j,k) \) for some \( j \).

To identify the set of partial elementary paths passing through node \( u_i \), the trellis is processed level by level starting with \( v_{i,0} \) at the 0-th level labeled with \( u_i \). A list of partial elementary paths is compiled at each level and updated to include all partial elementary paths at the next level. At the 0-th level, there is only one partial elementary path in \( \text{PEP}(i,0) \), namely \((u_i)\) while \( \text{PEP}(i,j,0) \) is empty for \( j \neq i \). Now, suppose we have processed the trellis up to the \( l \)-th level, \( l \geq 0 \), and have the set of partial elementary paths \( \text{PEP}(i,j,l) \) for all \( j \) where \( 1 \leq j \leq q_l \), i.e., \( 1 \leq j \leq n \) if \( l \) is even and \( 1 \leq j \leq m \) if \( l \) is odd. Each path in \( \text{PEP}(i,j,l) \) is a sequence of distinct labels \((v_{e_0,0} = v_{i,0}, v_{e_1,t}, v_{e_2,t}, \ldots, v_{e_l,t} = v_{j,l})\). The paths in \( \text{PEP}(i,j,l+1) \), for \( 1 \leq j \leq q_{l+1} \), are obtained by appending the label of \( v_{j,l+1} \) to every path in \( \text{PEP}(i,j,l) \) to obtain the sequence of labels \((v_{e_0,0} = v_{i,0}, v_{e_1,t}, v_{e_2,t}, \ldots, v_{e_l,t} = v_{j,l}, v_{e_{l+1},t+1} = v_{j,l+1})\) if and only if (1) the label of \( v_{j,l+1} \) is distinct from the labels of \((v_{e_0,0}, v_{e_1,t}, v_{e_2,t}, \ldots, v_{e_l,t})\), and (2) there is a branch in the \( l \)-th section of the trellis connecting \( v_{e_{l+1},t+1} = v_{j,l+1} \) to \( v_{j,l+1} \). Once the set of elementary partial paths \( \text{PEP}(i,j,k) \) is constructed for all \( j, 1 \leq j \leq n \) if \( k \) is even and \( 1 \leq j \leq m \) if \( k \) is odd, we determine all unordered pairs of distinct paths in \( \text{PEP}(i,j,k) \) that do not have any common nodes except for the starting and ending nodes. Then, by reading the nodes of a path in such a pair followed by reversely reading the nodes in the other path, we identify a cycle in \( G \) of length \( 2k \) starting and ending at \( u_i \). Conversely, any such cycle in \( G \) corresponds to a pair of such partial elementary paths.

By summing the numbers of such pairs in \( \text{PEP}(i,j,k) \) for all \( j \), we obtain the number of cycles in \( G \) of length \( 2k \) passing through \( u_i \). If we sum this number over all \( i \) and divide the result by \( k \), we obtain the number of cycles in \( G \) of length \( 2k \) as such each cycle passing through \( u_i \) contributes to \( k \) pairs of paths, and each pair is in \( \text{PEP}(i',j',k) \) for some \( i' \) and \( j' \) for which the cycle passes by \( u_{i'} \). To avoid this \( k \)-fold counting
repetitions, a node can be deactivated as soon as its cycles are identified. Deactivating a node amounts to removing the node and its adjacent edges from the graph $G$. This results in a trellis with a smaller number of nodes in each level with even label and a smaller number of branches in each subsection. As the algorithm proceeds, the number of nodes and branches in the trellis becomes smaller and smaller and the algorithm gets faster and faster. Notice that the ITBA is implemented sequentially by identifying cycles passing through the nodes one by one. We can apply the algorithm in a parallel way by searching the partial paths of all the nodes in $U$ simultaneously. Parallel implementation can speed up the finding process, but it does not allow for node deactivation to reduce computational complexity.

**Example 1.** Consider the bipartite graph $G$ in Fig. 1(a). Suppose we are interested in finding the cycles which pass by node $u_1$. First, we form a trellis as shown in Fig. 1(b). We start with $PEP(1, 1, 0) = \{u_1\}$ and $PEP(1, j, 0)$ is the empty set for all $j \neq 1$. We extend the paths in $PEP(1, 1, l)$, level by level, i.e., by incrementing the value of $l$, in order to identify cycles of arbitrary lengths as follows:

i) At the 1-st level, $PEP(1, 1, 1) = \{(u_1, w_1)\}$, $PEP(1, 2, 1) = \{(u_1, w_2)\}$, $PEP(1, 3, 1) = \{(u_1, w_3)\}$, and $PEP(1, 4, 1)$ is an empty set. As there are no pairs of paths in $PEP(1, 1, j)$, for $j, 1 \leq j \leq 4$, starting and ending at the same node, there is no cycle of length two in $G$ passing through $u_1$.

ii) At the 2-nd level, $PEP(1, 1, 2)$ is an empty set, $PEP(1, 2, 2) = \{(u_1, w_1, u_2)\}$, $PEP(1, 3, 2) = \{(u_1, w_1, u_2, u_3)\}$, and $PEP(1, 4, 2) = \{(u_1, w_1, u_2, u_4)\}$. There are two pairs of paths starting at $u_1$ and ending at a common node and not sharing any other node, namely, the pair $(u_1, u_1, u_3)$, $(u_1, u_2, u_3)\}$ in $PEP(1, 3, 2)$ and the pair $(u_1, w_2, u_4), (u_1, w_3, u_4)\}$ in $PEP(1, 4, 2)$. The first pair gives the cycle $(u_1, u_1, u_3, w_1, u_1)$ and the second pair gives the cycle $(u_1, w_2, u_4, w_1, u_1)$. These are the only cycles of length four in $G$ passing through $u_1$.

iii) At the 3-rd level, $PEP(1, 1, 3) = \{(u_1, w_1, u_3, w_3)\}$, $PEP(1, 2, 3) = \{(u_1, w_1, u_3, u_2)\}$, and $PEP(1, 3, 3) = \{(u_1, w_2, u_4, u_3)\}$, and $PEP(1, 4, 3) = \{(u_1, w_1, u_2, w_1, u_3, w_4)\}$. There are two pairs of paths starting at $u_1$ and ending at a common node and not sharing any other node, namely, $(u_1, u_1, u_3, w_2), (u_1, w_2, u_4, w_2)\}$ in $PEP(1, 2, 3)$ and $(u_1, u_1, w_2, u_4), (u_1, w_2, u_3, w_4)\}$ in $PEP(1, 4, 3)$, which define the two cycles $(u_1, u_1, u_3, w_2, u_4, w_2, u_1)$ and $(u_1, w_2, u_3, w_4, u_2, w_2, u_1)$, respectively. These are the only cycles of length six in $G$ passing through $u_1$.

iv) At the 4-th level, $PEP(1, 1, 4)$ is an empty set, $PEP(1, 2, 4) = \{(u_1, u_1, u_3, u_2, w_2), (u_1, w_2, u_4, u_2, w_2)\}$, $PEP(1, 3, 4) = \{(u_1, w_1, u_3, u_2, w_4, u_3)\}$, and $PEP(1, 4, 4) = \{(u_1, w_1, u_3, u_2, w_4, u_4)\}$. There is only one pair of paths starting at $u_1$ and ending at a common node and not sharing any other node, namely, $(u_1, u_1, u_2, w_4, u_3), (u_1, w_3, u_4, w_2, u_3)\}$ in $PEP(1, 3, 4)$. This pair gives the cycle $(u_1, w_1, u_3, u_2, w_4, u_3, w_2, u_4, u_3, u_2)\}$, which is the unique cycle of length eight in $G$ passing through $u_1$.

We can continue the algorithm to identify longer cycles.

**C. Complexity of the ITBA**

The computational complexity of the ITBA mainly depends on the number of branches in the trellis that we need to search. Suppose we want to find all cycles of lengths up to $2k$. Let $N^{u, \text{branches}}_{(2k)}$ denote the total number of branches to search for the cycles of length $2k$ which pass through node $u$. A trellis with $k$ sections is needed. For a $(d_u, d_w)$-regular bipartite graph, the number of branches to search from the $(l-1)$-th level to the $l$-th level, where $1 \leq l \leq k$, is $d_u(d_u-1)^{l-1}(d_u-1)^{-1}$ if $t$ is odd and $d_u(d_u-1)^{l-1}(d_u-1)^{l-1}$ if $t$ is even. Thus, to find all the cycles containing $u$, the total number of edges to search is

$$N^{u, \text{branches}}_{(2k)} = \sum_{i=1, \text{even}}^{k} d_u(d_u-1)^{i-1}(d_u-1)^{\frac{i}{2}} + \sum_{i=1, \text{odd}}^{k} d_u(d_u-1)^{\frac{i-1}{2}}(d_u-1)^{\frac{i-1}{2}} + d_u(d_u-1)^{\frac{k-1}{2}}(d_u-1)^{\frac{k-1}{2}}$$

In particular, the complexity of the proposed algorithm to find all cycles of length $2k$ is $O(nd_u^{\frac{k}{2}}d_w^{\frac{k}{2}})$, where $n$ is the number of variable nodes in $U$. One point worth mentioning here is that this complexity $O(nd_u^{\frac{k}{2}}d_w^{\frac{k}{2}})$ is an upper bound since by deactivating nodes once the cycles passing through them are found, the graph becomes smaller and smaller as the ITBA proceeds. Thus, the complexity of finding cycles of length $2k$ containing $u_{i+1}$ is smaller than that of $u_i$.

Regarding memory usage, for each $u \in U$, the total number of partial paths at the $k$-th level is $d_u(d_u-1)^{\frac{k-1}{2}}(d_u-1)^{\frac{k-1}{2}}$ when $k$ is odd, and $d_u(d_u-1)^{\frac{k-1}{2}}(d_u-1)^{\frac{k}{2}}$ when $k$ is even. For each partial path, we need to store the indices of $k$ nodes. The proposed algorithm requires $O(k nd_u^{\frac{1}{2}}d_w^{\frac{1}{2}})$ memory locations, each storing an integer number.

The proposed ITBA searches a trellis with half the number of sections as the trellis searched in the algorithm given in [8]. Actually, the algorithm in [8] identifies partial paths during the second half of the search process which have been already identified during the first half of the process. To find the cycles of length $2k$ in a $(d_u, d_w)$-regular graph $G$ with $n$ VN, the computational complexity of the algorithm in [8] is $O(nd_u^{\frac{k}{2}}d_w^{\frac{k}{2}})$, while the complexity of the proposed algorithm is $O(nd_u^{\frac{k}{2}}d_w^{\frac{k}{2}})$. We can see that the proposed algorithm has a large reduction in computational complexity. It also results in
in a large reduction in memory requirement compared with the TBA which requires \( O(2knd_u^2d_w^2) \) memory locations.

III. THE ITBA FOR QC-LDPC CODES

A. The QC-ITBA

The ITBA described in the previous section can be applied to any LDPC code. However, if the LDPC code is in QC form, the algorithm for locating cycles can be further simplified. A code is QC if it has a parity-check matrix \( H \) which is an array of circulants of equal size, where each row in a circulant is a cyclic shift of the row above it by one place to the right and the first row is the cyclic shift of the last row one place to the right. In most constructions of QC-LDPC codes, each circulant either has a single 1 in each row, in which case it is a circulant permutation matrix (CPM), or is a zero matrix (ZM). In particular, let \( H = [A_{s,r}]_{0 \leq s < J, 0 \leq r < L} \) be a \( J \times L \) array of \( p \times p \) matrices, \( A_{s,r} \)'s, which are either CPMs or ZMs. Then, \( H \) is a parity-check matrix of a QC-LDPC code of length \( n = Lp \).

For \( 0 \leq s < J \) and \( 0 \leq r < L \), suppose that \( A_{s,r} \) is a CPM in which the \((i,j)\) entry is 1, where \( 0 \leq i,j < p \). All other entries in the \( i\)-th row and in the \( j\)-th column are zeros. We use the pair of two ordered tuples, \( (s,r),(i,j) \), to indicate the location \((ps+i,pr+j)\) of this 1-entry in the overall array \( H \), where the first tuple, \((s,r)\), identifies the CPM, \( A_{s,r} \), containing this 1-entry and the second tuple, \((i,j)\), identifies the location of the 1-entry within the CPM. Since \( A_{s,r} \) is a CPM, then its \( p \) 1-entries are located at \((s,r),(i+1)_p,j+1)\), for \( 0 \leq l < p \), where \((i)_p\) for any integer \( i \), denotes the nonnegative integer less than \( p \) and equal to \( i \) modulo \( p \).

Let \( G = (U \cup W,E) \) be the Tanner graph of the QC-LDPC code corresponding to the parity-check matrix \( H \). Then \( G \) has \( n = |U| = pL \) VN's, \( m = |W| = pJ \) CNs, and \( E = |\mathcal{E}| \) edges. Let \( g \) be its girth. An edge in \( E \) corresponds to a 1-entry in \( H \). Suppose this entry is at location \((s,r),(i,j)\). We use this location to denote the edge. A cycle in \( G \) of length \( 2k \) has the form \( ((s_0,r_0),(i_0,j_0) \), \( (s_0,r_1),(i_0,j_1) \), \( (s_1-r_0),(i_1,j_0) \) \). Then, for \( 1 \leq l < p \), \( (s_0,r_0),(i_0+l_j),(j_0+l_i) \), \( (s_1-r_0),(i_0+l_j),(j_0+l_i) \), \( (s_2),(i_1+l_j),(j_1+l_i) \), \( \ldots \), \( (s_{k-1},r_0),(i_{k-1}+l_j),(j_{k-1}+l_i) \) \) denotes \( p-1 \) other cycles of the same length in \( G \).

We partition the set \( U \) of \( pL \) VN's into \( L \) subsets \( \mathcal{U}^{(0)}, \mathcal{U}^{(1)}, \ldots, \mathcal{U}^{(L-1)} \), where \( \mathcal{U}^{(r)} \), \( 0 \leq r < L \), is the set of VN's corresponding to the columns \( rp,rp+1, \ldots, rp+p-1 \) in \( H \). Similarly, we partition the set \( W \) of \( pJ \) CN's into \( J \) subsets \( \mathcal{W}^{(0)}, \mathcal{W}^{(1)}, \ldots, \mathcal{W}^{(J-1)} \), where \( \mathcal{W}^{(s)} \), \( 0 \leq s < J \), is the set of CN's corresponding to the rows \( sp,sp+1, \ldots, sp+p-1 \) in \( H \). If there is a cycle of length \( 2k \) passing through one node in \( \mathcal{U}^{(r)} \) or \( \mathcal{W}^{(s)} \), then this cycle identifies \( p-1 \) other cycles of length \( 2k \) passing through the nodes in \( \mathcal{U}^{(r)} \) or \( \mathcal{W}^{(s)} \). Therefore, for each subset \( \mathcal{U}^{(r)} \) or \( \mathcal{W}^{(s)} \), we only need to locate cycles passing through one node in that subset in order to determine all cycles passing through all other nodes in the subset. This leads to an improved message-passing algorithm for QC-codes, which we denote by the QC-ITBA.

Consider the special case in which \( J = L \) and \( H = [A_{s,r}]_{0 \leq s,r < L} \) is such that \( A_{s+r,l} \) is a CPM for \( 1 \leq l < L \). Then, each row-block of the array \( H \) consisting of \( L \) circulants, each of size \( p \times p \), is the cyclic shift of the row-block above it to the right by the size of one circulant, i.e., by \( p \) positions, and the first row-block is the cyclic shift of the last row-block by one circulant. In this case, we say that the parity-check matrix has a row-block cyclic structure. An example of such a parity-check matrix can be found in [2, p. 487] and [11, p. 2432]. If \( ((s_0,r_0),(i_0,j_0)) \), \( ((s_0,r_1),(i_0,j_1)) \), \( \ldots \), \( ((s_{L-1},r_0),(i_{L-1},j_0)) \) is a cycle of length \( 2k \), then for \( 1 \leq l < L \), \( ((s_0+l,r_0+l),(i_0+l,j_0),(i_0,j_1)) \), \( ((s_0+l,r_0+l),(i_0+l,j_0),(i_0,j_1)) \), \( \ldots \), \( ((s_0+l,r_0+l),(i_0+l,j_0),(i_0,j_1)) \) \) denotes \( L-1 \) other cycles of the same length in \( G \). Hence, when applying the QC-IMPA to a QC-LDPC code whose parity-check matrix has a row-block cyclic structure, the algorithm can be further simplified since we need to only locate cycles passing through one VN to determine all cycles in \( G \).

B. Complexity of the QC-ITBA

To find all short cycles of lengths up to \( 2k \), the computational complexity of the QC-ITBA is \( O(n^2d_u^3d_w^3/p) \), while the memory usage is \( O(knd_u^2d_w^2/p) \), which is the same as that of the ITBA.
The computational complexity of the QC-ITBA is $1/p$ of that of the ITBA when applied to a $(d_u, d_v)$-regular QC-LDPC code. If the parity-check matrix has a row-block cyclic structure, then a further reduction of computational complexity by a factor of $L$ is possible. Notice, however, that the ITBA can be applied to any LDPC code while the QC-ITBA can only be applied to QC-LDPC codes.

C. The QC-ITBA for Breaking Cycles

With the locations of cycles provided by the ITBA, we can break a certain number of cycles to get a new graph with a larger girth or a smaller number of cycles of short lengths, or both. For a QC-LDPC code, we require that the incident matrix of the new graph after breaking cycles still maintains the QC structure. In particular, the resulting LDPC code is QC. Therefore, if an edge connecting a variable node $u \in U(r)$ and a check node $w \in V(l(s))$ is removed, then all edges connecting variable nodes in $U(r)$ and check nodes in $V(l(s))$ are removed. Since we know the locations of cycles, we can break cycles through which these cycles pass, where for brevity, a cycle with an edge connecting $u \in U(r)$ and $w \in V(l(s))$ is said to pass through the CPM $A_{s,r}$.

The way described above to break cycles is equivalent to the masking technique [2], [12]. Masking can be modeled mathematically as follows. Let $Z = [z_{s,r}]_{0 \leq s < r \leq L}$ be a $L \times L$ matrix with zeros and ones as entries. Masking a $L \times L$ array $H$ of CPMs/ZMs of size $p \times p$ with $Z$ results in the matrix $H_{mask} = Z \otimes H = [z_{s,r}A_{s,r}]_{0 \leq s < r \leq L}$, where $z_{s,r}A_{s,r} = A_{s,r}$ if $z_{s,r} = 1$ and $z_{s,r}A_{s,r} = 0$ (a $p \times p$ ZM) if $z_{s,r} = 0$. Masking is defined by the masking matrix $Z$. We describe an algorithm for removing edges corresponding to CPMs and give a method to design a masking matrix which can increase the girth of the Tanner graph of a QC-LDPC code after masking.

The algorithm can increase the girth of a Tanner graph, from $g$ to $g+2$. To obtain a larger girth, we can apply the algorithm several times. In order not to disconnect the graph or make the degrees of some VNs too small and to maintain the regularity of the graph as much as possible, we specify two parameters $N_{mask, column}$ and $N_{mask, row}$ in the algorithm: $N_{mask, column}$ is the maximum number of masked CPMs in each column-block, and $N_{mask, row}$ is the maximum number of masked CPMs in each row-block. The algorithm can be described in the following steps:

Step 1 Use the QC-ITBA to calculate the number of cycles of length $g$ passing through each CPM in $H$.
Step 2 Sum up the number of cycles passing through the CPMs in each column-block.
Step 3 Choose a column-block that has fewer than $N_{mask, column}$ masked CPMs and has the largest number of cycles passing through its CPMs, and then choose a CPM in that column-block such that its row-block has fewer than $N_{mask, row}$ masked CPMs and has the largest number of cycles passing through it.
Step 4 Remove all the edges corresponding to the CPM, i.e., mask the CPM found in Step 3, and if there are other CPMs through which the broken cycles pass, then reduce the number of cycles passing through these CPMs.

Step 5 If the number of cycles of length $g$ passing through all the CPMs is zero, stop the algorithm and declare a success; otherwise, if all the column-blocks have $N_{mask, column}$ masked CPMs or all the row-blocks have $N_{mask, row}$ masked CPMs, stop the algorithm and declare a failure, otherwise, go to Step 2.

Example 2. Applying the code construction method in [13], we construct a $4 \times 8$ array $H$ of CPMs of size $330 \times 330$:

$$H = \begin{bmatrix}
39 & 123 & 79 & 294 & 328 & 297 & 68 \\
139 & 79 & 126 & 75 & 158 & 295 & 148 & 274 \\
206 & 209 & 287 & 324 & 163 & 129 & 202 & 325 \\
181 & 271 & 275 & 143 & 208 & 10 & 173 & 284
\end{bmatrix},$$

where the $(s, r)$ entry, which assumes a value in $\{0, 1, \ldots, 329\}$, represents the location of the 1-entry in the first row of the CPM $A_{s,r}$, $0 \leq s < 4$ and $0 \leq r < 8$. The null space of $H$ gives a $(4, 8)$-regular (2640, 1323) QC-LDPC code, denoted by $C$. The Tanner graph of the code $C$ has girth 6 and contains 990 cycles of length six, 24,750 cycles of length eight, and 389,400 cycles of length ten.

By applying the proposed cycle breaking algorithm to the above array $H$, we can obtain a sequence of Tanner graphs with girths up to 16. The incident matrices of these new graphs give a sequence of QC-LDPC codes. In this example, we obtain nine QC-LDPC codes, denoted by $C_i$, with $1 \leq i \leq 9$. The corresponding masking matrices for those codes are denoted by $Z_i$, where $1 \leq i \leq 9$, and listed in the following:

$$Z_1 = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix},$$

$$Z_2 = \begin{bmatrix}
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix},$$

$$Z_3 = \begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix},$$

$$Z_4 = \begin{bmatrix}
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix},$$

$$Z_5 = \begin{bmatrix}
1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix},$$

$$Z_6 = \begin{bmatrix}
1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}.$$
Using masking, we can break cycles in the Tanner graph of cyclic codes resulting in additional savings in complexity. The improvement can be further adapted to quasi-cyclic codes over the trellis-based algorithm of [8] to find cycles in bipartite graphs. A code may not perform well upon which decoding is applied leading to codes with better performance.

IV. CONCLUSION

In this paper, we presented an improvement in complexity over the trellis-based algorithm of [8] to find cycles in bipartite graphs. The improvement can be further adapted to quasi-cyclic codes resulting in additional savings in complexity. Using masking, we can break cycles in the Tanner graph of a QC-LDPC code once these cycles are found. This enables the code-designer to increase the girth of the Tanner graph upon which decoding is applied leading to codes with better performance.

REFERENCES

Fig. 2. (a) Bit error performances of the ten QC-LDPC codes given in Example 2 and the code $C_{\text{mask}}$ in [13] decoded with MSA with 50 iterations; and (b) Block error performances of the ten QC-LDPC codes given in Example 2.
<table>
<thead>
<tr>
<th>$C_i$</th>
<th>(length, dimension)</th>
<th>girth</th>
<th>$N_{(6)}$</th>
<th>$N_{(8)}$</th>
<th>$N_{(10)}$</th>
<th>$N_{(12)}$</th>
<th>$N_{(14)}$</th>
<th>$N_{(16)}$</th>
<th>$d_u$</th>
<th>$d_w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>(2640, 1323)</td>
<td>6</td>
<td>990</td>
<td>24,750</td>
<td>389,400</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>$C_1$</td>
<td>(2640, 1321)</td>
<td>8</td>
<td>0</td>
<td>13,860</td>
<td>193,050</td>
<td>3,148,750</td>
<td>-</td>
<td>-</td>
<td>3.4</td>
<td>7.8</td>
</tr>
<tr>
<td>$C_2$</td>
<td>(2640, 1320)</td>
<td>8</td>
<td>0</td>
<td>330</td>
<td>16,500</td>
<td>197,780</td>
<td>-</td>
<td>-</td>
<td>3.4</td>
<td>5.6</td>
</tr>
<tr>
<td>$C_3$</td>
<td>(2640, 1320)</td>
<td>8</td>
<td>0</td>
<td>330</td>
<td>8,250</td>
<td>88,220</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>$C_4$</td>
<td>(2640, 1320)</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>10,230</td>
<td>115,610</td>
<td>-</td>
<td>-</td>
<td>2.3</td>
<td>5.6</td>
</tr>
<tr>
<td>$C_5$</td>
<td>(2640, 1320)</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>9,570</td>
<td>117,260</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>4.6</td>
</tr>
<tr>
<td>$C_6$</td>
<td>(2640, 1320)</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>9,570</td>
<td>95,040</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>5.6</td>
</tr>
<tr>
<td>$C_7$</td>
<td>(2640, 1320)</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13,860</td>
<td>68,970</td>
<td>-</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>$C_8$</td>
<td>(2640, 1320)</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>660</td>
<td>7,920</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>$C_9$</td>
<td>(2640, 1321)</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2,970</td>
<td>2</td>
<td>3.4</td>
</tr>
<tr>
<td>$C_{\text{mask}}$</td>
<td>(2640, 1320)</td>
<td>8</td>
<td>0</td>
<td>990</td>
<td>8,580</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

**TABLE I**
PARAMETERS OF THE TEN QC-LDPC CODES GIVEN IN EXAMPLE 2