International Symposium on Information Theory and its Applications, ISITA2004
Parma, Italy, October 10–13, 2004

A Cycle Search Algorithm for an LDPC Code Design

Sang Hyun LEE†, Kwang Soon KIM‡, Yun Hee KIM† and Jae Young AHN†

† Telecommunications Research Laboratory
Electronics and Telecommunications Research Institute
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350 Korea
E-mail: sanghlee@ieee.org

‡ Department of Electrical and Electronic Engineering
Yonsei University
134 Sinchon-dong, Seodaemun-gu, Seoul, 120-749 Korea
E-mail: kskim@ieee.org

Abstract

A message-passing based search algorithm for cycles in a Tanner graph is proposed for designing good low-density parity-check (LDPC) codes. By applying the message-passing algorithm with only two message alphabets to a cycle search, we can easily perform the cycle search for a good LDPC code with an existing message-passing decoder.

1. INTRODUCTION

An LDPC code is a linear code defined by a sparse parity check matrix which is associated with a corresponding Tanner graph containing two types of node denoted as bit nodes and check nodes. In [1], it was shown that the performance of an LDPC code, associated with a random Tanner graph under the assumption of a cycle-free structure and message-passing decoding, was very close to the Shannon capacity limit. However, a relatively short-length code, whose associated Tanner graph is not cycle-free, suffers from significant performance degradation [2]. Therefore, constructing a Tanner graph for a good code requires a careful elimination of short-length cycles to enhance its decoding capability, especially for that with a relatively short length. A cycle of length $2d$ in a Tanner graph is a set of d bit nodes and d check nodes connected by edges such that there exists a path that travels through every node in the set without traversing an edge twice. Although a cycle search is a well-known problem to find the length of the shortest cycle, existing algorithms are basically tree-based searches whose search trees often grow exponentially. Thus, a large search depth would be computationally impractical with a tree-based search algorithm. In this paper, we present a search algorithm to find the length of the shortest cycle using the message-passing algorithm, which has a polynomial-time complexity. In addition, the proposed algorithm can be directly implemented by using an existing message-passing decoder with slight modification of the node update equations of the decoder.

2. PROPOSED ALGORITHM

The basic idea of the proposed algorithm is to run a message-passing algorithm on a Tanner graph with only '*one*' and '*zero*' as message alphabets. It indicates the length of the shortest path in the Tanner graph that the smallest number of iterations for a message sent by some node to propagate back to the node itself. Then, we can examine the existence of a cycle as follows: i) send a particular message to an edge of a node, ii) check whether that message comes back to the node, and iii) find the length of a cycle by counting the number of iterations until the message comes back. Although the bit node update equation is different from the check node update equation in an ordinary message-passing LDPC decoder, the proposed algorithm requires only a single node update equation since it does not need a classification of nodes and allows only two message alphabets. The new node update equation is defined as

$$y_{ij} = \left(\bigoplus_{j' \in E_i} x_{ij'} \right) \bigotimes x_{ij}, \tag{1}$$

where x_{ij} is the input message of the jth edge entering the ith node, y_{ij} is the output message of the jth edge leaving the ith node, and the operations $\oplus$ and $\otimes$ denote the logical OR and XOR operations, respectively. Also, E_i denotes the set of edges connected to the ith node. For each node, if all input messages are '0', then all output messages are '0'. If the input messages through some edges are '1', then the output messages through those edges are '0', while the output messages through other edges are '1'.

Now, consider an LDPC code design method by examining the length of the shortest cycle on an edge-by-edge basis. In this scheme, edges are sequentially added

to construct a Tanner graph. Once an edge is added to the graph being constructed, a search for cycles caused by the edge is performed. Initially, the output message, leaving a bit node, of the edge under examination is set to '1' while the output messages of other edges of the node are set to '0'. During the message-passing processing, each node accepts messages through the edges connected to the node itself. An input message '1' to a node means that the node and the edge where the message came from are included in a path that might form cycles. Since the message '1' cannot pass through an edge more than once, counting a cycle twice in two opposite directions is avoided. Thus, the number of cycles is not doubly counted. We refer to a single check node processing and a single bit node processing together as a single iteration. If, at each iteration, there exists any nonzero message coming into the bit node that the edge under examination is connected to, it means that the initial message comes back to where it is initially sent. Thus, a cycle in the graph can be detected with the proposed algorithm.

In Fig. 1, the outline of the proposed algorithm is shown. Here, l_0 is the edge index of the newly added edge, N_{iter} is the maximum number of iteration, and E is the set of all edge indices in the graph. The indices of the check node and the bit node connected to the pth edge are denoted as i_p and j_p, respectively. The function IND maps each edge index to its check and bit node indices. Also, $E(i_l)$ and $E(j_l)$ are the set of all edges connected to the check node i_l and the bit node j_l, respectively. Initially, the bit node message $n(l_0)$ of the newly added edge is set to '1' and other messages are set to '0'. During the message-passing processing, the check node message $m(l)$ and the bit node message $n(l)$ of the lth edge are calculated by using (1) or (2). If, at each iteration, the number of nonzero input message to the bit node connected to the new edge is nonzero, then a cycle is detected. The number of nonzero messages entering the node represents the number of the shortest cycles caused by the new edge. Note that cycles of length $2k$ are found with k iterations. We can determine the exact number of cycles by redefining the operations $\oplus$ and $\otimes$ in (1) as

$$x \oplus y = x + y \quad x, y \in \mathbb{Z}^+ \cup \{0\} \tag{2}$$

$$\begin{cases} 0 \otimes 0 = 0 \\ 0 \otimes x = x \\ x \otimes 0 = x \\ x \otimes y = 0 \end{cases} \quad x, y \in \mathbb{Z}^+ \tag{3}$$

where $+$ is a simple addition and $\mathbb{Z}^+$ is the set of all positive integers. Then, a message means the number

Input : l_0, N_{iter}, E

1: if (the new edge is not the first edge of two nodes which it is connected to) {
2: set $k=0$, $Flag\_cycle=0$, $Length\_cycle=0$, $(i_p, j_p) = IND(l_p)$ $l_p \in E$
3: set $n(l_0)=1$, $n(l)=0$ $l \in E \setminus \{l_0\}$
4: while ($k < N_{iter}$) {
5: for ($l=0; l<|E|; l++$) $m(l) = \left(\bigoplus_{l' \in E(i)} n(l') \right) \otimes n(l)$
6: if ($\sum_{l \in E} m(l) == 0$) break
7: for ($l=0; l<|E|; l++$) $n(l) = \left(\bigoplus_{l' \in E(j)} m(l') \right) \otimes m(l)$
8: if ($\sum_{l \in E} n(l) == 0$) break
9: if ($\sum_{l \in E(j_0)} n(l) > 0$) $Flag\_cycle=1$, $Length\_cycle = 2*k$ break
10: $k = k+1$
11: }
12: }

Output : $Flag\_cycle$, $Length\_cycle$

Figure 1: The outline of the proposed cycle detection method.

of multiple cycles associated with the current searching path and the exact number of length-$2k$ cycles containing the starting variable node is the sum of all nonzero input messages returning the starting variable node at the kth iteration.

Also, some nodes other than the starting bit node may have multiple nonzero input messages during iterations, which implies that the edges that have already been included in the previous graph form cycles. In addition, for detecting larger-length cycles containing the starting bit node, one can set the output messages of the starting bit node at the next iteration to zero and continue the iteration until another nonzero message arrived at the starting bit node.

While above outline of the algorithm is described in an edge-by-edge manner, it is straightforward to apply the proposed algorithm to a node-by-node LDPC code design. To do so, we employ a message vector of the length equal to the number of edges connected to the bit node under examination. Initially, only the ith element of the message vector for the ith edge connected to the node under examination is set to '1' for all message vectors sent from the node, while other message vectors sent from other nodes are set to all zero vectors. The operations between two message vectors are defined as the element-wise calculation of (1). Note that a cycle could be counted more than once in this case.

3. COMPLEXITY AND EXAMPLE

We will now roughly analyze the computational complexity of the proposed algorithm. Basically, the

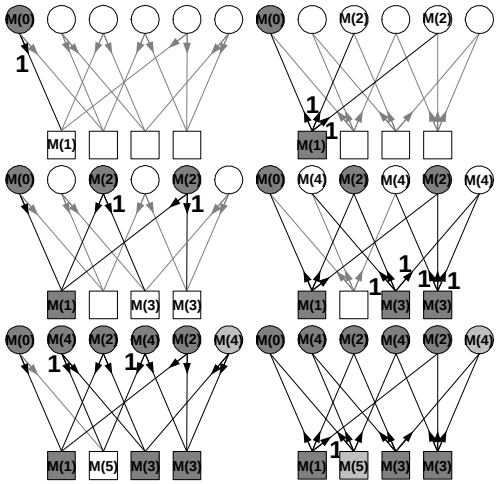

Figure 2: An example of search for cycle with length 6.

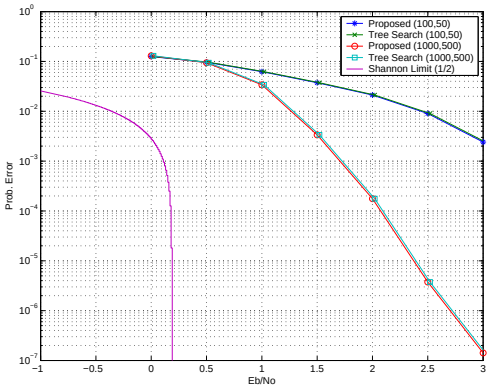

Figure 3: An example of several codes designed by both algorithms

Table 1: Search results of several example graphs.

Avg. bit node degree	3	3	6
Code size (n,k)	(100, 50)	(1000, 500)	(100, 50)
Detected cycles	1740	12490	3420
	Required memory units		
Proposed algorithm	156	1541	468
Algorithm in [3]	172	2953	559
	Elapsed time		
Proposed algorithm	1.266 s	208 s	1.969 s
Algorithm in [3]	7.375 s	2740 s	14.340 s

complexity of the proposed algorithm is the same as that of the LDPC message-passing decoding. If the maximum degree of nodes is d_{max}, the node processing for a single edge, which can be implemented by using very simple logical operations, requires $O(d_{max})$ operations, where $O(f(m))$ denotes the case where the complexity count is upper-bounded by $cf(m), m \geq m_0$ for a positive constant c and m_0 . Thus, the entire node processing at a single node requires $O((d_{max})^2)$ operations. Therefore, a cycle search for a single node requires approximately $O(L(d_{max})^2 m)$ operations, where L is the number of iterations and m is the number of bit nodes. Finally, processing for the entire bit nodes requires roughly $O(L(d_{max})^2 m^2)$ operations. Thus, the proposed algorithm has the polynomial-time complexity and is quite favorable in the context of the computational complexity as compared to the tree-based search algorithms.

In Fig. 2, a search example for a length-six cycle associated with the leftmost edge in a (3,2) regular Tanner graph is shown. Here, each row means an iteration. The left (right) column is bit (check) node processings at each iteration. Circles and squares in the graph denote the bit nodes and the check nodes, respectively. Dark gray circles, squares and edges indicate that they have already included in the searching path. Only nonzero output messages at the nodes are shown. Light gray nodes at the third iteration represent that the multiple searching paths have simultaneously visited. Multiple paths passing through such nodes form another small cycle. Therefore, it is easily found that two smaller cycles are attached to the cycle of length 6.

Table 1 compares the proposed algorithm to the algorithm in [3] with the maximum number of required memories and the elapsed times for searching all cycles of length up to 12 in several Tanner graphs. From the results, it is shown that the elapsed time increases as the node degree or the code length grows. Also, one can see that, while the same number of cycles is detected, the proposed algorithm takes less searching time and smaller length of memory queue. Although both algorithms have similar performances for the case of searching for the shortest cycle in the graph, the proposed algorithm improves the searching time significantly for the case of searching for all cycles up to a pre-defined length included in a Tanner graph. Fig. 3 depicts the performance of the codes designed by both algorithms with the parameters of above comparison. The result shows that the proposed algorithm gives identical performance with less code searching time.

4. CONCLUSION

In this paper, a simple cycle search scheme using a message-passing algorithm was proposed for designing a good LDPC code. The proposed scheme can de-

tect the existence of cycles and compute the length of detected cycle by propagating a particular message through the Tanner graph and monitoring messages coming back to the node. Since the proposed algorithm is very similar to the message-passing decoding, it can be directly implemented by using an existing message-passing decoder without additional cost. Also, the proposed algorithm can easily adopt different kind of criteria for an LDPC code design, such as ACE in [4] etc., with a slight modification in the node update equations.

References

[1] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 638-656, Feb. 2001.

[2] D. J. C. MacKay, "Good error correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399-431, Mar. 1999.

[3] Y. Mao and A. H. Banihashemi, "A heuristic search for good low-density parity-check codes at short block lengths," *Proc. IEEE Int. Conf. Comm.*, vol. 1, pp. 11-14, Helsinki, Finland, June 2001.

[4] T. Tian, C. Jones, J. D. Villasenor and R. D. Wesel, "Construction of irregular LDPC codes with low error floors," *Proc. IEEE Int. Conf. Comm.*, vol. 5, pp. 3125-3129, Anchorage, AK, U.S.A., May 2003.