Iterative Decoding of Linear Block Codes: A Parity-Check Orthogonalization Approach

Sundararajan Sankaranarayanan, *Student Member, IEEE,* and
Bane Vasić, *Senior Member, IEEE*

***Abstract*—It is widely accepted that short cycles in Tanner graphs deteriorate the performance of message-passing algorithms. This discourages the use of these algorithms on Tanner graphs (TGs) of well-known algebraic codes such as Hamming codes, Bose–Chaudhuri–Hocquenghem codes, and Reed–Solomon codes. Yedidia *et al.* presented a method to generate code representations suitable for message-passing algorithms. This method does not guarantee a representation free of four-cycles. In this correspondence, we present an algorithm to convert an arbitrary linear block into a code with orthogonal parity-check equations. A combinatorial argument is used to prove that the algorithm guarantees a four-cycle free representation for any linear code. The effects of removing four-cycles on the performance of a belief propagation decoder for the binary erasure channel are studied in detail by analyzing the structures in different representations. Finally, we present bit-error rate (BER) and block-error rate (BLER) performance curves of linear block codes under belief propagation algorithms for the binary erasure channel and the additive white Gaussian noise (AWGN) channel in order to demonstrate the improvement in performance achieved with the help of the proposed algorithm.**

***Index Terms*—Auxiliary checks, auxiliary variables, belief-propagation algorithm, four-cycles, Tanner graphs (TGs).**

I. Introduction

Linear block codes have been a subject of great interest since the discovery of Hamming codes in the late 1940s. The design of early codes was based on algebraic structures, and these structures were exploited in developing *bounded-distance decoders*. The efficiency of such decoders is dependent on the minimum-distance of the code. In the early 1960s, Gallager [4] introduced a class of linear block codes called *low-density parity-check* (LDPC) codes, and this was one of the first instances of codes defined on graphs. In his graphical representation of the code, vertices correspond to codeword bits and edges correspond to linear check constraints. Such a graphical representation grew out of the necessity to facilitate a class of decoding algorithms often referred to as *message-passing* algorithms. A decoder based on a message-passing algorithm is not a bounded-distance decoder, and it has been shown that long LDPC codes approach Shannon capacity under such a decoder. A great interest in the general study of codes defined on graphs emerged after Tanner's paper in 1981.

In [9], Tanner founded the general study of codes on graphs, introduced generalized constraints, and proved the optimality of the sum–product algorithm on cycle-free graphs. The next development in the study of codes on graphs was achieved by introducing *state variables* into *Tanner graphs* (TGs) [12], [11], and this allowed to establish connections to trellises and to turbo codes. In [8], [1], and [5], the authors have presented important results on minimal trellis-like representations of linear block codes that minimize one or more complexity measures. In [6], Kschischang *et al.* further abstracted the idea of codes on graphs to encompass representation of general functions, and such graphs are referred to as *factor graphs*.

Manuscript received June 15, 2004; revised May 20, 2005. This work was supported by the National Science Foundation under Grant CCR–0208597 and by INSIC. The material in this correspondence was presented in part at the 42nd Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL, September/October 2004.

The authors are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721 USA (e-mail: ssundar@ece.arizona.edu; vasic@ece.arizona.edu).

Communicated by K. A. S. Abdel-Ghaffar, Associate Editor for Coding Theory.

The state variables in TGs, introduced by Wiberg [12], are auxiliary variables that are introduced by the designer to improve code realization in some sense. These variables are not transmitted over the channel, and hence, they are not observed at the receiver end. In [7], MacKay discussed *generalized parity-check matrix* (GPCM) representations of linear block codes obtained by introducing additional columns and additional rows. These additional columns corresponding to auxiliary variables can be interpreted as state variables because they are not transmitted over the channel. In [13], Yedidia *et al.* presented an algorithm to generate GPCM representations of codes that have the following characteristics: small number of ones in each row, large number of ones in each column, and a small number of four-cycles. With empirical results, they demonstrated an improvement in bit-error rate (BER) performance of codes, transmitted over *binary erasure channel* (BEC) or *additive white Gaussian noise* (AWGN) channel, under message-passing algorithms applied on these new representations. This paper posed an interesting problem of finding a GPCM of a code that is suitable for message-passing algorithms.

In Section II, we present an algorithm that is specific to removing four-cycles in a TG of a linear block code. Such an effort is required to facilitate *message-passing decoders* (MPDs) of well-known algebraic codes. It is well known that the performance of a code under an MPD is affected by the cycle structure of the corresponding TG. In [6], Kschischang dealt with techniques of clustering (or merging) and stretching nodes in order to remove short cycles in a TG of the code. Unfortunately, these techniques require passing message-vectors along edges of the modified TG and, hence, increasing the complexity of MPDs. The algorithm presented in this correspondence preserves binary message-passing and so, the increase in decoding complexity is not significant. Although the idea of introducing *auxiliary variables* and *auxiliary checks* to the parity-check matrix of the code is similar to the one presented in [13], the approach taken in achieving a four-cycle-free representation is distinct. In Section III, we prove that this technique is guaranteed to remove all four-cycles in a TG of the code. Now, instead of the conventional TG of the code, the four-cycle-free graph (or the modified graph) can be used to iteratively decode on BEC and AWGN channel. In Section IV, we prove that the error rate performance of the iterative erasure decoder working on the modified graph is at least as good as that obtained by decoding on the conventional graph of the code. Also, we present simulation results of the BER and block-error rate (BLER) performances of an algebraic linear code over the BEC. Theoretical analysis of the effectiveness of GPCM representations in improving the performance of an MPD for AWGN channels is rather difficult. Hence, we resort to empirical results to demonstrate the effectiveness of four-cycle-free representations. The results presented in Section V demonstrate improvements in BER and BLER performances of algebraic codes obtained by decoding iteratively on the modified graphs. Finally, in Section VI, we conclude with a brief discussion on generalizing this technique in order to remove cycles of any specified length in a graph.

II. Algorithm to Remove Four-Cycles in TGs

Consider a linear block code $\mathcal{C}$ of length n defined as the solution-space (over $\mathbb{F}_2$) of the system of linear equations $\boldsymbol{Hx} = \boldsymbol{0}$, where $\boldsymbol{H}$ is an $m \times n$ binary matrix. The bipartite graph representation $\mathcal{G}$ of $\mathcal{C}$ is assumed to have at least one four-cycle. It is necessary to identify four-cycles in $\mathcal{G}$ before they can be removed. The four-cycle structure of $\mathcal{G}$ is obtained from the square (over $\Re$) of the adjacency matrix $\boldsymbol{A}$ of the graph. The adjacency matrix and its square can be written as

$$\boldsymbol{A} = \begin{pmatrix} 0 & \boldsymbol{H} \\ \boldsymbol{H}^T & 0 \end{pmatrix} \quad \text{and} \quad \boldsymbol{A}^2 = \begin{pmatrix} \boldsymbol{H}\boldsymbol{H}^{\mathrm{T}} & 0 \\ 0 & \boldsymbol{H}^{\mathrm{T}}\boldsymbol{H} \end{pmatrix}.$$

The element in position (i, j) of $\boldsymbol{A}^2$ is the number of distinct paths of length 2 between the nodes a_i and a_j. In $\boldsymbol{A}^2$, the component $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ calculates the number of paths of length 2 between variable nodes, and the component $\boldsymbol{H}\boldsymbol{H}^{\mathrm{T}}$ calculates the number of paths of length 2 between check nodes. These component matrices are symmetric. In the bipartite graph $\mathcal{G}$, there is no path of length 2 between a variable node and a check node. Also, any two distinct paths of length 2 between two nodes contribute to a four-cycle. The total number of four-cycles in $\mathcal{G}$ is denoted by $\psi(\mathcal{G})$.

Theorem 1: In a simple bipartite graph $\mathcal{G}$, the total number of four-cycles $\psi(\mathcal{G})$ is given by

$$\psi(\mathcal{G}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \binom{(\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H})_{ij}}{2}.$$

Proof: Assume that there are l, say $l > 1$, distinct paths of length 2 between distinct variable nodes x_i and x_j. This means that the element in position (i, j) of $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ is l. A pair of distinct paths between these nodes contribute to a four-cycle, and there are $\binom{l}{2}$ different ways to choose such a pair. Hence, the total number of four-cycles in $\mathcal{G}$ can be obtained by summing the quantities $\binom{(\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H})_{ij}}{2}$ over every (ordered) pair of variable nodes. $\square$

Corollary 2: In a complete bipartite graph $\mathcal{K}_{m,n}$, the total number of four-cycles is $\psi(\mathcal{K}_{m,n}) = \binom{n}{2}\binom{m}{2}$.

It is easy to show that the element in position (i, i) of $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ is equal to the degree of the ith variable node. In [13], the first step of the algorithm involves the process of taking intersections of all pairs of the constraint sets to identify four-cycles in the code. In this correspondence, we compute the product $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ to identify the four-cycles in the code. Also, it will be evident from the following discussion that we do not consider all the intersection sets. Now equipped with the tools to identify the four-cycle structure of the graph, it is easy to remove all four-cycles in $\mathcal{G}$ as shown below.

Assume that there is a four-cycle involving variables x_u and x_v. Taking a componentwise product of the uth and vth columns of $\boldsymbol{H}$, we can determine the two check nodes, say S_i and S_j, that are involved in the four-cycle. Let $x_u - S_i - x_v - S_j - x_u$ represent the four-cycle. In order to remove this four-cycle, replace the term $x_u + x_v$ in S_i and S_j with an auxiliary variable $x_{u,v}$. This is equivalent to appending a new column to the original matrix $\boldsymbol{H}$, and replacing elements in positions $(i, u), (j, u), (i, v)$, and (j, v) of $\boldsymbol{H}$ with zeros. In addition, an auxiliary check equation of the form $x_u + x_v + x_{u,v} = 0$ is added to the original set of constraints. This is equivalent to appending a new row to the original matrix $\boldsymbol{H}$ that corresponds to the auxiliary check equation. If these variable nodes are involved in more than one four-cycle, then every check equation containing the term $x_u + x_v$ is replaced with the auxiliary variable $x_{u,v}$. This process removes all four-cycles involving variables x_u and x_v, and results in a modified parity-check matrix with an additional (auxiliary) constraint and variable in comparison to $\boldsymbol{H}$. By introducing an auxiliary variable, we have not changed the code rate because this variable is not transmitted over the channel. The introduction of the auxiliary variable results in the extension of the vector subspace $\mathcal{C}$ to a vector subspace $\mathcal{C}_1$. For any vector $\boldsymbol{x} = \langle x_1, \ldots, x_u, \ldots, x_v, \ldots, x_n \rangle$ in $\mathcal{C}$, there is a unique vector $\boldsymbol{x}_1 = \langle x_1, \ldots, x_u, \ldots, x_v, \ldots, x_n, x_u + x_v \rangle$ in $\mathcal{C}_1$. On the other hand, the auxiliary check guarantees that the dimension of $\mathcal{C}_1$ is equal to that of $\mathcal{C}$.

By iteratively applying the procedure on the modified parity-check matrix, it is possible to remove all four-cycles involving actual and/or auxiliary variables. The iterative algorithm is summarized as follows.

```
Initialize:
    • Let $l = 0$.
    • Let $\boldsymbol{H}_l := \boldsymbol{H}$.
    • $\boldsymbol{H}_l$ is a $m_l \times n_l$ matrix such that $m_0 = m$ and $n_0 = n$.
Step 1:
    • Compute ${\boldsymbol{H}_l}^{\mathrm{T}}\boldsymbol{H}_l$.
    • If there is at least one four-cycle in $\boldsymbol{H}_l$, then go to Step 2.
    • If $\boldsymbol{H}_l$ is four-cycle free, then go to Step 4.
Step 2:
    • Let $x_u$ and $x_v$ be a pair of variables involved in a four-cycle.
    • The column vector $\boldsymbol{u} \cdot \boldsymbol{v}$ is the componentwise product of the
      $u$th and $v$th columns of $\boldsymbol{H}_l$.
    • Let $\boldsymbol{B}$ be an $m_l \times n_l$ matrix defined as
          $$\boldsymbol{B} = \left[\underbrace{\overbrace{\mathbf{0}, \ldots, \mathbf{0}}^{u-1}, \boldsymbol{u} \cdot \boldsymbol{v}, \overbrace{\mathbf{0}, \ldots, \mathbf{0}}^{v-u-1}, \boldsymbol{u} \cdot \boldsymbol{v}, \ldots, \mathbf{0}}_{n_l}\right]$$
      where $\mathbf{0}$ is a zero column vector, and $\boldsymbol{u}$ and $\boldsymbol{v}$ are the $u$th and
      $v$th columns of $\boldsymbol{H}_l$.
    • Let
          $$S' := \langle \underbrace{\overbrace{0, \ldots, 0}^{u-1}, 1, \overbrace{0, \ldots, 0}^{v-u-1}, 1, 0, \ldots, 0, 1}_{n_l+1} \rangle$$
      where $S'$ corresponds to the auxiliary check equation.
Step 3:
    • The transformed matrix is written as
          $$\boldsymbol{H}_{l+1} := \left( \begin{array}{c|c} \boldsymbol{H}_l + \boldsymbol{B} & \boldsymbol{u} \cdot \boldsymbol{v} \\ \hline \multicolumn{2}{c}{S'} \end{array} \right).$$
      The addition operation is defined over the binary field.
    • The dimension of $\boldsymbol{H}_{l+1}$ is $(m_l + 1) \times (n_l + 1)$.
    • $l \longleftarrow l + 1$.
    • Go to Step 1.
Step 4:
    • $\boldsymbol{H}' \longleftarrow \boldsymbol{H}_l$
    • The matrix $\boldsymbol{H}'$ is free of four-cycles with dimension $M \times N$.
```

In the algorithm, Step 1, Step 2, and Step 3 constitute an iteration. After the lth iteration, one auxiliary variable node and an auxiliary check node are inserted into the graph of $\boldsymbol{H}_{l-1}$. The weight of the augmented row is 3, and the weight of the augmented column is one more than the weight of the column-vector $\boldsymbol{u} \cdot \boldsymbol{v}$ computed in Step 2. In Section III, we show that in fact this algorithm guarantees to remove all four-cycles in the original graph $\mathcal{G}$ after a finite number of iterations. The matrix $\boldsymbol{H}'$, that is free of four-cycles, is referred to as the output GPCM of $\mathcal{C}$; $\mathcal{G}'$ is the bipartite graph corresponding to $\boldsymbol{H}'$. Steps 2 and 3 of this algorithm are similar to Steps 2–4 of the algorithm presented in [13]. Unlike in [13], we avoid creating a four-cycle by allowing only one way to break down each constraint set. Also, this reduces the complexity of organizing sets into a *partially ordered set* of sets, and decreases the dimension of the output GPCM. The following example is designed to better illustrate the application of the algorithm.

Example 1: Consider a $(7, 4)$ Hamming code, defined by a parity-check matrix $\boldsymbol{H}$ with $n = 7$ variables and $m = 3$ check equations. The ith column of $\boldsymbol{H}$ corresponds to a variable node x_i of the graph $\mathcal{G}$, and the jth row of the matrix corresponds to a check node S_j of $\mathcal{G}$. From the component matrix $\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$, we can compute that there are exactly three four-cycles in $\mathcal{G}$. The parity-check matrix and the component matrix are

$$\boldsymbol{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 2 & 1 & 1 & 2 \\ 1 & 0 & 1 & 1 & 2 & 1 & 2 \\ 0 & 1 & 1 & 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 & 3 \end{pmatrix}.$$

The four-cycle $x_4 - S_1 - x_7 - S_2 - x_4$ in $\boldsymbol{H}$ is removed by introducing an auxiliary variable $x_{4,7}$ and an auxiliary check equation written as $x_4 + x_7 + x_{4,7} = 0$. The introduction of $x_{4,7}$ is equivalent to appending a new column to $\boldsymbol{H}$, and replacing elements in positions $(1, 4), (1, 7), (2, 4)$, and $(2, 7)$ of $\boldsymbol{H}$ with zeros. The matrix $\boldsymbol{H}_1$ obtained after the first iteration is written as

$$\boldsymbol{H}_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

Since $\boldsymbol{H}_1$ is free of four-cycles, it is the desired GPCM of the $(7, 4)$ Hamming code. Thus, in this case, just one auxiliary variable is required to remove three four-cycles. □

For a given graph $\mathcal{G}$, an upper bound on the number of auxiliary variables (or auxiliary checks) required to remove all four-cycles in the graph can be computed from the following observations about the algorithm.

- At each iteration of the algorithm, exactly one auxiliary variable is introduced.
- Just one auxiliary variable $x_{u,v}$ is required to remove any number of four-cycles involving a pair of variables x_u and x_v.

In Section III, we will show that the number of four-cycles in $\mathcal{G}_l$ (output of the lth iteration) is strictly less than that in $\mathcal{G}_{l-1}$. Hence, the total number of auxiliary variables $\Delta(N, n)$ in $\mathcal{G}'$ is upper-bounded by the total number of four-cycles in $\mathcal{G}$

$$\Delta(N, n) = N - n \le \psi(\mathcal{G}).$$

The summation quantity is an upper bound because there is always the possibility of an auxillary variable removing more four-cycles than intended.

III. Why Does the Algorithm Work?

In this section, we present a combinatorial argument to substantiate the claim that the algorithm removes all four-cycles in the original graph $\mathcal{G}$ after a finite number of iterations. Assume that $\mathcal{G}_l$ is the bipartite graph corresponding to $\boldsymbol{H}_l$, obtained after l iterations of the algorithm on the $m \times n$ matrix $\boldsymbol{H}$. The key idea of the proof is to show that the number of four-cycles in $\mathcal{G}_l$ is strictly less than that in $\mathcal{G}_{l-1}$. We prove this key idea in Lemma 3.

Lemma 3: $\psi(\mathcal{G}_l) < \psi(\mathcal{G}_{l-1}), l \in \mathbb{N}$.

Proof: For some l, $\mathcal{G}_l$ is obtained from an iteration of the algorithm on $\mathcal{G}_{l-1}$. Without loss of generality, assume that there are $\binom{p}{2}, p \ge 2$, four-cycles in $\mathcal{G}_{l-1}$ involving variables x_u and x_v. This means that there are exactly p check equations (in $\boldsymbol{H}_{l-1}$) with ones in the uth and the vth columns. Rearrange the rows of $\boldsymbol{H}_{l-1}$ such that the first p rows have ones in the uth and the vth columns. The ith row of the matrix corresponds to the check equation S_i. The general form of $\boldsymbol{H}_{l-1}$ is

$$\begin{pmatrix} h_{1,1} & \dots & 1 & \dots & 1 & \dots & h_{1,n_{l-1}} \\ h_{2,1} & \dots & 1 & \dots & 1 & \dots & h_{2,n_{l-1}} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{p,1} & \dots & 1 & \dots & 1 & \dots & h_{p,n_{l-1}} \\ h_{p+1,1} & \dots & h_{p+1,u} & \dots & h_{p+1,v} & \dots & h_{p+1,n_{l-1}} \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ h_{m_{l-1},1} & \dots & h_{m_{l-1},u} & \dots & h_{m_{l-1},v} & \dots & h_{m_{l-1},n_{l-1}} \end{pmatrix}.$$

The four-cycles involving x_u and x_v are removed by augmenting a new variable node x_{n_l}, and a new check node S_{m_l}. The general form of the resultant matrix $\boldsymbol{H}_l$ is given in

$$\left(\begin{array}{ccccccc|c} h_{1,1} & \dots & 0 & \dots & 0 & \dots & h_{1,n_{l-1}} & 1 \\ h_{2,1} & \dots & 0 & \dots & 0 & \dots & h_{2,n_{l-1}} & 1 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ h_{p,1} & \dots & 0 & \dots & 0 & \dots & h_{p,n_{l-1}} & 1 \\ h_{p+1,1} & \dots & h_{p+1,u} & \dots & h_{p+1,v} & \dots & h_{p+1,n_{l-1}} & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ h_{m_{l-1},1} & \dots & h_{m_{l-1},u} & \dots & h_{m_{l-1},v} & \dots & h_{m_{l-1},n_{l-1}} & 0 \\ \hline 0 & \dots & 1 & \dots & 1 & \dots & 0 & 1 \end{array}\right). \tag{1}$$

Now, let us examine the four-cycle structure of $\boldsymbol{H}_l$, and compare it with that of $\boldsymbol{H}_{l-1}$. If $\mathcal{G}_l$ is free of four-cycles, then the lemma is trivially true. In order to prove the nontrivial case, we have assumed in the following list of arguments that there is at least one four-cycle in $\mathcal{G}_l$.

- The construction guarantees that there is no four-cycle involving x_u and x_v in $\mathcal{G}_l$. On the other hand, there are $\binom{p}{2}$ four-cycles involving these variable nodes in $\mathcal{G}_{l-1}$.
- If there is a four-cycle involving x_u and x_{n_l} in $\mathcal{G}_l$, then this pair of variables should participate in two distinct check equations of $\mathcal{H}_l$. But the algorithm guarantees that there is only one check equation, namely S_{m_l}, that contains this pair of variables. Hence, by contradiction, we can conclude that there is no four-cycle involving this pair of variables. Such an argument can be used to prove the absence of four-cycles involving x_v and x_{n_l}.
- Let $x_{n_l} - S_q - x_s - S_r - x_{n_l}$, where $s \ne u, v$, be a four-cycle in $\mathcal{G}_l$. The general form of $\boldsymbol{H}_l$ reveals that q and r have to take values from $\{1, 2, \dots, p\}$. For every such four-cycle in $\mathcal{G}_l$, there exist two four-cycles, namely, $x_u - S_q - x_s - S_r - x_u$ and $x_v - S_q - x_s - S_r - x_v$, in $\mathcal{G}_{l-1}$.

From the preceding arguments and the fact that $p \ge 2$, we can conclude that the number of four-cycles in $\mathcal{G}_l$ is strictly less than that in $\mathcal{G}_{l-1}$. □

Now we restate the claim of the algorithm as a theorem, and prove it using Lemma 3.

Theorem 4: Any linear binary block code $\mathcal{C}$ has an equivalent bipartite graph representation $\mathcal{G}'$ that is free of four-cycles.

Proof: Let $\boldsymbol{H}$ be the parity-check matrix of $\mathcal{C}$. If the bipartite graph $\mathcal{G}$ corresponding to $\boldsymbol{H}$ is free of four-cycles, then the theorem is trivially true. Otherwise, let $\psi(\mathcal{G})$ be the number of four-cycles in $\mathcal{G}$. After l iterations of the algorithm with $\mathcal{G}$ as the input, we know that the following statement is true from Lemma 3:

$$\psi(\mathcal{G}) > \psi(\mathcal{G}_1) > \psi(\mathcal{G}_2) > \cdots > \psi(\mathcal{G}_l).$$

The number of four-cycles in $\mathcal{G}$ with m check nodes and n variable nodes is upper-bounded by $\binom{n}{2}\binom{m}{2}$, see Corollary 2. Thus, for finite m and n, there exists a finite l in $\mathbb{N}$ such that l iterations of the algorithm with $\mathcal{G}$ as the input result in a four-cycle-free graph $\mathcal{G}'$. Since we started with $\mathcal{G}$, the algorithm guarantees that $\mathcal{G}'$ is an equivalent representation of $\mathcal{C}$. □

IV. Iterative Decoding of Linear Block Codes Over BEC

In BEC, the input alphabet is binary, and the output alphabet, besides symbols 0 and 1, includes an erasure symbol ε. The probability of receiving an ε is called the *erasure probability*. An optimal decoding of a linear block code $\mathcal{C}$ over the channel is achieved by a direct application of Gaussian elimination. The complexity of such an optimal decoder is $\mathcal{O}(n^3)$. In this section, we are interested in a suboptimal erasure decoder, referred to as the MPD, that decodes iteratively by passing messages over the edges of the graph $\mathcal{G}$ of $\mathcal{C}$. The complexity of such a decoder is linear in the length of the code. A *decoding failure* in an MPD is caused exactly when every check equation has more than one variable in erasure. For a fixed $\boldsymbol{H}$ of $\mathcal{C}$, a subset $\mathcal{S}$ of variable nodes such that all neighbors of $\mathcal{S}$ are connected to $\mathcal{S}$ at least twice is called a *stopping set* [2]. A decoding failure occurs precisely when the set of variables in erasure contains a stopping set. The knowledge of all stopping sets of $\boldsymbol{H}$ is sufficient to theoretically compute the error performance of the MPD for a given BEC.

The simplicity of the iterative decoder provides a convenient framework to analyze the effects of removing four-cycles on its performance. Let $\mathcal{G}'$ be a four-cycle-free representation of $\mathcal{C}$. Now $\mathcal{G}'$ can be used to iteratively decode the linear code. In general, the performance of the decoder working on $\mathcal{G}'$ is better than that of the decoder working on $\mathcal{G}$. This will be shown by comparing the stopping set structures of $\mathcal{G}$ and $\mathcal{G}'$. Finally, we compare simulated BER and BLER performances of the MPD working on several representations of a $(23, 12)$ Golay code (GC).

A. Analysis

First, we present an example to illustrate that eliminating four-cycles helps to remove certain stopping sets and in turn improves the performance of the code under MPD. Also, this example illustrates the iterative decoding process on a GPCM representation of the code.

Example 2: Let S_1, S_2, and S_3 be the constraints of a binary code of length 7. Each codeword, represented as

$$\boldsymbol{x} = \langle x_1, x_2, x_3, x_4, x_5, x_6, x_7\rangle$$

satisfies the following set of constraints:

$$\begin{aligned} S_1: &\quad x_1 + x_2 + x_3 + x_4 = 0\\ S_2: &\quad x_1 + x_2 + x_5 + x_6 = 0\\ S_3: &\quad x_1 + x_3 + x_7 = 0. \end{aligned}$$

Let $\boldsymbol{y} = \langle \varepsilon, \varepsilon, \varepsilon, 1, 1, 1, 1\rangle$ be the channel observation of a codeword $\boldsymbol{x}$. From the received $\boldsymbol{y}$, and the set of constraints, it is easy to check that $\mathcal{S} = \{y_1, y_2, y_3\}$ is a stopping set. In other words, each check has at least two neighbors which take value ε. Hence, from the preceding discussion, we can say that the iterative decoder will fail.

We can remove the four-cycle $x_1 - S_1 - x_2 - S_2 - x_1$ in the set of constraints by introducing an auxiliary variable $x_{1,2}$. Let S_1', S_2', S_3, and S_4' be the set of modified constraints

$$\begin{aligned} S_1': &\quad x_{1,2} + x_3 + x_4 = 0\\ S_2': &\quad x_{1,2} + x_5 + x_6 = 0\\ S_3: &\quad x_1 + x_3 + x_7 = 0\\ S_4': &\quad x_{1,2} + x_1 + x_2 = 0. \end{aligned}$$

Let us try to decode $\boldsymbol{y}$ with the modified set of constraints. Since y_1 and y_2 are in erasure, the auxiliary variable takes on the value ε. It is clear that the subset of variables $\{y_1, y_2, y_3\}$ does not contain a stopping set and, hence, we are able to decode $\boldsymbol{y}$. Let $\hat{\boldsymbol{x}}$ represent the decoded word. Then

$$\begin{aligned} \hat{x}_{1,2} + 1 + 1 = 0 &\Rightarrow \hat{x}_{1,2} = 0 \text{ (from } S_2')\\ 0 + \hat{x}_3 + 1 = 0 &\Rightarrow \hat{x}_3 = 1 \text{ (from } S_1')\\ \hat{x}_1 + 1 + 1 = 0 &\Rightarrow \hat{x}_1 = 0 \text{ (from } S_3)\\ 0 + 0 + \hat{x}_2 = 0 &\Rightarrow \hat{x}_2 = 0 \text{ (from } S_4'). \end{aligned}$$

The decoded word is $\hat{\boldsymbol{x}} = \langle 0, 0, 1, 1, 1, 1, 1\rangle$. □

The example also illustrates that auxiliary variables are initialized with values based on channel observations. It will be shown that each auxiliary variable can be expressed as a sum of standard variables by manipulating the auxiliary check equations. Thus, an auxiliary variable takes the value ε if at least one of its component standard variables is in erasure. Otherwise, it takes a value 0 or 1. Also. it is clear from the example that a stopping set is a characteristic of the representation (matrix or graphical), and not necessarily that of the code. In other words, a decoding failure that occured using a particular graphical representation of a code could have been averted with a different graphical representation of the code.

Let $\mathcal{S}'$ be a stopping set in the four-cycle-free GPCM $\boldsymbol{H}'$, obtained by applying the iterative algorithm of the code. The variables (checks) corresponding to columns (rows) of $\boldsymbol{H}'$ are classified into two classes; namely, *standard variables* (*checks*) and *auxiliary variables* (*checks*). Any variable x_i corresponding to the ith column of $\boldsymbol{H}'$ is a standard variable if $i \leq n$. Otherwise, x_i is an auxiliary variable. Similarly, any check equation S_i corresponding to the ith row of $\boldsymbol{H}'$ is a standard check if $i \leq m$. Otherwise, S_i is an auxiliary check. The degree of any auxiliary check node is exactly three.

Any auxiliary variable can be written as a sum (over $\mathbb{F}_2$) of two or more standard variables. We prove this statement by describing a process to achieve such a summation. For a given auxiliary variable x_u, choose the corresponding auxiliary check equation S_{m+u-n}. Such a check exists because for every auxiliary variable introduced to remove four-cycles, a new check equation is also introduced. In general, this check equation can be written as

$$S_{m+u-n}\colon x_i + x_j = x_u$$

where $i < u$ and $j < u$. It is important to notice that the ith and the jth columns are strictly to the left of the uth column. If the components of the equation, x_i and x_j, are standard variables, then we are done. Otherwise, for every component that is an auxiliary variable, repeat the above process. This recursive process is guaranteed to achieve the goal in finite number of steps because

- for any auxiliary variable, every component variable corresponds to a column to the left of that of the auxiliary variable;
- the dimension of $\boldsymbol{H}'$ is finite.

Let us examine the stopping set structure of $\mathcal{G}'$ and compare it to that of $\mathcal{G}$.

Theorem 5: No subset of auxiliary variables is a stopping set.

Proof: Assume that $\mathcal{S}'$ is a nonempty subset of auxiliary variables which is a stopping set. This means that there will be a decoding failure if every element in $\mathcal{S}'$ takes the value ε, and every other variable

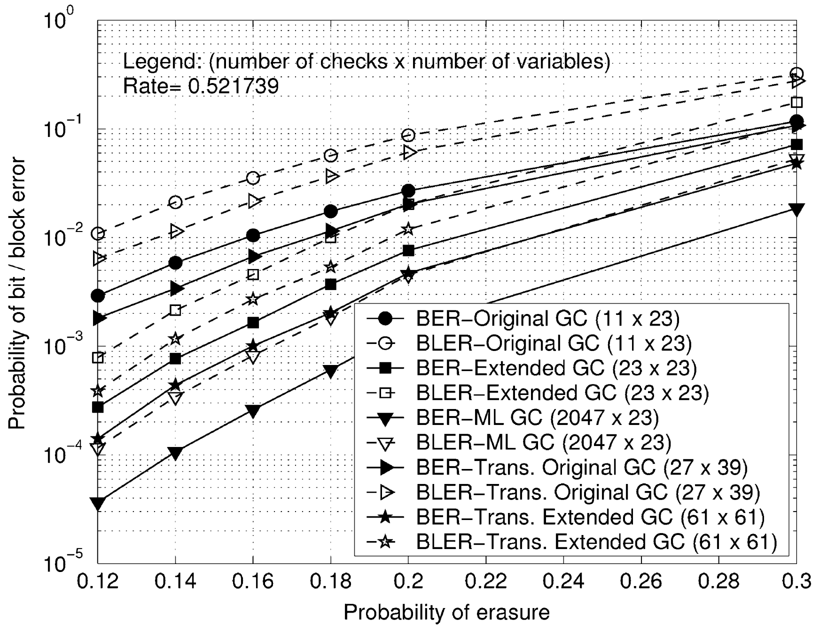

Fig. 1. Performance of different representations of $(23,12)$ GC under belief propagation for BEC.

takes a value 0 or 1. Let $\mathcal{N}(\mathcal{S}')$ be the set of all neighbors of $\mathcal{S}'$. For an auxiliary variable $x_u \in \mathcal{S}'$ there exists an auxiliary check S_{m+u-n} that belongs to $\mathcal{N}(\mathcal{S}')$. Since we are interested in the auxiliary variables, let us examine the structure in the last $\Delta(N,n)$ columns of $\boldsymbol{H}'$. The general form of these columns can be written as

$$\left(\begin{array}{ccccc} h_{1,n+1} & h_{1,n+2} & h_{1,n+3} & \dots & h_{1,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{m,n+1} & h_{m,n+2} & h_{m,n+3} & \dots & h_{m,N} \\ \hline 1 & 0 & 0 & \dots & 0 \\ h_{m+2,n+1} & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{M,n+1} & h_{M,n+2} & h_{M,n+3} & \dots & 1 \end{array}\right).$$

It is easy to see from the above submatrix that the last $M-m$ rows are in a form equivalent to the echelon form. This observation implies that any number of auxiliary variables in erasures can be decoded using these auxiliary check equations. In other words, every auxiliary variable $x_u \in \mathcal{S}'$ can be decoded by using the check equations S_{m+1} to S_M (in that order). But this contradicts the assumption that $\mathcal{S}'$ is a stopping set. Hence, the theorem is proved. □

Hence, any stopping set in $\boldsymbol{H}'$ contains at least one standard variable.

The following theorem shows that for any stopping set $\mathcal{S}'$ in $\boldsymbol{H}'$ there is a corresponding stopping set $\mathcal{S}$ in $\boldsymbol{H}$. This stopping set in $\boldsymbol{H}$ is the set of all standard variables in $\mathcal{S}'$. This result is important to show that removing four-cycles cannot hurt the performance of the iterative decoder.

Theorem 6: For any stopping set $\mathcal{S}'$ in $\boldsymbol{H}'$, define a set $\mathcal{V}$ such that

$$\mathcal{V} := \{x_v \in \mathcal{S}' \text{ and } v \leq n\}.$$

Now, $\mathcal{V}$ is a stopping set in $\boldsymbol{H}$.

Proof: Let $\mathcal{N}(\mathcal{S}')$ be the set of all neighbors of $\mathcal{S}'$ in $\mathcal{G}'$. Similarly, $\mathcal{N}(\mathcal{V})$ is the set of all neighbors of $\mathcal{V}$ in $\mathcal{G}$. For the purpose of clarity, we use $S_j{}'$ to represent the jth row of $\boldsymbol{H}'$, and S_i to represent the ith row of $\boldsymbol{H}$.

The set $\mathcal{V}$ is nonempty because $\mathcal{S}'$ should contain at least one standard variable, see Theorem 5. If $\mathcal{S}'$ does not contain any auxiliary variable, then the theorem is trivially true. To prove the nontrivial case, assume that $\mathcal{V}$ is not a stopping set. This means that there exists at least one check, say S_i, in $\mathcal{N}(\mathcal{V})$ that is connected to exactly one variable in $\mathcal{V}$. Let $x_u \in \mathcal{V}$ be the neighbor of S_i. Then, let us consider the following two possible cases.

- If x_u in S_i is not involved in any four-cycle with any other variable in S_i, then the element in position (i,u) of $\boldsymbol{H}'$ is a 1. This means that $S_i{}'$ is in $\mathcal{N}(\mathcal{S}')$ because $x_u \in \mathcal{V}$ implies $x_u \in \mathcal{S}'$ (from the definition of $\mathcal{V}$). And x_u is the only neighbor of $S_i{}'$ in $\mathcal{N}(\mathcal{S}')$. So, $\mathcal{S}'$ cannot be a stopping set. But this contradicts the assumption and, hence, $\mathcal{V}$ is a stopping set.
- If x_u in S_i is involved in at least one four-cycle, then $S_i{}'$ in $\boldsymbol{H}'$ involves an auxiliary variable whose components include x_u. This implies that $S_i{}'$ in $\mathcal{N}(\mathcal{S}')$ has exactly one neighbor in $\mathcal{S}'$. But this contradicts the assumption that $\mathcal{S}'$ is a stopping set.Hence, $\mathcal{V}$ is a stopping set.

Hence, for any stopping set $\mathcal{S}'$ in $\boldsymbol{H}'$, there is an equivalent stopping set $\mathcal{V}$ in $\boldsymbol{H}$. □

Since $\mathcal{V}$ has been shown to be a stopping set, we will refer to it as a stopping set $\mathcal{S}$ equivalent to that of $\mathcal{S}'$ in $\boldsymbol{H}'$. It is important to notice that the number of standard variables in $\mathcal{S}$ is equal to that in $\mathcal{S}'$. This means that a decoding failure due to $\mathcal{S}'$ will have the same impact on BER and BLER as that of $\mathcal{S}$. On the other hand, it is easy to show that a stopping set in $\boldsymbol{H}$ does not necessarily have an equivalent stopping set in $\boldsymbol{H}'$. In fact, the example presented in this section demonstrates this fact. Hence, it can be concluded that the performance of the iterative decoder working on $\mathcal{G}'$ is at least as good as that working on $\mathcal{G}$. In fact, for most cases, simulation results indicate that an improvement in performance is obtained by iteratively decoding on $\mathcal{G}'$.

B. Results

The $(23,12)$ GC is an algebraic code that is considered not suitable for iterative decoders. We obtained four-cycle free GPCM's of the code by iteratively applying the algorithm on the conventional matrix representations of the code. The performance curves of $(23,12)$ GC, shown in Fig. 1, indicate a decrease in BERs and BLERs when decoded using four-cycle-free GPCMs of the code. The GC can be represented by a parity-check matrix with 11 check equations, referred to as *original GC*, and by a parity-check matrix with 23 check equations, referred to as *extended GC*. The original GC has 88 ones that contribute to 190 four-cycles. The extended GC has 184 ones that contribute to

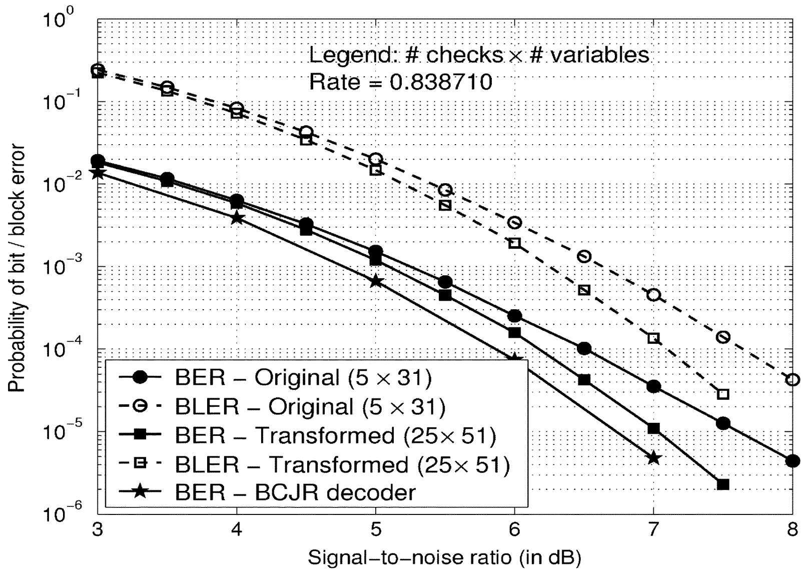

Fig. 2. Comparison of performance of $(31, 26)$ Hamming code on AWGN channel under MPD and Bahl–Cocke–Jelinek–Raviv (BCJR) decoder.

598 four-cycles. By applying the algorithm, we removed all four-cycles in the original GC and the extended GC and, hence, obtained two GPCMs, referred to as *transformed original GC* and *transformed extended GC*, respectively. The transformed original GC has 96 ones that is just eight more than that in the original GC. The transformed extended GC has 209 ones that is just 25 more than in the extended GC. The performance of the code obtained by decoding with each one of these representations is compared with that of the maximum-likelihood decoding. The maximum-likelihood decoding on BEC is performed by iteratively decoding on a bipartite graph obtained from a set of all possible constraints defining the GC. Note that the size of the transformed matrices are small compared to those presented in [13].

V. Iterative Decoding of Linear Block Codes Over AWGN Channel

An MPD based on the sum–product algorithm is a suboptimal (in general) and low-complexity decoder that works by passing soft messages (log-likelihood ratio in particular) over the edges of a graph representing the code $\mathcal{C}$. This decoding technique is optimal in a maximum *a posteriori* sense if and only if the graph of $\mathcal{C}$ is a tree (graph with no cycles) [6]. In general, good codes have several cycles in their graph representations [3]. Although the MPD working on such graphs is a suboptimal decoding technique, its performance is very good when girths of these graphs are large.

The algorithm presented in Section II helps to increase the girth of a graph from 4 to 6. Let $\mathcal{G}'$ be a four-cycle-free representation of $\mathcal{C}$. The following example illustrates that $\mathcal{G}'$ can be a more suitable representation for the iterative decoder than $\mathcal{G}$.

Example 3: Let S_1 and S_2 be the constraints of a binary code of length 6. Each codeword, represented as $\boldsymbol{x} = \langle x_1, x_2, x_3, x_4, x_5, x_6 \rangle$, satisfies the following constraints S_1 and S_2:

$$S_1 : x_1 + x_2 + x_3 + x_4 = 0$$
$$S_2 : x_1 + x_2 + x_5 + x_6 = 0.$$

Note that the graph of the code has six variable nodes and two check nodes. It is obvious that variable nodes x_1 and x_2 participate in a four-cycle. Let $\boldsymbol{y}$ be the received vector when $\boldsymbol{x}$ is sent over an AWGN channel. Based on $\boldsymbol{y}$, and the constraints, S_1 and S_2, the ith bit of the decoded vector $\hat{\boldsymbol{x}}$ is a 1 if it maximizes the *a posteriori* probability that is written as

$$\Pr(x_i = 1 \,|\, \boldsymbol{y}, S_1, S_2) = \frac{\Pr(S_1, S_2 \,|\, x_i = 1, \boldsymbol{y}) \Pr(x_i = 1 \,|\, \boldsymbol{y}) \Pr(\boldsymbol{y})}{\Pr(\boldsymbol{y}, S_1, S_2)}.$$

For such a simple case, the probability of S_1 and S_2 being satisfied given $x_1 = 1$ and $\boldsymbol{y}$ can be calculated by brute force, and it is given by

$$\Pr(S_1, S_2 \,|\, x_1 = 1, \boldsymbol{y}) = \{q_2(q_5 p_6 + p_5 q_6)(q_3 p_4 + p_3 q_4)\} + \{p_2(p_3 p_4 + q_3 q_4)(p_5 p_6 + q_5 q_6)\}$$

where $p_i = \Pr(x_i = 1 \,|\, \boldsymbol{y})$ and $q_i = 1 - p_i$.

The conventional sum–product algorithm assumes that the constraints are mutually independent, and hence, the probability of S_1 and S_2 being satisfied given $x_1 = 1$ is $\Pr(S_1 \,|\, x_1 = 1, \boldsymbol{y}) \Pr(S_2 \,|\, x_1 = 1, \boldsymbol{y})$. The above assumption is not true because x_1 and x_2 are involved in both S_1 and S_2. This means that the *a posteriori* probability of x_1 calculated by iteratively applying (a finite number of iterations) the sum–product algorithm is not accurate.

In order to remove the four-cycle in the set of constraints, we replace $x_1 + x_2$ in S_1 and S_2 with an auxiliary variable $x_{1,2}$. The modified set of constraints are

$$S_1{}' : x_{1,2} + x_3 + x_4 = 0$$
$$S_2{}' : x_{1,2} + x_5 + x_6 = 0$$
$$S_3{}' : x_{1,2} + x_1 + x_2 = 0.$$

Note that the corresponding bipartite graph is a tree. The probability of S_1 and S_2 being satisfied given $x_1 = 1$ and $\boldsymbol{y}$ can be written as

$$\Pr(S_1, S_2 | x_1 = 1, \boldsymbol{y}) = (q_2 \Pr(S_1', S_2' | x_{1,2} = 1, \boldsymbol{y})) + (p_2 \Pr(S_1', S_2' | x_{1,2} = 0, \boldsymbol{y})) \quad (2)$$

where

$$\Pr(S_1{}', S_2{}' | x_{1,2} = 1, \boldsymbol{y}) = (q_3 p_4 + p_3 q_4)(q_5 p_6 + p_5 q_6)$$
$$\Pr(S_1{}', S_2{}' | x_{1,2} = 0, \boldsymbol{y}) = (p_3 p_4 + q_3 q_4)(p_5 p_6 + q_5 q_6).$$

From (2), we see that the introduction of the auxiliary variable has facilitated the calculation of the exact *a posteriori* probability of x_1. $\square$

The sum–product algorithm is applied on $\mathcal{G}'$ after initializing the log-likelihood ratios of the auxiliary variables to zero.

Since it is difficult to theoretically analyze the effects of removing four-cycles, we present Monte Carlo simulation results that demonstrate the improvement in BER and BLER performance achieved by decoding iteratively (at most 100 iterations) on four-cycle-free representations of algebraic codes. The performance curves of $(31, 26)$ Hamming code, shown in Fig. 2, were obtained by running the message-passing algorithm on two different representations of the code: conventional 5×31 matrix representation, and transformed 25×51 GPCM representation. The conventional representation has 80 ones that contribute to 280 four-cycles. The transformed representation is

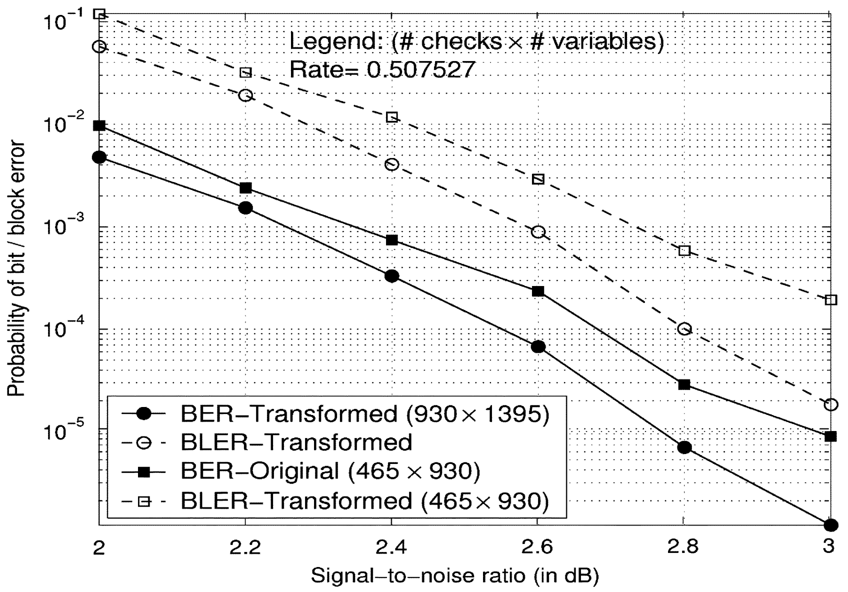

Fig. 3. Performance of different representations of $(930, 472)$ EG superposition code on AWGN channel.

a four-cycle free GPCM obtained at the cost of adding 18 more ones. The performance gain obtained from using the transformed-GPCM increases with an increase in signal-to-noise ratio (SNR). For instance, the gain at a BER of 10^{-3} is approximately 0.25 dB and it increases to approximately 0.5 dB at a BER of 10^{-5}. The BER performance of these two representations under the MPD is compared with that of a symbol-by-symbol maximum *a posteriori* decoder on the trellis of the code. At a BER of 10^{-5}, the BER performance of the transformed GPCM is 0.25 dB away from that of the optimal decoder.

The performance curves of a $(930, 472)$ LDPC code (see Fig. 3), obtained from superposition construction based on Euclidean geometry (EG) [10], were obtained by running the message-passing algorithm on two different representations of the code: a conventional 465×930 matrix representation, and a transformed 930×1395 matrix representation. Each variable in the conventional matrix is involved in exactly one four-cycle, and there are 465 four-cycles altogether. Note that the error floor in the performance of GPCM representation is an order of magnitude lower than that of the original representation.

The results thus far agree with the intuition that increasing the girth of a TG helps to improve the performance of the corresponding code. Contrary to this belief, Yedidia *et al.* had presented a special case of the $(23, 12)$ GC whose performance deteriorates significantly when iteratively decoded with transformed GPCMs [13]. The performance results that we obtained for the GC by running the message-passing algorithm on a four-cycle-free graph of the code agree with their observations. However, we would like to note that the size of our transformed matrix is at least 25 times smaller than that of their GPCMs. Also, we have observed that decoding iteratively over four-cycle-free graphs of binary representations of Reed–Solomon codes is beneficial only if they are short and have low code rates.

VI. Conclusion

In this correspondence, we have presented an algorithm to remove four-cycles from TGs of a linear block code with a minimal increase in decoding complexity. The results presented in this correspondence indicate that this method of removing four-cycles improves the suitability of iterative decoders for conventional algebraic codes. A logical extension of this work is to develop a general algorithm to remove cycles of any specified length. In fact, we have been able to use a technique, similar to the one presented here, to remove all six-cycles from small graphs. Specifically, we have observed a fractional improvement in performance by iteratively decoding on the TG obtained by removing all six-cycles and four-cycles from a $(7, 3)$ LDPC code, constructed from the point-line incidence matrix of the smallest projective plane.

Acknowledgment

The authors wish to thank Yifei Zhang and Bill Ryan for providing LDPC codes constructed using the superposition method. The authors would like to thank the anonymous referees and Khaled Abdel-Ghaffar, Associate Editor for Coding Theory, for their helpful comments.

References

[1] A. R. Calderbank, G. D. Forney, and A. Vardy, “Minimal tail-biting trellises: The Golay code and more,” *IEEE Trans. Inf. Theory*, vol. 45, no. 4, pp. 1435–1455, Jul. 1999.

[2] C. Di, D. Proietti, T. Richardson, İ. E. Telatar, and R. Urbanke, “Finite length analysis of low-density parity-check codes,” *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[3] T. Etzion, A. Trachtenberg, and A. Vardy, “Which codes have cycle-free Tanner graphs?,” *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 2173–2181, Sep. 1999.

[4] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[5] G. D. Forney, Jr., “Codes on graphs: Normal realizations,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.

[6] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[7] D. J. C. MacKay, “Relationships between sparse graph codes,” in *Proc. Information-Based Induction Sciences*, Japan, 2000. Available: [Online] at http://www.inference.phy.cam.ac.uk/mackay/abstracts/ibis. html.

[8] R. J. McEliece, “On the BCJR trellis for linear block codes,” *IEEE Trans. Inf. Theory*, vol. 42, no. 4, pp. 1072–1092, Jul. 1996.

[9] M. Tanner, “A recursive approach to low complexity codes,” *IEEE Trans. Inf. Theory*, vol. IT-27, no. 5, pp. 533–547, Sep. 1981.

[10] J. Xu and S. Lin, “A combinatoric superposition method for constructing low-density parity-check codes,” in *Proc. 2003 IEEE Int. Symp. Information Theory*, Yokohama, Japan, Jun./Jul. 2003, p. 30.

[11] N. Wiberg, H.-A. Loeliger, and R. Kötter, “Codes and iterative decoding on general graphs,” *Europ. Trans. Telecomm.*, vol. 6, pp. 513–525, Sep./Oct. 1995.

[12] N. Wiberg, “Codes and decoding on general graphs,” Ph.D. dissertation, Univ. Linköping, Linköping, Sweden, 1996.

[13] J. S. Yedidia, J. Chen, and M. C. Fossorier, “Generating code representations suitable for belief propagation decoding,” in *Proc. 40th Annu. Allerton Conf. Communications, Control, and Computing*, Monticello, IL, Oct. 2002.