The Extraction and Complexity Limits of Graphical Models for Linear Codes

Thomas R. Halford and Keith M. Chugg
Communication Sciences Institute
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2565
Email: {halford,chugg}@usc.edu

Abstract—Two broad classes of graphical modeling problems for codes can be identified in the literature: constructive and extractive problems. The former class of problems concern the construction of a graphical model in order to define a new code. The latter class of problems concern the extraction of a graphical model for a (fixed) given code. The design of a new low-density parity-check code for some given criteria (e.g. target block length and code rate) is an example of a constructive problem. The determination of a graphical model for a classical linear block code which implies a decoding algorithm with desired performance and complexity characteristics is an example of an extractive problem. This work focuses on extractive graphical model problems and aims to lay out some of the foundations of the theory of such problems for linear codes.

The primary focus of this work is a study of the space of all graphical models for a (fixed) given code. The tradeoff between cyclic topology and complexity in this space is characterized via the introduction of a new bound: the tree-inducing cut-set bound. The proposed bound provides a more precise characterization of this tradeoff than that which can be obtained using existing tools (e.g. the Cut-Set Bound) and can be viewed as a generalization of the square-root bound for tail-biting trellises to graphical models with arbitrary cyclic topologies. Searching the space of graphical models for a given code is then enabled by introducing a set of basic graphical model transformation operations which are shown to span this space. Finally, heuristics for extracting novel graphical models for linear block codes using these transformations are investigated.

I. INTRODUCTION

Graphical models of codes have been studied since the 1960s and this study has intensified in recent years due to the discovery of turbo codes by Berrou et al. [1], the rediscovery of Gallager’s low-density parity-check (LDPC) codes [2] by Spielman et al. [3] and MacKay et al. [4], and the pioneering work of Wiberg, Loeliger and Koetter [5], [6]. It is now well-known that together with a suitable message passing schedule, a graphical model implies a soft-in soft-out (SISO) decoding algorithm which is optimal for cycle-free models and suboptimal, yet often substantially less complex, for cyclic models (cf. [6], [7], [8], [9], [10]). It has been observed empirically in the literature that there exists a correlation between the cyclic topology of a graphical model and the performance of the decoding algorithms implied by that graphical model (cf. [5], [10], [11], [12], [13], [14], [15], [16]). To summarize this empirical “folk-knowledge”, those graphical models which imply near-optimal decoding algorithms tend to have large girth, a small number of short cycles and a cycle structure that is not overly regular.

Two broad classes of graphical modeling problems can be identified in the literature:

- Constructive problems: Given a set of design requirements, design a suitable code by constructing a good graphical model (i.e. a model which implies a low-complexity, near-optimal decoding algorithm).
- Extractive problems: Given a specific (fixed) code, extract a graphical model for that code which implies a decoding algorithm with desired complexity and performance characteristics.

Constructive graphical modeling problems have been widely addressed by the coding theory community. Capacity approaching LDPC codes have been designed for both the additive white Gaussian noise (AWGN) channel (cf. [17], [18]) and the binary erasure channel (cf. [19], [20]). Other classes of modern codes have been successfully designed for a wide range of practically motivated block lengths and rates (cf. [21], [22], [23], [24], [25]).

Less is understood about extractive graphical modeling problems, however. The extractive problems that have received the most attention are those concerning Tanner graph [11] and trellis representations of block codes. Tanner graphs imply low-complexity decoding algorithms; however, the Tanner graphs corresponding to many block codes of practical interest, e.g. high-rate Reed-Muller (RM), Reed-Solomon (RS), and Bose-Chaudhuri-Hocquenghem (BCH) codes, necessarily contain many short cycles [26] and thus imply poorly performing decoding algorithms. There is a well-developed theory of conventional trellises [27] and tail-biting trellises [28], [29] for linear block codes. Conventional and tail-biting trellises imply optimal and, respectively, near-optimal decoding algorithms; however, for many block codes of practical interest these decoding algorithms are prohibitively complex thus motivating the study of more general graphical models (i.e. models with a richer cyclic topology than a single cycle).

The goal of this work is to lay out some of the foundations of the theory of extractive graphical modeling problems. Following a review of graphical models for codes in Section II a complexity measure for graphical models is introduced in Section III. The proposed measure captures a cyclic graphical model analog of the familiar notions of state and branch
complexity for trellises [27]. The minimal tree complexity of a code, which is a natural generalization of the well-understood minimal trellis complexity of a code to arbitrary cycle-free models, is then defined using this measure.

The tradeoff between cyclic topology and complexity in graphical models is studied in Section IV. Wiberg’s Cut-Set Bound (CSB) is the existing tool that best characterizes this fundamental tradeoff [6]. While the CSB can be used to establish the square-root bound for tail-biting trellises [28] and thus provides a precise characterization of the potential tradeoff between cyclic topology and complexity for single-cycle models, as was first noted by Wiberg et al. [5], it is very challenging to use the CSB to characterize this tradeoff for graphical models with cyclic topologies richer than a single cycle. In order to provide a more precise characterization of this tradeoff than that offered by the CSB alone, this work introduces a new bound in Section IV: the tree-inducing cut-set bound - which may be viewed as a generalization of the square-root bound to graphical models with arbitrary cyclic topologies. Specifically, it is shown that an \( r \)-th-order complexity reduction (with respect to the minimal tree complexity as defined in Section III) requires the introduction of at least \( r(r - 1)/2 \) cycles. The proposed bound can thus be viewed as an extension of the square-root bound to graphical models with arbitrary cyclic topologies.

The transformation of graphical models is studied in Section V and VI. Whereas minimal conventional and tail-biting trellis models can be characterized algebraically via trellis-oriented generator matrices [27], there is in general no known analog of such algebraic characterizations for arbitrary cycle-free graphical models [30], let alone cyclic models. In the absence of such an algebraic characterization, it is initially unclear as to how cyclic graphical models can be extracted. In Section V a set of basic transformation operations on graphical models for codes is introduced and it is shown that any graphical model for a given code can be transformed into any other graphical model for the same code via the application of a finite number of these basic transformations. The transformations studied in Section V thus provide a mechanism for searching the space of all all graphical models for a given code. In Section VI the basic transformations introduced in Section V are used to extract novel graphical models for linear block codes. Starting with an initial Tanner graph for a given code, heuristics for extracting other Tanner graphs, generalized Tanner graphs, and more complex cyclic graphical models are investigated. Concluding remarks and directions for future work are given in Section VII.

II. BACKGROUND

A. Notation

The binomial coefficient is denoted \( \binom{n}{k} \) where \( a, b \in \mathbb{Z} \) are integers. The finite field with \( q \) elements is denoted \( \mathbb{F}_q \). Given a finite index set \( I \), the vector space over \( \mathbb{F}_q \) defined on \( I \) is the set of vectors

\[
\mathbb{F}_q^I = \{ f = (f_i \in \mathbb{F}_q, i \in I) \}.
\]

Suppose that \( J \subseteq I \) is some subset of the index set \( I \). The projection of a vector \( f \in \mathbb{F}_q^I \) onto \( J \) is denoted

\[
f|_J = (f_i, i \in J).
\]

B. Codes, Projections, and Subcodes

Given a finite index set \( I \), a linear code over \( \mathbb{F}_q \) defined on \( I \) is some vector subspace \( C \subseteq \mathbb{F}_q^I \). The block length and dimension of \( C \) are denoted \( n(C) = |I| \) and \( k(C) = \dim C \), respectively. If known, the minimum Hamming distance of \( C \) is denoted \( d(C) \) and \( C \) may be described by the triplet \( [n(C), k(C), d(C)] \). This work considers only linear codes and the terms code and linear code are used interchangeably.

A code \( C \) can be described by an \( r_G \times n(C) \), \( r_G \geq k(C) \), generator matrix \( G_C \) over \( \mathbb{F}_q \), the rows of which span \( C \). An \( r_G \times n(C) \) generator matrix is redundant if \( r_G \) is strictly greater than \( k(C) \). A code \( C \) can also be described by an \( r_H \times n(C) \), \( r_H \geq n(C) - k(C) \), parity-check matrix \( H_C \) over \( \mathbb{F}_q \), the rows of which span the null space of \( C \) (i.e. the dual code \( C^\perp \)). Each row of \( H_C \) defines a \( q \)-ary single-parity-check equation which every codeword in \( C \) must satisfy. An \( r_H \times n(C) \) parity-check matrix is redundant if \( r_H \) is strictly greater than \( k(C) \).

Given a subset \( J \subseteq I \) of the index set \( I \), the projection of \( C \) onto \( J \) is the set of all codeword projections:

\[
C|_J = \{ e_{|J} \mid e \in C \}.
\]

Closely related to \( C|_J \) is the subcode \( C_J \): the projection onto \( J \) of the set of codewords satisfying \( e_i = 0 \) for \( i \in I \setminus J \). Both \( C|_J \) and \( C_J \) are linear codes.

Suppose that \( C_1 \) and \( C_2 \) are two codes over \( \mathbb{F}_q \) defined on the same index set \( I \). The intersection \( C_1 \cap C_2 \) of \( C_1 \) and \( C_2 \) is a linear code defined on \( I \) comprising the vectors in \( \mathbb{F}_q^I \) that are contained in both \( C_1 \) and \( C_2 \).

Finally, suppose that \( C_a \) and \( C_b \) are two codes defined on the disjoint index sets \( J_a \) and \( J_b \), respectively. The Cartesian product \( C = C_a \times C_b \) is the code defined on the index set \( J = \{ J_a, J_b \} \) such that \( C|_{J_a} = C_a \) and \( C|_{J_b} = C_b \).

C. Generalized Extension Codes

Let \( C \) be a linear code over \( \mathbb{F}_q \) defined on the index set \( I \). Let \( J \subseteq I \) be some subset of \( I \) and let

\[
\beta = \{ \beta_j \neq 0 \in \mathbb{F}_q, j \in J \}.
\]

be a vector of non-zero elements of \( \mathbb{F}_q \). A generalized extension of \( C \) is formed by adding a \( q \)-ary parity-check on the codeword coordinates indexed by \( J \) to \( C \) (i.e. a \( q \)-ary partial parity symbol). The generalized extension code \( \tilde{C} \) is defined on the index set \( \tilde{I} = I \cup \{ \beta \} \) such that if \( e = (e_i, i \in I) \in \tilde{C} \) then \( \tilde{e} = (\tilde{e}_i, i \in I) \in \tilde{C} \) where \( \tilde{e}_i = e_i \) if \( i \in I \) and

\[
\tilde{e}_\beta = \sum_{j \in J} \beta_j e_j.
\]
1 for all \( j \in J \), then \( \bar{C} \) is simply a classically defined extended code [31]. More generally, a degree-\( q \) generalized extension of \( C \) is formed by adding \( g \) \( q \)-ary partial parity symbols to \( C \) and is defined on the index set \( I \cup \{ p_1, p_2, \ldots, p_g \} \). The \( j^{th} \) partial parity symbol \( c_{p_j} \) in such an extension is defined as a partial parity on some subset of \( I \cup \{ p_1, \ldots, p_{j-1} \} \).

D. Graph Theory

A graph \( G = (V, E, \mathcal{H}) \) consists of:

- A finite non-empty set of vertices \( V \).
- A set of edges \( E \), which is some subset of the pairs \( \{ \{ u, v \} : u, v \in V, \; u \neq v \} \).
- A set of half-edges \( \mathcal{H} \), which is any subset of \( V \).

It is non-standard to define graphs with half-edges; however, as will be demonstrated in Section III, half-edges are useful in the context of graphical models for codes. A walk of length \( n \) in \( G \) is a sequence of vertices \( v_1, v_2, \ldots, v_n, v_{n+1} \) in \( V \) such that \( \{ v_i, v_{i+1} \} \in E \) for all \( i \in [1, \ldots, n] \). A path is a walk on distinct vertices while a cycle of length \( n \) is a walk such that \( v_1 \) through \( v_n \) are distinct and \( v_1 = v_{n+1} \). Cycles of length \( n \) are often denoted \( n \)-cycles. A tree is a graph containing no cycles (i.e. a cycle-free graph). Two vertices \( u, v \in V \) are adjacent if a single edge \( \{ u, v \} \in E \) connects \( u \) to \( v \). A graph is connected if any two of its vertices are linked by a walk. A cut in a connected graph \( G \) is some subset of edges \( \mathcal{X} \subseteq E \) removing which yields a disconnected graph. Cuts thus partition the vertex set \( V \). Finally, a graph is bipartite if its vertex set can be partitioned \( V = U \cup W \), \( U \cap W = \emptyset \), such that any edge in \( E \) joins a vertex in \( U \) to one in \( W \).

E. Graphical Models of Codes

Graphical models for codes have been described by a number of different authors using a wide variety of notation (e.g. [6], [7], [8], [9], [10], [11]). The present work uses the notation described below which was established by Forney in his Codes on Graphs papers [10], [30].

A linear behavioral realization of a linear code \( C \subseteq F_q^t \) comprises three sets:

- A set of visible (or symbol) variables \( \{ V_i, i \in I \} \) corresponding to the codeword coordinate with alphabets \( \{ F_q, i \in I \} \).
- A set of hidden (or state) variables \( \{ S_i, i \in I_S \} \) with alphabets \( \{ F_q^T, i \in I_S \} \).
- A set of linear local constraint codes \( \{ C_i, i \in I_C \} \).

Each variable is \( q \)-ary while the hidden variable \( S_i \) with alphabet \( F_q^T \) is \( q^{T_i} \)-ary. The hidden variable alphabet index sets \( \{ T_i, i \in I_S \} \) are disjoint and unrelated to \( I \). Each local constraint code \( C_i \) involves a certain subset of the visible, \( I_V(i) \subseteq I \), and hidden, \( I_S(i) \subseteq I_S \), variables and defines a subspace of the local configuration space:

\[
C_i \subseteq \left( \prod_{j \in I_V(i)} F_q^{T_j} \right) \times \left( \prod_{j \in I_S(i)} F_q^{T_j} \right).
\]  

(7)

1Observe that this definition is slightly different than that proposed in [30] which permitted the use of \( q' \)-ary visible variables corresponding to \( r \) codeword coordinates. By appropriately introducing equality constraints and \( q \)-ary hidden variables, it can be seen that these two definitions are essentially equivalent.

Each local constraint code \( C_i \) thus has a well-defined block length

\[
n(C_i) = |I_V(i)| + \sum_{j \in I_S(i)} |T_j|
\]

(8)

dimension \( k(C_i) = \dim C_i \) over \( F_q \). Local constraints that involve only hidden variables are internal constraints while those involving visible variables are interface constraints. The full behavior of the realization is the set \( \mathcal{B} \) of all visible and hidden variable configurations which simultaneously satisfy all local constraint codes:

\[
\mathcal{B} \subseteq \left( \prod_{i \in I} F_q^{T_i} \right) \times \left( \prod_{j \in I_S} F_q^{T_j} \right) = F_q^I \times \left( \prod_{j \in I_S} F_q^{T_j} \right).
\]

(9)

The projection of the linear code \( \mathcal{B} \) onto \( I \) is precisely \( C \).

Forney demonstrated in [10] that it is sufficient to consider only those realizations in which all visible variables are involved in a single local constraint and all hidden variables are involved in two local constraints. Such normal realizations have a natural graphical representation in which local constraints are represented by vertices, visible variables by half-edges and hidden variables by edges. The half-edge corresponding to the visible variable \( V_i \) is incident on the vertex corresponding to the single local constraint which involves \( V_i \). The edge corresponding to the hidden variable \( S_j \) is incident on the vertices corresponding to the two local constraints which involve \( S_j \). The notation \( G_C \) and term graphical model is used throughout this work to denote both a normal realization of a code \( C \) and its associated graphical representation.

It is assumed throughout that the graphical models considered are connected. Equivalently, it is assumed throughout that the codes studied cannot be decomposed into Cartesian products of shorter codes [10]. Note that this restriction will apply only to the global code considered and not to the local constraints in a given graphical model.

F. Tanner Graphs and Generalized Tanner Graphs

The term Tanner graph has been used to describe different classes of graphical models by different authors. Tanner graphs denote those graphical models corresponding to parity-check matrices in this work. Specifically, let \( H_C \) be an \( r_H \times n(C) \) parity-check matrix for the code \( C \) over \( F_q \) defined on the index set \( I \). The Tanner graph corresponding to \( H_C \) contains \( r_H + n(C) \) local constraints of which \( n(C) \) are interface repetition constraints, one corresponding to each codeword coordinate, and \( r_H \) are internal \( q \)-ary single parity-check constraints, one corresponding to each row of \( H_C \). An edge (hidden variable) connects a repetition constraint \( C_i \) to a single parity-check constraint \( C_j \) if and only if the codeword coordinate corresponding to \( C_i \) is involved in the single parity-check equation defined by the row corresponding to \( C_j \). A Tanner graph for \( C \) is redundant if it corresponds to a redundant parity-check matrix. A degree-\( g \) generalized Tanner graph for \( C \) is simply a Tanner graph corresponding to some degree-\( g \) generalized extension of \( C \) in which the visible variables corresponding to the partial parity symbols have been removed. Generalized Tanner graphs have been studied previously in the literature under the rubric of generalized parity-check matrices [32], [33].
III. A Complexity Measure for Graphical Models

A. \(q^m\)-ary Graphical Models

This work introduces the term \(q^m\)-ary graphical model to denote a normal realization of a linear code \(C\) over \(\mathbb{F}_q\) that satisfies the following constraints:

- The alphabet index size of every hidden variable \(S_i, i \in I_S\), satisfies \(|T_i| \leq m\).
- Every local constraint \(C_i, i \in I_C\), either satisfies

\[
\min (k(C_i), n(C_i) - k(C_i)) \leq m \quad (10)
\]

or can be decomposed as a Cartesian product of codes, each of which satisfies this condition.

The complexity measure \(m\) simultaneously captures a cyclic graphical model analog of the familiar notions of state and branch complexity for trellises [27]. From the above definition, it is clear that Tanner graphs and generalized Tanner graphs for codes over \(\mathbb{F}_q\) are \(q\)-ary graphical models. The efficacy of this complexity measure is discussed further in Section IV.

B. Properties of \(q^m\)-ary Graphical Models

The following three properties of \(q^m\)-ary graphical models will be used in the proof of Theorem 3 in Section IV:

1) Internal Local Constraint Involvement Property: Any hidden variable in a \(q^m\)-ary graphical model can be made to be incident on an internal local constraint \(C_i\) which satisfies \(n(C_i) - k(C_i) \leq m\) without fundamentally altering the complexity or cyclic topology of that graphical model.

2) Internal Local Constraint Removal Property: The removal of an internal local constraint from a \(q^m\)-ary graphical model results in a \(q^m\)-ary graphical model for a new code defined on same index set.

3) Internal Local Constraint Redefinition Property: Any internal local constraint \(C_i\) in a \(q^m\)-ary graphical model satisfying \(n(C_i) - k(C_i) = m' \leq m\) can be equivalently represented by \(m'\) \(q\)-ary single parity-check equations over the visible variable index set.

These properties, which are defined in detail in the appendix, are particularly useful in concert. Specifically, let \(G_C\) be a \(q^m\)-ary graphical model for the linear code \(C\) over \(\mathbb{F}_q\) defined on an index set \(I\). Suppose that the internal constraint \(C_r\) satisfying \(n(C_r) - k(C_r) = m' \leq m\) is removed from \(G_C\) resulting in the new code \(C^{\varnothing}\). Denote by \(C^{(1)}_r, \ldots, C^{(m')}_r\) the set of \(m'\) \(q\)-ary single parity-check equations that result when \(C_r\) is redefined over \(I\). A vector in \(\mathbb{F}_q^I\) is a codeword in \(C\) if and only if it is contained in \(C^{\varnothing}\) and satisfies each of these \(m'\) single parity-check equations so that

\[
C = C^{\varnothing} \cap C^{(1)}_r \cap \cdots \cap C^{(m')}_r. \quad (11)
\]

C. The Minimal Tree Complexity of a Code

The minimal trellis complexity \(s(C)\) of a linear code \(C\) over \(\mathbb{F}_q\) is defined as the \(\text{base-}q\) logarithm of the maximum hidden variable alphabet size in its minimal trellis [34]. Considerable attention has been paid to this quantity (cf. [34], [35], [36], [37], [38], [39]) as it is closely related to the important, and difficult, study of determining the minimum possible complexity of optimal SISO decoding of a given code. This work introduces the minimal tree complexity of a linear code as a generalization of minimal trellis complexity to arbitrary cycle-free graphical model topologies.

**Definition 1:** The minimal tree complexity of a linear code \(C\) over \(\mathbb{F}_q\) is the smallest integer \(t(C)\) such that there exists a cycle-free \(q^{t(C)}\)-ary graphical model for \(C\).

Much as \(s(C) = s(C^\perp)\), the minimal tree complexity of a code \(C\) is equal to that of its dual.

**Proposition 1:** Let \(C\) be a linear code over \(\mathbb{F}_q\) with dual \(C^\perp\). Then

\[
t(C) = t(C^\perp). \quad (12)
\]

**Proof:** The dualizing procedure described by Forney [10] can be applied to a \(q^{t(C)}\)-ary graphical model for \(C\) in order to obtain a graphical model for \(C^\perp\) which is readily shown to be \(q^{t(C)}\)-ary.

Since a trellis is a cycle-free graphical model, \(t(C) \leq s(C)\), and all known upper bounds on \(s(C)\) extend to \(t(C)\). Specifically, consider the section of a minimal trellis for \(C\) illustrated in Figure 1. The hidden (state) variables have alphabet sizes \(|T_i| \leq s(C)\) and \(|T_{i-1}| \leq s(C), respectively. The local constraint \(C_i\) has length

\[
n(C_i) = |T_i| + |T_{i-1}| + 1 \quad (13)
\]

and dimension

\[
k(C_i) = 1 + \min (|T_i|, |T_{i-1}|) \quad (14)
\]

so that

\[
n(C_i) - k(C_i) = \max (|T_i|, |T_{i-1}|) \leq s(C). \quad (15)
\]

Lower bounds on \(s(C)\) do not, however, necessarily extend to \(t(C)\). For example, the minimal trellis complexity of a length \(2^m\), dimension \(m + 1\), binary first-order Reed Muller code is known to be \(m\) for \(m \geq 3\) [36], whereas the conditionally cycle-free generalized Tanner graphs for these codes described in [40] have a natural interpretation as \(2^{m-1}\)-ary cycle-free graphical models. The Cut-Set Bound, however, precludes \(t(C)\) from being significantly smaller than \(s(C)\) [10], [30].

The following lemma concerning minimal tree complexity will be used in the proof of Theorem 3 in Section IV.

**Lemma 1:** Let \(C\) and \(C^{\text{SPC}}\) be linear codes over \(\mathbb{F}_q\) defined on the index sets \(I\) and \(J \subseteq I\), respectively, such that \(C^{\text{SPC}}\) is a \(q\)-ary single parity-check code. Define by \(\tilde{C}\) the intersection of \(C\) and \(C^{\text{SPC}}\):

\[
\tilde{C} = C \cap C^{\text{SPC}}. \quad (16)
\]

The minimal tree complexity of \(\tilde{C}\) is upper-bounded by

\[
t(\tilde{C}) \leq t(C) + 1. \quad (17)
\]
Proof: By explicit construction of a \( q^{i(C)+1} \)-ary graphical model for \( \tilde{C} \). Let \( G_C \) be some \( q^{i(C)} \)-ary cycle-free graphical model for \( C \) and let \( T \) be a minimal connected subtree of \( G_C \) containing the set of \( |J| \) interface constraints which involve the visible variables in \( J \). Denote by \( IS(T) \subseteq IS \) and \( IC(T) \subseteq IC \) the subset of hidden variables and local constraints, respectively, contained in \( T \). Choose some local constraint vertex \( C_j, \Lambda \in IC(T) \), as a root for \( T \). Observe that the choice of \( C_j, \Lambda \), while arbitrary, induces a directionality in \( T \): downstream toward the root vertex or upstream away from the root vertex. For every \( S_i, i \in IS(T) \), denote by \( J_i \subseteq J \) the subset of visible variables in \( J \) which are upstream from that hidden variable edge.

A \( q^{i(C)+1} \)-ary graphical model for \( \tilde{C} \) is then constructed from \( G_C \) by updating each hidden variable \( S_i, i \in IS(T) \), to also contain the \( q \)-ary partial parity of the upstream visible variables in \( J_i \subseteq J \). The local constraints \( C_j, j \in IC(T) \), are updated accordingly. Finally, \( C_j \) is updated over \( G \). This updating procedure increases the alphabet size of each hidden variable \( S_i, i \in IS(T) \), by at most one and adds at most one single parity-check (or repetition) constraint to the definition of each \( C_j, j \in IC(T) \), and the resulting cycle-free graphical model is thus at most \( q^{i(C)+1} \)-ary.

The proof of Lemma 1 is detailed further by example in the appendix.

IV. THE TRADEOFF BETWEEN CYCLIC TOPOLOGY AND COMPLEXITY

A. The Cut-Set and Square-Root Bounds

Wiberg’s Cut-Set Bound (CSB) [5], [6] is stated below without proof in the language of Section II.

Theorem 1 (Cut-Set Bound): Let \( C \) be a linear code over \( \mathbb{F}_q \) defined on the index set \( I \). Let \( G_C \) be a graphical model for \( C \) containing a cut \( X \) corresponding to the hidden variables \( S_i, i \in IS(X) \), which partitions the index set into \( J_1 \subseteq I \) and \( J_2 \subseteq I \). Let the base-\( q \) logarithm of the midpoint hidden variable alphabet size of the minimal two-section trellis for \( C \) on the two-section time axis \( \{J_1, J_2\} \) be \( s_{X \text{ mid}} \). The sum of the hidden variable alphabet sizes corresponding to the cut \( X \) is lower-bounded by

\[
\sum_{i \in IS(X)} |T_i| \geq s_{X \text{ mid}}. \tag{18}
\]

The CSB provides insight into the tradeoff between cyclic topology and complexity in graphical models for codes and it is natural to explore its power to quantify this tradeoff. Two questions which arise for a given linear code \( C \) over \( \mathbb{F}_q \) in such an exploration are:

1) For a given complexity \( m \), how many cycles must be contained in a \( q^m \)-ary graphical model for \( C \)?

2) For a given number of cycles \( N \), what is the smallest \( m \) such that a \( q^m \)-ary model containing \( N \) cycles for \( C \) can exist?

For a fixed cyclic topology, the CSB can be simultaneously applied to all cuts yielding a linear programming lower bound on the hidden variable alphabet sizes [5]. For the special case of a single-cycle graphical model (i.e. a tail-biting trellis), this technique yields a simple solution [28]:

Theorem 2 (Square-Root Bound): Let \( C \) be a linear code over \( \mathbb{F}_q \) of even length and let \( s_{\text{mid}, \text{min}}(C) \) be the base-\( q \) logarithm of the minimum possible hidden variable alphabet size of a conventional trellis for \( C \) at its midpoint over all coordinate orderings. The base-\( q \) logarithm of the minimum possible hidden variable alphabet size \( s_{TB}(C) \) of a tail-biting trellis for \( C \) is lower-bounded by

\[
s_{TB}(C) \geq s_{\text{mid}, \text{min}}(C)/2. \tag{19}
\]

The square-root bound can thus be used to answer the questions posed above for a specific class of single-cycle graphical models. For topologies richer than a single cycle, however, the aforementioned linear programming technique quickly becomes intractable. Specifically, there are

\[
2^{n(C)} - 1 \tag{20}
\]

ways to partition a size \( n(C) \) visible variable index set into two non-empty, disjoint, subsets. The number of cuts to be considered by the linear programming technique for a given cyclic topology thus grows exponentially with block length and a different minimal two-stage trellis must be constructed in order to bound the size of each of those cuts.

B. Tree-Inducing Cuts

Recall that a cut in a graph \( G \) is some subset of the edges \( X \subseteq E \) the removal of which yields a disconnected graph. A cut is thus defined without regard to the cyclic topology of the disconnected components which remain after its removal. In order to provide a characterization of the tradeoff between cyclic topology and complexity which is more precise than that provided by the CSB alone, this work focuses on a specific type of cut which is defined below. Two useful properties of such cuts are established by Propositions 2 and 3.

Definition 2: Let \( G \) be a connected graph. A tree-inducing cut is some subset of edges \( X_T \subseteq E \) the removal of which yields a tree with precisely two components.

Proposition 2: Let \( G = (V, E, H) \) be a connected graph. The size \( X_T \) of any tree-inducing cut \( X_T \) in \( G \) is precisely

\[
X_T = |E| - |V| + 2. \tag{21}
\]

Proof: It is well-known that a connected graph is a tree if and only if (cf. [41])

\[
|E| = |V| - 1. \tag{22}
\]

Similarly, a graph composed of two cycle-free components satisfies

\[
|E| = |V| - 2. \tag{23}
\]

The result then follows from the observation that the size of a tree-inducing cut is the number of edges which must be removed in order to satisfy (23). □

Proposition 3: Let \( G \) be a connected graph with tree-inducing cut size \( X_T \). The number of cycles \( N_G \) in \( G \) is lower-bounded by

\[
N_G \geq \left( \frac{X_T}{2} \right). \tag{24}
\]
Proof: Let the removal of a tree-inducing cut \( \mathcal{C}_T \) in the connected graph \( \mathcal{G} \) yield the cycle-free components \( \mathcal{G}_1 \) and \( \mathcal{G}_2 \) and let \( e_i \neq e_j \in \mathcal{C}_T \). Since \( \mathcal{G}_1 \) (\( \mathcal{G}_2 \)) is a tree, there is a unique path in \( \mathcal{G}_1 \) (\( \mathcal{G}_2 \)) connecting \( e_i \) and \( e_j \). There is thus a unique cycle in \( \mathcal{G} \) corresponding to the edge pair \( \{e_i, e_j\} \). There are \( \binom{x}{2} \) such distinct edge pairs which yields the lower bound. Note that this is a lower bound because for certain graphs, there can exist cycles which contain more than two edges from a tree-inducing cut. 

\[ \square \]

C. The Tree-Inducing Cut-Set Bound

With tree-inducing cuts defined, the required properties of \( q^m \)-ary graphical models described and Lemma 1 established, the main result concerning the tradeoff between cyclic topology and graphical model complexity can now be stated and proved.

**Theorem 3:** Let \( \mathcal{C} \) be a linear code over \( \mathbb{F}_q \) defined on the index set \( I \) and suppose that \( \mathcal{G}_C \) is a \( q^m \)-ary graphical model for \( \mathcal{C} \) with tree-inducing cut size \( X_T \). The minimal tree complexity of \( \mathcal{C} \) is upper-bounded by 

\[
t(\mathcal{C}) \leq mX_T.
\]

**Proof:** By induction on \( X_T \). Let \( X_T = 1 \) and suppose that \( e \in \mathcal{X}_T \) is the sole edge in some tree-inducing cut \( \mathcal{X}_T \) in \( \mathcal{G}_C \). Since the removal of \( e \) partitions \( \mathcal{G}_C \) into disconnected cycle-free components, \( \mathcal{G}_C \) must be cycle-free and \( t(\mathcal{C}) \leq m \) by construction.

Now suppose that \( X_T = x > 1 \) and let \( e \in \mathcal{X}_T \) be an edge in some tree-inducing cut \( \mathcal{X}_T \) in \( \mathcal{G}_C \). By the first \( q^m \)-ary graphical model property of Section II-B, \( e \) is incident on some internal local constraint \( \mathcal{C}_i \) satisfying \( n(\mathcal{C}_i) - k(\mathcal{C}_i) = m' \leq m \). Denote by \( \mathcal{G}_{C,i} \) the \( q^m \)-ary graphical model that results when \( \mathcal{C}_i \) is removed from \( \mathcal{G}_C \), and by \( \mathcal{C}_{i}^{(t)} \) the corresponding code over \( I \). The tree-inducing cut size of \( \mathcal{G}_{C,i} \) is at most \( x - 1 \) since the removal of \( \mathcal{C}_i \) from \( \mathcal{G}_C \) results in the removal of a single vertex and at least two edges. By the induction hypothesis, the minimal tree complexity of \( \mathcal{C}_{i}^{(t)} \) is upper-bounded by

\[
t(\mathcal{C}_{i}^{(t)}) \leq m(x - 1).
\]

From the discussion of Section II-B, it is clear that \( \mathcal{C}_i \) can be redefined as \( m' \leq m \) single parity check equations, \( \mathcal{C}_i^{(j)} \) for \( j \in [1, m'] \), over \( \mathbb{F}_q \) on \( I \) such that

\[
\mathcal{C} = \mathcal{C}_{i}^{(t)} \cap \mathcal{C}_i^{(1)} \cap \cdots \cap \mathcal{C}_i^{(m')}. 
\]

It follows from Lemma 1 that

\[
t(\mathcal{C}) \leq t(\mathcal{C}_{i}^{(t)}) + m' \leq mx
\]

completing the proof.

An immediate corollary to Theorem 3 results when Proposition 3 is applied in conjunction with the main result:

**Corollary 1:** Let \( \mathcal{C} \) be a linear code over \( \mathbb{F}_q \) with minimal tree complexity \( t(\mathcal{C}) \). The number of cycles \( N_m \) in any \( q^m \)-ary graphical model for \( \mathcal{C} \) is lower-bounded by

\[
N_m \geq \left( \frac{t(\mathcal{C})/m}{2} \right).
\]

D. Interpretation of the TI-CSB

Provided \( t(\mathcal{C}) \) is known or can be lower-bounded, the tree-inducing cut-set bound (TI-CSB) (and more specifically Corollary 1) can be used to answer the questions posed in Section IV-A. The TI-CSB is further discussed below.

1) The TI-CSB and the CSB: On the surface, the TI-CSB and the CSB are similar in statement; however, there are three important differences between the two. First, the CSB does not explicitly address the complexity of the local constraints on either side of a given cut. Forney provided a number of illustrative examples in [30] that stress the importance of characterizing graphical model complexity in terms of both hidden variable size and local constraint complexity. Second, the CSB does not explicitly address the cyclic topology of the graphical model that results when the edges in a cut are removed. The removal of a tree-inducing cut results in two cycle-free disconnected components and the size of a tree-inducing cut can thus be used to make statements about the complexity of optimal SISO decoding using variable conditioning in a cyclic graphical model (cf. [10], [40], [42], [43], [44], [45]). Finally, and most fundamentally, the TI-CSB addresses the aforementioned intractability of applying the CSB to graphical models with rich cyclic topologies.

2) The TI-CSB and the Square-Root Bound: Theorem 3 can be used to make a statement similar to Theorem 2 which is valid for all graphical models containing a single cycle.

**Corollary 2:** Let \( \mathcal{C} \) be a linear code over \( \mathbb{F}_q \) with minimal tree complexity \( t(\mathcal{C}) \) and let \( m_1 \) be the smallest integer such that there exists a \( q^{m_1} \)-ary graphical model for \( \mathcal{C} \) which contains at most one cycle. Then

\[
m_1 \geq t(\mathcal{C})/2.
\]

More generally, Theorem 3 can be used to establish the following generalization of the square-root bound to graphical models with arbitrary cyclic topologies.

**Corollary 3:** Let \( \mathcal{C} \) be a linear code over \( \mathbb{F}_q \) with minimal tree complexity \( t(\mathcal{C}) \) and let \( m_{(q)} \) be the smallest integer such that there exists a \( q^{m_{(q)}} \)-ary graphical model for \( \mathcal{C} \) which contains at most \( q \) cycles. Then

\[
m_{(q)} \geq t(\mathcal{C})/r.
\]

A linear interpretation of the logarithmic complexity statement of Corollary 3 yields the desired generalization of the square-root bound: an \( r^{th} \)-root complexity reduction with respect to the minimal tree complexity requires the introduction of at least \( r(r - 1)/2 \) cycles.

There are few known examples of classical linear block codes which meet the square-root bound with equality. Shany and Be’ery proved that many RM codes cannot meet this bound under any bit ordering [46]. There does, however, exist a tail-biting trellis for the extended binary Golay code \( \mathcal{C}_G \) which meets the square-root bound with equality so that \( t(\mathcal{C}_G) \)

\[
s_{\text{mid-min}}(\mathcal{C}_G) = 8 \quad \text{and} \quad s_{\text{TB}}(\mathcal{C}_G) = 4.
\]
the extended binary Golay code can be upper-bounded by Corollary 2
as
\[ t(C_G) \leq 8. \]  
(33)

Note that the minimal bit-level conventional trellis for \( C_G \) contains (non-central) state variables with alphabet size 512
and is thus a \( 2^9 \)-ary graphical model [35]. The proof of Lemma 1 provides a recipe for the construction of a \( 2^9 \)-ary
cycle-free graphical model for \( C_G \) from its tail-biting trellis. It remains open as to where the minimal tree complexity of \( C_G \)
is precisely 8, however.

3) Asymptotics of the TI-CSB: Denote by \( N_m \) the minimum number of cycles in any \( q^m \)-ary graphical model for a linear
code \( C \) over \( \mathbb{F}_q \) with minimal tree complexity \( t(C) \). For large values of \( t(C)/m \), the lower bound on \( N_m \) established by
Corollary 1 becomes
\[ N_m \geq \left( \frac{t(C)/m}{2} \right)^2 \approx \frac{t(C)^2}{2m^2}. \]  
(34)
The ratio of the minimal complexity of a cycle-free model for \( C \) to that of an \( q^m \)-ary graphical model is thus upper-bounded by
\[ \frac{t(C)}{q^m} \leq q^{2m\sqrt{\log m}}. \]  
(35)

In order to further explore the asymptotics of the tree-inducing cut-set bound, consider a code of particular practical
interest: the binary image \( C_{RS} \) of the [255, 223, 33] Reed-Solomon code \( C_{RS} \). Since \( C_{RS} \) is maximum distance separable, a reasonable estimate for the minimal tree complexity of this code is obtained from Wolf’s bound [47]
\[ t(C_{RS}) \approx 8(n(C_{RS}) - k(C_{RS})) = 256. \]  
(36)
Figure 2 plots \( N_m \) as a function of \( m \) for \( C_{RS} \) assuming (36). Note that since the complexity of the decoding algorithms implied by \( 2^m \)-ary graphical models grow roughly as \( 2^{m^\log m} \),
\( \log m \) is roughly a \( \log \log \) decoding complexity measure.

4) On Complexity Measures: Much as there are many valid complexity measures for conventional trellises, there are many
reasonable metrics for the measurement of cyclic graphical model complexity. While there exists a unique minimal trellis
for any linear block code which simultaneously minimizes all reasonable measures of complexity [48], even for the class
cyclic graphical models with the most basic cyclic topology - tail-biting trellises - minimal models are not unique [29].
The complexity measure introduced by this work was motivated by the desire to have a metric which simultaneously captures
hidden variable complexity and local constraint complexity thus disallowing local constraints from “hiding” complexity.
There are many conceivable measures of local constraint complexity: one could upper-bound the state complexity of the
local constraints or even their minimal tree complexity (thus defining minimal tree complexity recursively). The local
constraint complexity measure used in this work is essentially Wolf’s bound [47] and is thus a potentially conservative upper bound on any reasonable measure of local constraint decoding complexity.

V. GRAPHICAL MODEL TRANSFORMATION

Let \( G_C \) be a graphical model for the linear code \( C \) over \( \mathbb{F}_q \).
This work introduces eight basic graphical model operations
the application of which to \( G_C \) results in a new graphical model for \( C \):
- The merging of two local constraints \( C_{i_1} \) and \( C_{i_2} \) into the
new local constraint \( C_i \) which satisfies
\[ C_i = C_{i_1} \cap C_{i_2}. \]  
(37)
- The splitting of a local constraint \( C_j \) into two new local
constraints \( C_{j_1} \) and \( C_{j_2} \) which satisfy
\[ C_{j_1} \cap C_{j_2} = C_j. \]  
(38)
- The insertion/removal of a degree-2 repetition constraint.
- The insertion/removal of a trival length 0, dimension 0
local constraint.
- The insertion/removal of an isolated partial parity-check
constraint.

Note that some of these operations have been introduced implicitly in this work and others already. For example, the
proof of the local constraint involvement property of \( q^m \)-ary
graphical models presented in Section III-B utilizes degree-2
repetition constraint insertion. Local constraint merging has been considered by a number of authors under the rubric of clustering (e.g. [9], [10]). This work introduces the term merging specifically so that it can be contrasted with its inverse operation: splitting. Detailed definitions of each of the eight basic graphical model operations are given in the appendix. In
this section, it is shown that these basic operations span the
entire space of graphical models for \( C \).

Theorem 4: Let \( G_C \) and \( \tilde{G}_C \) be two graphical models for
the linear code \( C \) over \( \mathbb{F}_q \). Then \( G_C \) can be transformed into \( \tilde{G}_C \)
via the application of a finite number of basic graphical model operations.

Proof: Define the following four sub-transformations which can be used to transform \( G_C \) into a Tanner graph \( G_C^T \):

![Fig. 2. Minimum number of cycles required for \( 2^m \)-ary graphical models of the binary image of the [255, 223, 33] Reed-Solomon code.](image-url)
1) The transformation of $G_C$ into a $q$-ary model $G_C^q$.
2) The transformation of $G_C^q$ into a (possibly redundant) generalized Tanner graph $G_C^q$.
3) The transformation of $G_C^q$ into a non-redundant generalized Tanner graph $G_C^q$.
4) The transformation of $G_C^q$ into a Tanner graph $G_C^T$.

Since each basic graphical model operation has an inverse, $G_C^T$ can be transformed into $G_C$ by inverting each of the four sub-transformations. In order to prove that $G_C$ can be transformed into $G_C$ via the application of a finite number of basic graphical model operations, it suffices to show that each of the four sub-transformations requires a finite number of operations and that the transformation of the Tanner graph $G_C^T$ into a Tanner graph $G_C^T$ corresponding to $G_C$ requires a finite number of operations. This proof summary is illustrated in Figure 3.

![Fig. 3. The transformation of $G_C$ into $G_C^T$ via five sub-transformations.](image)

That each of the five sub-transformations from $G_C$ to $G_C^T$ illustrated in Figure 3 requires only a finite number of basic graphical model operations is proved below.

1) $G_C \rightarrow G_C^q$: The graphical model $G_C$ is transformed to the $q$-ary model $G_C^q$ as follows. Each local constraint $C_i$ in $G_C$ is split into the $n(C_i) - k(C_i)$ $q$-ary single parity-check constraints which define it. A degree-$2$ repetition constraint is then inserted into every hidden variable with alphabet index set size $m > 1$ and these repetition constraints are then each split into $m$ $q$-ary repetition constraints as illustrated in Figure 4. Each local constraint $C_j$ in the resulting graphical model satisfies $n(C_j) - k(C_j) = 1$. Similarly, each hidden variable $S_j$ in the resulting graphical model satisfies $|T_j| = 1$.

![Fig. 4. Transformation of the $q^m$-ary hidden variable $S_j$ into $q$-ary hidden variables.](image)

2) $G_C^q \rightarrow G_C^q$: A (possibly redundant) generalized Tanner graph is simply a bipartite $q$-ary graphical model with one vertex class corresponding to repetition constraints and one to single parity-check constraints in which visible variables are incident only on repetition constraints. By appropriately inserting degree-$2$ repetition constraints, the $q$-ary model $G_C^q$ can be transformed into $G_C^q$.

3) $G_C^q \rightarrow G_C^q$: Let the generalized Tanner graph $G_C^q$ correspond to an $r_H \times n(C) + g$ redundant parity-check matrix $H_C^{(r,g)}$ for a degree-$g$ generalized extension of $C$ with rank

$$\text{rank}(H_C^{(r,g)}) = n(C) - k(C) + g. \quad (39)$$

A finite number of row operations can be applied to $H_C^{(r,g)}$ resulting in a new parity-check matrix the last $r_H - \text{rank}(H_C^{(r,g)})$ rows of which are all zero. Similarly, a finite number of basic row operations can be applied to $G_C^q$ resulting in a generalized Tanner graph containing $r_H - \text{rank}(H_C^{(r,g)})$ trivial constraints which can then be removed to yield $G_C^q$. Specifically, consider the row operation on $H_C^{(r,g)}$ which replaces a row $h_i$ by

$$h_i = h_i + \beta_j h_j \quad (40)$$

where $\beta_j \in \mathbb{F}_q$. The graphical model transformation corresponding to this row operation first merges the $q$-ary single parity-check constraints $C_i$ and $C_j$ (which correspond to rows $h_i$ and $h_j$, respectively) and then splits the resulting check into the constraints $C_i$ and $C_j$ (which correspond to rows $h_i$ and $h_j$, respectively). Note that this procedure is valid since

$$C_i \cap C_j = \hat{C_i} \cap \hat{C_j}. \quad (41)$$

4) $G_C^q \rightarrow G_C^T$: Let the degree-$g$ generalized Tanner graph $G_C^q$ correspond to an $(n(C) - k(C) + g + n(C) + g)$ parity-check matrix $H_C^{(g)}$. A degree-$(g-1)$ generalized Tanner graph $G_C^{q-1}$ is obtained from $G_C^q$ as follows. Denote by $H_C^{(g)}$ the parity-check matrix for the degree-$g$ generalized extension defined by $H_C^{(g)}$ which is systematic in the position corresponding to the $g$-th partial parity symbol. Since a finite number of row operations can be applied to $H_C^{(g)}$ to yield $H_C^{(g)}$, a finite number of local constraint merge and split operations can be applied to $G_C^q$ to yield the corresponding generalized Tanner graph $G_C^q$. Removing the now isolated partial-parity check constraint corresponding to the $g$-th partial parity symbol in $G_C^q$ yields the desired degree-$(g-1)$ generalized Tanner graph $G_C^{q-1}$. By repeatedly applying this procedure, all partial parity symbols can be removed from $G_C^q$ resulting in $G_C^T$.

5) $G_C^T \rightarrow G_C^T$: Let the Tanner graphs $G_C^T$ and $G_C^T$ correspond to the parity-check matrices $H_C$ and $H_C$, respectively. Since $H_C$ can be transformed into $H_C$ via a finite number of row operations, $G_C^T$ can be similarly transformed into $G_C^T$ via the application of a finite number of local constraint merge and split operations.

VI. GRAPHICAL MODEL EXTRACTION VIA TRANSFORMATION

The set of basic model operations introduced in the previous section enables the space of all graphical models for a given code $C$ to be searched, thus allowing for model extraction to be expressed as an optimization problem. The challenges of defining extraction as optimization are twofold. First, a cost measure on the space of graphical models must be found which is simultaneously meaningful in some real sense (e.g. highly correlated with decoding performance) and computationally tractable. Second, given that discrete optimization problems are in general very hard, heuristics for extraction must be found. In this section, heuristics are investigated for the extraction of graphical models for binary linear block codes from an initial Tanner graph. The cost measures considered are functions of the short cycle structure of graphical models. The use of such cost measures is motivated first by empirical
Following the notation introduced in Section II-F, suppose that a series of local constraint merge and split operations of graphical model transformation. The insertion of partial parity symbols, an operation which is empirically to adversely affect decoding performance [52]. Partial parity symbols in a generalized Tanner graph and the cause no information is available to the decoder about the with a particular focus on models which are improve the short cycle structure of greedy search continues until there are no more row operations largest short cycle reduction in. Theorem 4, if cycles starting with an initial Tanner graph algorithm for cyclic codes [49] and the random redundant decoding algorithms which utilize Tanner graphs such as Jiang and Narayanan’s stochastic shifting based iterative decoding for counting short cycles in bipartite graphs [16]. Simulation evidence concerning the detrimental effect of short cycles on decoding performance (cf. [6], [10], [11], [12], [13], [14], [15], [16]) and second by the existence of an efficient algorithm for counting short cycles in bipartite graphs [16].

A. A Greedy Heuristic for Tanner Graph Extraction

The Tanner graphs corresponding to many linear block codes of practical interest necessarily contain many short cycles [26]. Suppose that any Tanner graph for a given code \( C \) must have girth at least \( g_{\text{min}}(C) \); an interesting problem is the extraction of a Tanner graph for \( C \) containing the smallest number of \( g_{\text{min}}(C) \)-cycles. The extraction of such Tanner graphs is especially useful in the context of ad-hoc decoding algorithms which utilize Tanner graphs such as Jiang and Narayanan’s stochastic shifting based iterative decoding for cyclic codes [49] and the random redundant iterative decoding algorithm presented in [50].

Algorithm 1 performs a greedy search for a Tanner graph for \( C \) with girth \( g_{\text{min}}(C) \) and the smallest number of \( g_{\text{min}}(C) \)-cycles starting with an initial Tanner graph \( \text{TG}(H_C) \) which corresponds to some binary parity-check matrix \( H_C \). Define an \((i, j)\)-row operation as the replacement of row \( h_j \) in \( H_C \) by the binary sum of rows \( h_i \) and \( h_j \). As detailed in the proof of Theorem 2 if \( C_i \) and \( C_j \) are the single parity-check constraints in \( \text{TG}(H_C) \) corresponding to \( h_i \) and \( h_j \), respectively, then an \((i, j)\)-row operation in \( H_C \) is equivalent to merging \( C_i \) and \( C_j \) to form a new constraint \( C_{i,j} = C_i \cap C_j \) and then splitting \( C_{i,j} \) into \( C_i \) and \( C_j \) (where \( C_j \) enforces the binary sum of rows \( h_i \) and \( h_j \)). Algorithm 1 iteratively finds the rows \( h_i \) and \( h_j \) in \( H_C \) with corresponding \((i, j)\)-row operation that results in the largest short cycle reduction in \( \text{TG}(H_C) \) at every step. This greedy search continues until there are no more row operations that improve the short cycle structure of \( \text{TG}(H_C) \).

B. A Greedy Heuristic for Generalized Tanner Graph Extraction

A number of authors have studied the extraction of generalized Tanner graphs (GTGs) of codes for which \( g_{\text{min}}(C) = 4 \) with a particular focus on models which are 4-cycle-free and which correspond to generalized code extensions of minimal degree [51], [52]. Minimal degree extensions are sought because no information is available to the decoder about the partial parity symbols in a generalized Tanner graph and the introduction of too many such symbols has been observed empirically to adversely affect decoding performance [52].

Generalized Tanner graph extraction algorithms proceed via the insertion of partial parity symbols, an operation which is most readily described as a parity-check matrix manipulation. Following the notation introduced in Section II-F, suppose that a partial parity on the coordinates indexed by

\[ J \subseteq I \cup \{p_1, p_2, \ldots, p_g\} \]  

Note that partial parity insertion can also be viewed through the lens of graphical model transformation. The insertion of partial parity symbol proceeds via the insertion of an isolated partial parity check followed by a series of local constraint merge and split operations.

Algorithm 1: Greedy heuristic for the reduction of short cycles in Tanner graphs for binary codes.

\[
\begin{align*}
\text{Input:} & \quad r_H \times n(C) \text{ binary parity-check matrix } H_C. \\
\text{Output:} & \quad r_H \times n(C) \text{ binary parity-check matrix } H'_C. \\
\text{H}'_C & \leftarrow H_C; \quad i^* \leftarrow -1; \quad j^* \leftarrow -1; \quad g^* \leftarrow \text{girth of } \text{TG}(H'_C); \\
N_g^* & \leftarrow \text{number of } g^*\text{-cycles in } \text{TG}(H'_C); \\
N_g^* + 2 & \leftarrow \text{number of } g^* + 2\text{-cycles in } \text{TG}(H'_C); \\
\text{repeat} & \\
\text{if } i^* \neq j^* \text{ then} & \text{ Replace row } h_{j^*} \text{ in } H'_C \text{ with binary sum of rows } h_i \text{ and } h_{j^*}; \\
& i^* \leftarrow -1; \quad j^* \leftarrow -1; \\
& \text{for } i, j = 0, \ldots, r_H - 1, i \neq j \text{ do} \\
& \text{Replace row } h_j \text{ in } H'_C \text{ with binary sum of rows } h_i \text{ and } h_j; \\
& g \leftarrow \text{girth of } \text{TG}(H'_C); \\
& N_g \leftarrow \text{number of } g\text{-cycles in } \text{TG}(H'_C); \\
& N_{g+2} \leftarrow \text{number of } g + 2\text{-cycles in } \text{TG}(H'_C); \\
& \text{if } g > g^* \text{ then} \\
& \quad g^* \leftarrow g; \quad i^* \leftarrow i; \quad j^* \leftarrow j; \quad N_g^* \leftarrow N_g; \\
& \quad N_{g^* + 2} \leftarrow N_{g^* + 2}; \\
& \text{else if } g = g^* \text{ AND } N_g < N_g^* \text{ then } i^* \leftarrow i; \\
& \quad j^* \leftarrow j; \quad N_g^* \leftarrow N_g; \quad N_{g^* + 2} \leftarrow N_{g^* + 2}; \\
& \text{else if } g = g^* \text{ AND } N_g = N_g^* \text{ then} \\
& \quad \text{if } N_{g^* + 2} < N_{g^* + 2} \text{ then } i^* \leftarrow i; \quad j^* \leftarrow j; \\
& \quad N_{g^* + 2} \leftarrow N_{g^* + 2}; \\
& \text{end} \\
& \text{Undo row replacement}; \\
& \text{end} \\
& \text{end} \\
& \text{until } i^* = -1 \text{ & } j^* = -1; \\
& \text{return } H'_C.
\end{align*}
\]

is to be introduced to a GTG for \( C \) corresponding to a degree- \( g \) generalized extension \( \tilde{C} \) with parity-check matrix \( H_{\tilde{C}} \). A row \( h_p \) is first appended to \( H_{\tilde{C}} \) with a 1 in the positions corresponding to coordinates indexed by \( J \) and a 0 in the other positions. A column is then appended to \( H_{\tilde{C}} \) with a 1 only in the position corresponding to \( h_p \). The resulting parity-check matrix \( H_{\tilde{C}} \) describes a degree- \( g + 1 \) generalized extension \( \tilde{C} \). Every row \( h_i \neq h_p \) in \( H_{\tilde{C}} \) which contains a 1 in all of the positions corresponding to coordinates indexed by \( J \) is then replaced by the binary sum of \( h_i \) and \( h_p \). Suppose that there are \( r(J) \) such rows. It is readily verified that the tree-inducing cut size \( \tilde{X}_T \) of the GTG that results from this insertion is related to that of the initial GTG, \( X_T \), by

\[ \Delta X_T = \tilde{X}_T - X_T = (|J| - 1)(r(J) - 1). \]  

Algorithm 2 performs a greedy search for a 4-cycle-free generalized Tanner graph for \( C \) with the smallest number of inserted partial parity symbols starting with an initial Tanner graph \( \text{TG}(H_C) \) which corresponds to some binary parity-check matrix \( H_C \). Algorithm 3 iteratively finds the symbol subsets that result in the largest tree-inducing cut size reduction and then introduces the partial parity symbol corresponding to one of those subsets. At each step, Algorithm 3 uses Algorithm 2 to generate a candidate list of partial parity symbols to insert.
and chooses from that list the symbol which reduces the most short cycles when inserted. This greedy procedure continues until the generalized Tanner graph contains no 4-cycles.

Algorithm 3 is closely related to the GTG extraction heuristics proposed by Sankaranarayanan and Vasić in [51] and Kumar and Milenkovic in [52] (henceforth referred to as the SV and KM heuristics, respectively). It is readily shown that Algorithm 3 is guaranteed to terminate using the proof technique of [51]. The SV heuristic considers only the insertion of partial parity symbols corresponding to coordinate index sets of size 2 (i.e. \(|J| = 2\)). The KM heuristic considers only the insertion of partial parity symbols corresponding to coordinate index sets satisfying \(r(J) = 2\). Algorithm 2, however, considers all coordinate index sets satisfying \(|J| = 2, 3, 4\) and \(r(J) = 2, 3, 4\) and then uses \(\Delta T\) to evaluate which of these coordinate sets results in the largest tree-inducing cut size reduction. Algorithm 3 is thus able to extract GTGs corresponding to generalized extensions of smaller degree than the SV and KM heuristics. In order to illustrate this observation, the degrees of the generalized code extensions that result when the SV, KM and proposed (HC) heuristics are applied to parity-check matrices for three codes are provided in Table 1. Figure 2 compares the performance of the three extracted GTG decoding algorithms for the \([31, 21, 5]\) BCH code in order to illustrate the efficacy of extracting GTGs corresponding to extensions of smallest possible degree.

| Code    | SV | KM | HC |
|---------|----|----|----|
| \([23, 12, 7]\) Golay | 18 | 11 | 10 |
| \([31, 21, 5]\) BCH | 47 | 19 | 12 |
| \([63, 30, 13]\) BCH | 264 | 121 | 69 |

Table 1

Generalized code extension degrees corresponding to the 4-cycle-free GTGs obtained via the SV, KM, and HC heuristics.

![Figure 2](image_url)

Fig. 2. Bit error rate performance of three GTG decoding algorithms for the \([31, 21, 5]\) BCH code. One-hundred iterations of a flooding schedule were performed. Binary antipodal signaling over an AWGN channel is assumed.

**Input:** Binary generalized parity-check matrix \(H_{\tilde{C}}\).

**Output:** List \(S\) of best partial parity symbols sets.

\(S \leftarrow \emptyset\); \(\Delta X^*_T \leftarrow 0\);

// Consider pairs of columns of \(H_{\tilde{C}}\).

\(J_2 \leftarrow \) coordinate pair that maximizes \(r(J_2)\);

Append \(J_2\) to \(S\); \(\Delta X^*_T \leftarrow r(J_2) - 1\);

// Consider 3-tuples of columns of \(H_{\tilde{C}}\).

\(J_3 \leftarrow \) coordinate 3-tuple that maximizes \(r(J_3)\);

if \(2(r(J_3) - 1) > \Delta X^*_T\) then

\(S \leftarrow \emptyset\); Append \(J_3\) to \(S\); \(\Delta X^*_T \leftarrow 2(r(J_3) - 1)\);

end

else if \(2(r(J_3) - 1) = \Delta X^*_T\) then Append \(J_3\) to \(S\);

// Consider 4-tuples of columns of \(H_{\tilde{C}}\).

\(J_4 \leftarrow \) coordinate 4-tuple that maximizes \(r(J_4)\);

if \(3(r(J_4) - 1) > \Delta X^*_T\) then

\(S \leftarrow \emptyset\); Append \(J_4\) to \(S\); \(\Delta X^*_T \leftarrow 3(r(J_4) - 1)\);

end

else if \(3(r(J_4) - 1) = \Delta X^*_T\) then Append \(J_4\) to \(S\);

// Consider pairs of rows of \(H_{\tilde{C}}\).

\(J_1 \leftarrow \) largest coordinate subset such that \(r(J_1) = 2\);

if \(|J_1| - 1 > \Delta X^*_T\) then

\(S \leftarrow \emptyset\); Append \(J_1\) to \(S\); \(\Delta X^*_T \leftarrow |J_1| - 1\);

end

else if \(|J_1| - 1 = \Delta X^*_T\) then Append \(J_1\) to \(S\);

// Consider 3-tuples of rows of \(H_{\tilde{C}}\).

\(J_3 \leftarrow \) largest coordinate subset such that \(r(J_3) = 3\);

if \(2(|J_3| - 1) > \Delta X^*_T\) then

\(S \leftarrow \emptyset\); Append \(J_3\) to \(S\); \(\Delta X^*_T \leftarrow 2(|J_3| - 1)\);

end

else if \(2(|J_3| - 1) = \Delta X^*_T\) then Append \(J_3\) to \(S\);

// Consider 4-tuples of rows of \(H_{\tilde{C}}\).

\(J_k \leftarrow \) largest coordinate subset such that \(r(J_k) = 4\);

if \(3(|J_k| - 1) > \Delta X^*_T\) then

\(S \leftarrow \emptyset\); Append \(J_k\) to \(S\); \(\Delta X^*_T \leftarrow 3(|J_k| - 1)\);

end

else if \(3(|J_k| - 1) = \Delta X^*_T\) then Append \(J_k\) to \(S\);

return \(S\)

Algorithm 2: Heuristic for generating candidate partial parity symbols.

**Input:** Binary parity-check matrix \(H_C\).

**Output:** Binary generalized parity-check matrix \(H_{\tilde{C}}\).

\(H_{\tilde{C}} \leftarrow H_C\);

while \(\text{GTG}(H_{\tilde{C}})\) contains 4-cycles do

\(S \leftarrow \) set of candidate partial parity symbol subsets from Algorithm 2

\(J^* \leftarrow \) subset in \(S\) the insertion of which reduces the most 4-cycles in \(\text{GTG}(H_{\tilde{C}})\);

Insert symbol corresponding to \(J^*\) in \(H_{\tilde{C}}\);

end

return \(H_{\tilde{C}}\)

Algorithm 3: Greedy heuristic for the removal of 4-cycles in binary generalized Tanner graphs.
C. A Greedy Heuristic for $2^m$-ary Model Extraction

For most codes, the decoding algorithms implied by generalized Tanner graphs exhibit only modest gains with respect to those implied by Tanner graphs, if any, thus motivating the search for more complex graphical models. Algorithm 4 iteratively applies the constraint merging operation in order to obtain a $2^m$-ary graphical model from an initial Tanner graph $\text{TG}(H_C)$ for some prescribed maximum complexity $m^*$. At each step, Algorithm 4 determines the pair of local constraints $C_i$ and $C_j$ which when merged reduces the most short cycles without violating the maximum complexity constraint $m^*$. In order to ensure that the efficient cycle counting algorithm of [16] can be utilized, only pairs of checks which are both internal or both interface are merged at each step. Since the initial Tanner graph is bipartite with vertex classes corresponding to interface (repetition) and internal (single parity-check) constraints, the graphical models that result from every such local constraint merge operations are similarly bipartite.

**Input:** Tanner graph $\text{TG}(H_C)$. Max. complexity $m^*$.  
**Output:** $2^m$-ary graphical model GM for $C$.

$$\text{GM} \leftarrow \text{TG}(H_C);$$

**repeat** 

- $(C_i, C_j) \leftarrow$ pair of incident or internal constraints the removal of which removes the most 4-cycles from GM while not violating the $2^m$-ary complexity constraint;
- Merge local constraints $C_i$ and $C_j$ in GM;

**until** No allowed 4-cycle reducing merge operations remain;

**return** GM

**Algorithm 4:** Greedy heuristic for the extraction of $2^m$-ary graphical models.

D. Simulation Results

The proposed extraction heuristics were applied to two extended BCH codes with parameters $[32, 21, 6]$ and $[64, 51, 6]$, respectively. In both Figures 6 and 7 the performance of a number of suboptimal SISO decoding algorithms for these codes is compared to algebraic hard-in hard-out (HIHO) decoding and optimal trellis SISO decoding. Binary antipodal signaling over AWGN channels is assumed throughout.

Initial parity-check matrices $H$ were formed by extending cyclic parity-check matrices for the respective $[31, 21, 5]$ and $[63, 51, 5]$ BCH codes [31]. These initial parity-check matrices were used as inputs to Algorithm 1 yielding the parity-check matrices $H'$, which in turn were used as inputs to Algorithm 2 yielding 4-cycle-free generalized Tanner graphs. The suboptimal decoding algorithms implied by these graphical models are labeled $\text{TG}(H)$, $\text{TG}(H')$, and $\text{GTG}(H')$, respectively. The generalized Tanner graphs extracted for the $[32, 21, 6]$ and $[64, 51, 6]$ codes correspond to degree-17 and degree-40 generalized extensions, respectively. Finally, the parity-check matrices $H'$ were used as inputs to Algorithm 3 with various values of $m^*$. The number 4-, 6-, and 8-cycles ($N_4$, $N_6$, $N_8$) contained in the extracted graphical models for the $[32, 21, 6]$ and $[64, 51, 6]$ codes are given in Tables II and III, respectively.

The utility of Algorithm 1 is illustrated in both Figures 6 and 7: the $\text{TG}(H')$ algorithms outperform the $\text{TG}(H)$ algorithms by approximately 0.1 dB and 0.5 dB at a bit error rate (BER) of $10^{-4}$ for the $[32, 21, 6]$ and $[64, 51, 6]$ codes, respectively. For both codes, the 4-cycle-free generalized Tanner graph decoding algorithms outperform Tanner graph decoding by approximately 0.2 dB at a BER of $10^{-4}$. Further performance improvements are achieved for both codes by going beyond binary models. Specifically, at a BER of $10^{-5}$, the suboptimal SISO decoding algorithm implied by the extracted 16-ary graphical model for the $[32, 21, 6]$ code outperforms algebraic HIHO decoding by approximately 1.5 dB. The minimal trellis for this code is known to contain state variables with alphabet size at least 1024 [34], yet the 16-ary suboptimal SISO decoder performs only 0.7 dB worse at a BER of $10^{-5}$. At a BER of $10^{-4}$, the suboptimal SISO decoding algorithm implied by the extracted 32-ary graphical model for the $[64, 51, 6]$ code outperforms algebraic HIHO decoding by approximately 1.2 dB. The minimal trellis for this code is known to contain state variables with alphabet size at least 4096 [34]; that a 32-ary suboptimal SISO decoder loses only 0.7 dB with respect to the optimal SISO decoder at a BER of $10^{-4}$ is notable.

![Figure 6: Bit error rate performance of different decoding algorithms for the $[32, 21, 6]$ extended BCH code. Fifty iterations of a flooding schedule were performed for all of the suboptimal SISO decoding algorithms.](attachment:image.png)
performed for all of the suboptimal SISO decoding algorithms.

of those codes which meet or approach the TI-CSB is also an
terms of the more familiar minimal trellis complexity. A study

a bound would allow for a re-expression of the TI-CSB in

complexity

open problem is the use of the Cut-Set Bound to establish an

There are also a number of interesting directions for future work motivated by the study of graphical model transformation. While the extracted graphical models presented in Section [VI-D] are notable, ad-hoc techniques utilizing massively redundant models and judicious message filtering outperform the models presented in this work [49], [50]. Such massively redundant models contain many more short cycles than the models presented in Section [VI-D] indicating that short cycle structure alone is not a sufficiently meaningful cost measure for graphical model extraction. It is known that redundancy can be used to remove pseudocodewords (cf. [53]) thus motivating the study of cost measures which consider both short cycle structure and pseudocodeword spectrum. Finally, it would be interesting to study extraction heuristics beyond simple greedy searches, as well as those which use all of the basic graphical model operations (rather than just constraint merging).

APPENDIX

This appendix provides detailed definitions of both the \( q \)-ary graphical model properties described in Section [III-B] and the basic graphical model operations introduced in Section [VI]. The proof of Lemma [II] is also further illustrated by example. In order to elucidate these properties and definitions, a single-cycle graphical model for the extended Hamming code is studied throughout.

A. Single-Cycle Model for the Extended Hamming Code

Figure [1] illustrates a single-cycle graphical model (i.e. a tail-biting trellis) for the length 8 extended Hamming code \( C_H \). The hidden variables \( S_1 \) and \( S_3 \) are binary while \( S_2, S_3, S_6, S_7, \) and \( S_8 \) are 4-ary. All of the local constraint codes in this model are interface constraints. Equations [44]-[47] define the local constraint codes via generator matrices.

![Figure 8. Tail-biting trellis graphical model for the length 8 extended Hamming code \( C_8 \).](image-url)
pose there exists some hidden variable $C$ in Section III-B are discussed in detail in the following where $B$ for a linear code $q$-ary Graphical Model Properties

$$G_1 = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 01 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 10 & 1 & 10 \\ 01 & 1 & 01 \end{bmatrix} \tag{44}$$

$$G_3 = \begin{bmatrix} 10 & 0 & 01 \\ 01 & 0 & 11 \\ 00 & 1 & 01 \end{bmatrix}, \quad G_4 = \begin{bmatrix} 10 & 1 & 0 \\ 01 & 1 & 1 \end{bmatrix} \tag{45}$$

$$G_5 = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 01 \end{bmatrix}, \quad G_6 = \begin{bmatrix} 10 & 1 & 10 \\ 01 & 1 & 01 \end{bmatrix} \tag{46}$$

$$G_7 = \begin{bmatrix} 10 & 0 & 1 \\ 01 & 0 & 11 \\ 00 & 1 & 01 \end{bmatrix}, \quad G_8 = \begin{bmatrix} 10 & 1 & 0 \\ 01 & 1 & 1 \end{bmatrix} \tag{47}$$

The graphical model for $C_H$ illustrated in Figure 8 is 4-ary (i.e. $q = 2$, $m = 2$): the maximum hidden variable alphabet index set size is 2 and all local constraints satisfy $\min(k(C_i), n(C_i) - k(C_i)) \leq 2$. The behavior, $B_H$, of this graphical model is generated by

$$G_B = \begin{bmatrix} G_1 & G_2 & G_3 & G_4 & G_5 & G_6 & G_7 & G_8 \end{bmatrix}$$

The projection of $B_H$ onto the visible variable index set $I$, $B_{H,I}$, is thus generated by

$$G_{B_{H,I}} = \begin{bmatrix} V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 & V_8 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{48}$$

which coincides precisely with a generator matrix for $C_H$.

B. $q^m$-ary Graphical Model Properties

The three properties of $q^m$-ary graphical models introduced in Section III-B are discussed in detail in the following where it is assumed that a $q^m$-ary graphical model $G_C$ with behavior $B$ for a linear code $C$ over $\mathbb{F}_q$ defined on an index set $I$ is given.

1) Internal Local Constraint Involvement Property: Suppose there exists some hidden variable $S_j$ (involved in the local constraints $C_{j_1}$ and $C_{j_2}$) that does not satisfy the local constraint involvement property. A new hidden variable $S_i$ that is a copy of $S_j$ is introduced to $G_C$ by first redefining $C_{j_2}$ over $S_i$ and then inserting a local repetition constraint $C_i$ that enforces $S_j = S_i$. The insertion of $S_i$ and $C_i$ does not fundamentally alter the complexity of $G_C$ since $n(C_i) - k(C_i) = |T_j| \leq m$ and since degree-2 repetition constraints are trivial from a decoding complexity viewpoint. Furthermore, the insertion of $S_i$ and $C_i$ does not fundamentally alter the cyclic topology of $G_C$ since no new cycles can be introduced by this procedure.

As an example, consider the binary hidden variable $S_1$ in Figure 8 which is incident on the interface constraints $C_1$ and $C_8$. By introducing the new binary hidden variable $S_9$ and binary repetition constraint $C_9$, as illustrated in Figure 9, $S_1$ can be made to be incident on the internal constraint $C_9$. The

![Fig. 9. Insertion of hidden variable $S_9$ and internal local constraint $C_9$ into the tail-biting trellis for $C_H$.](image)

insertion of $S_9$ and $C_9$ redefines $C_8$ over $S_9$ resulting in the generator matrices

$$G_8 = \begin{bmatrix} 10 & 1 & 0 \\ 01 & 1 & 1 \end{bmatrix}, \quad G_9 = \begin{bmatrix} 1 & 1 \end{bmatrix} \tag{50}$$

Clearly, the modified local constraints $C_8$ and $C_9$ satisfy the condition for inclusion in a 4-ary graphical model.

2) Internal Local Constraint Removal Property: The removal of the internal constraint $C_r$ from $G_C$ in order to define the new code $C^r$ proceeds as follows. Each hidden variable $S_i$, $i \in I_S(r)$, is first disconnected from $C_r$ and connected to a new degree-1 internal constraint $C'_r$ which does not impose any constraint on the value of $S_i$ (since it is degree-1). The local constraint $C_r$ is then removed from the resulting graphical model yielding $G_{C^r}$ with behavior $B^r$. The new code $C^r$ is the projection of $B^r$ onto $I$.

As an example, consider the removal of the internal local constraint $C_9$ from the graphical model for $C_H$ described above; the resulting graphical model update is illustrated in Figure 10. The new codes $C_{10}$ and $C_{11}$ are length 1, dimension 1 codes which thus impose no constraints on $S_1$ and $S_9$, respectively. It is readily verified that the code $C_{10}^r$ which results from the removal of $C_9$ from $C_H$ has dimension 5 and is generated by

$$G_{H,5} = \begin{bmatrix} V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 & V_8 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{51}$$

Note that $C_{10}^r$ corresponds to all paths in the tail-biting trellis representation of $C_H$, not just those paths which begin and end in the same state.
The removal of an internal local constraint $C_r$ results in the introduction of $|I_S(r)|$ new degree-1 local constraints. Forney described such constraints as “useless” in [30] and they can indeed be removed from $G_{C}^{(r)}$, since they impose no constraints on the variables they involve. Specifically, for each hidden variable $S_i$, $i \in I_S(r)$, involved in the (removed) local constraint $C_r$, denote by $C_j$ the other constraint involving $S_i$ in $G_C$. The constraint $C_j$ can be redefined as its projection onto $I_V(j) \cup \{I_S(j) \setminus i\}$. It is readily verified that the resulting constraint $C_j^{(r)}$ satisfies the condition for inclusion in a $q^m$-ary graphical model.

Continuing with the above example, $C_{10}$, $C_{11}$, $S_1$, and $S_9$ can be removed from the graphical model illustrated in Figure 10 by redefining $C_1$ and $C_8$ with generator matrices

$$G_3 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad G_8 = \begin{bmatrix} 10 & 1 \\ 0 & 1 \end{bmatrix}.$$  

(52)

3) Internal Local Constraint Redefinition Property: Let $C_i$ satisfy $n(C_i) - k(C_i) = m' \leq m$ and consider a hidden variable $S_j$ involved in $C_i$ (i.e. $j \in I_S(i)$) with alphabet index set $T_j$. Each of the $|T_j|$ coordinates of $S_j$ can be redefined as a $q$-ary sum of some subset of the visible variable set as follows. Consider the behavior $B_{S_j}$ and corresponding code $C_{S_j}$ which result when $C_i$ is removed from $G_{C}$ (before $S_j$ is discarded). The projection of $B_{S_j}$ onto $T_j \cup I$, $B_{S_j,T_j,I}$, has length

$$n(C) + |T_j| + I$$  

(53)

and dimension

$$k(C_{S_j}) \geq k(C)$$  

(54)

over $\mathbb{F}_q$. There exists a generator matrix for $B_{S_j,T_j,I}$ that is systematic in some size $k(C_{S_j})$ subset of the index set $I$ [31]. A parity-check matrix $H_j$ that is systematic in the $|T_j|$ positions corresponding to the coordinates of $S_j$ can thus be found for this projection; each coordinate of $S_j$ is defined as a $q$-ary sum of some subset of the visible variables by $H_j$. Following this procedure, the internal local constraint $C_i$ is redefined over $I$ by substituting the definitions of $S_j$ implied by $H_j$ for each $j \in I_S(i)$ into each of the $m'$ $q$-ary single parity-check equations which determine $C_i$.

Returning to the example of the tail-biting trellis for $C_H$, the internal local constraint $C_9$ is redefined over the visible variable set as follows. The projection of $B_{S_9}^{\varphi}$ onto $T_1 \cup I$ is generated by

$$C_{S_9,T_1,I} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$  

(55)

A valid parity-check matrix for this projection which is systematic in the position corresponding to $S_1$ is

$$H_1 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix},$$  

(56)

which defines the binary hidden variable $S_1$ as

$$S_1 = V_1 + V_2$$  

(57)

where addition is over $\mathbb{F}_2$. A similar development defines the binary hidden variable $S_9$ as

$$S_9 = V_5 + V_8.$$  

(58)

The local constraint $C_9$ thus can be redefined to enforce the single parity-check equation

$$V_1 + V_2 + V_5 + V_8 = 0.$$  

(59)

Finally, in order to illustrate the use of the $q^m$-ary graphical model properties in concert, denote by $C_9^{(1)}$ the single parity-check constraint enforcing (59). It is readily verified that only the first four rows of $H_{S_9}^{(1)}$ (as defined in (51)) satisfy $c_9^{(1)}$. It is precisely these four rows which generate $C_H$ proving that

$$C_H = C_9^{(1)} \cap C_9^{(1)}.$$  

(60)

C. Illustration of Proof of Lemma 7

In the following, the proof of Lemma 1 is illustrated by updating a cycle-free model for $C_H$ with the single parity-check constraint defined by (59) in order to obtain a cycle-free graphical model for $C_H$. A cycle-free binary graphical model for $C_H$ is illustrated in Figure 11. All hidden variables in Figure 11 are binary and the local constraints labeled $C_{14}$, $C_{17}$, $C_{20}$, and $C_{23}$ are binary single parity-check constraints while the remaining local constraints are repetition codes. By construction, it has thus been shown that

$$t(C_H^{(1)}) = 1.$$  

(61)

In light of (59) and (60), a 4-ary graphical model for $C_H$ can be constructed by updating the graphical model illustrated in Figure 11 to enforce a single parity-check constraint on $V_1$, $V_2$, $V_5$, and $V_8$. A natural choice for the root of the minimal spanning tree containing the interface constraints incident on

\footnote{In order to emphasize that the code and hidden variable labels in Figure 11 are in no way related to those labels used previously, the labeling of hidden variables and local constraints begin at $S_{12}$ and $C_{12}$, respectively.}
Finally, for the redefined variables incident on these constraints, need updating. The hidden variables are accomplished by redefining the local constraints \( C_{14}, C_{17}, C_{20}, \) and \( C_{23} \); the respective generator matrices for the redefined codes are

\[
G_{14} = \begin{bmatrix}
1 & 0 & 11 \\
0 & 1 & 10
\end{bmatrix}, \quad G_{17} = \begin{bmatrix}
1 & 0 & 11 \\
0 & 1 & 10
\end{bmatrix}
\]

(62)

\[
G_{18} = \begin{bmatrix}
1 & 0 & 11 \\
0 & 1 & 10
\end{bmatrix}, \quad G_{21} = \begin{bmatrix}
1 & 0 & 10 \\
0 & 1 & 11
\end{bmatrix}
\]

(63)

The updated graphical model is 4-ary and it has thus been shown by construction that

\[
t(C_H) \leq t(C_H^g) + 1 = 2.
\]

(65)

### D. Graphical Model Transformations

The eight basic graphical model operations introduced in Section V are discussed in detail in the following where it is assumed that a \( q^m \)-ary graphical model \( G_C \) with behavior \( \mathcal{B} \) for a linear code \( C \) over \( \mathbb{F}_q \) defined on an index set \( I \) is given.

1) Local Constraint Merging: Suppose that two local constraints \( C_{i_1} \) and \( C_{i_2} \) are to be merged. Without loss of generality, assume that there is no hidden variable incident on both \( C_{i_1} \) and \( C_{i_2} \) (since if there is, a degree-2 repetition constraint can be inserted). The hidden variables incident on \( C_{i_1} \) may be partitioned into two sets

\[
I_S(i_1) = I_S^{(c)}(i_1) \cup I_S^{(nc)}(i_1)
\]

(66)

where each \( S_j \), \( j \in I_S^{(c)}(i_1) \), is also incident on a constraint \( C_j \) which is adjacent to \( C_{i_1} \). The hidden variables incident on \( C_{i_2} \) may be similarly partitioned. The set of local constraints incident on hidden variables in both \( I_S^{(c)}(i_1) \) and \( I_S^{(c)}(i_2) \) are denoted common constraints and indexed by \( I_C^{(c)}(i_1, i_2) \). Figure 13 illustrates this notation.

The merging of local constraints \( C_{i_1} \) and \( C_{i_2} \) proceeds as follows. For each common local constraint \( C_j \), \( j \in I_C^{(c)}(i_1, i_2) \), denote by \( S_{j_1} \) (\( S_{j_2} \)) the hidden variable incident on \( C_j \) and \( C_{i_1} \) (\( C_{i_2} \)). Denote by \( C_{j_{(j_1, j_2)}} \) the projection of \( C_j \) onto the two variable index set \( \{j_1, j_2\} \) and define a new \( q^{k(C_{j_{(j_1, j_2)}})} \)-ary hidden variable \( S_{j_{(j_1, j_2)}} \) which encapsulates the possible simultaneous values of \( S_{j_{1}} \) and \( S_{j_{2}} \) (as constrained by \( C_{j_{(j_1, j_2)}} \)). After defining such hidden variables for each \( C_j \), \( j \in I_C^{(c)}(i_1, i_2) \), a set of new hidden variables results which is indexed by \( I_S(i_1, i_2) \). The local constraints \( C_{i_1} \) and \( C_{i_2} \) are then merged by replacing \( C_{i_1} \) and \( C_{i_2} \) by a code defined over

\[
I_V(i_1) \cup I_V(i_2) \cup I_S^{(nc)}(i_1) \cup I_S^{(nc)}(i_2) \cup I_S^{(c)}(i_1, i_2)
\]

(67)

which is equivalent to \( C_{i_1} \cap C_{i_2} \) and redefining each local constraint \( C_j \), \( j \in I_C(i_1, i_2) \), over the appropriate hidden variables in \( I_S(i_1, i_2) \).

As an example, consider again the 4-ary cycle-free graphical model for \( C_H \) derived in the previous section, a portion of which is re-illustrated on the bottom left of Figure 13 and suppose that the local constraints \( C_{14} \) and \( C_{17} \) are to be merged.
The local constraints $C_{14}$, $C_{17}$, and $C_{24}$ are defined by (62) and (64).

The hidden variables incident on $C_{14}$ are partitioned into the sets $I_S^{(c)}(14) = \{14\}$ and $I_S^{(nc)}(14) = \{12, 13\}$. Similarly, $I_S^{(c)}(17) = \{17\}$ and $I_S^{(nc)}(17) = \{15, 16\}$. The sole common constraint is thus $C_{24}$. The projection of $C_{24}$ onto $S_{14}$ and $S_{17}$ has dimension 3 and the new 8-ary hidden variable $S_{25}$ is defined by the generator matrix

$$G_{14,17} = \begin{bmatrix} s_{25} & s_{14} & s_{27} \\ 100 & 10 & 10 \\ 010 & 01 & 00 \\ 001 & 00 & 01 \end{bmatrix}. \quad (68)$$

The local constraints $C_{14}$ and $C_{17}$ when defined over $S_{25}$ rather than $S_{14}$ and $S_{17}$, respectively, are generated by

$$G'_{14} = \begin{bmatrix} 1 & 0 & 110 \\ 1 & 0 & 111 \\ 0 & 1 & 100 \end{bmatrix}, \quad G'_{17} = \begin{bmatrix} 1 & 0 & 101 \\ 1 & 0 & 111 \\ 0 & 1 & 100 \end{bmatrix}. \quad (69)$$

Finally, $C_{24}$ is redefined over $S_{25}$ and generated by

$$G_{24} = \begin{bmatrix} s_{25} & s_{26} & s_{23} \\ 100 & 10 & 10 \\ 010 & 00 & 01 \\ 001 & 00 & 01 \\ 000 & 01 & 01 \end{bmatrix}. \quad (70)$$

while $C_{14}$ and $C_{17}$ are replaced by $C_{25}$ which is equivalent to $C_{14} \cap C_{17}$ and is generated by

$$G_{25} = \begin{bmatrix} s_{25} & s_{12} & s_{13} & s_{14} & s_{16} \\ 100 & 0 & 1 & 0 & 1 \\ 010 & 1 & 1 & 0 & 0 \\ 001 & 0 & 0 & 0 & 1 \\ 000 & 0 & 1 & 1 & 1 \end{bmatrix}. \quad (71)$$

Note that the graphical model which results from the merging of $C_{14}$ and $C_{17}$ is 8-ary. Specifically, $S_{24}$ is an 8-ary hidden variable while $n(C_{24}) = k(C_{24}) = 3$ and $k(C_{25}) = 3$.

2) Local Constraint Splitting: Local constraint splitting is simply the inverse operation of local constraint merging. Consider a local constraint $C_j$ defined on the visible and hidden variables indexed by $I_V(j)$ and $I_S(j)$, respectively. Suppose that $C_j$ is to be split into two local constraints $C_{j1}$ and $C_{j2}$ defined on the index sets $I_V(j_1) \cup I_S(j_1)$ and $I_V(j_2) \cup I_S(j_2)$, respectively, such that $I_V(j_1)$ and $I_V(j_2)$ partition $I_V(j)$ while $I_S(j_1) \cup I_S(j_2) = I_S(j)$ but $I_S(j_1)$ and $I_S(j_2)$ need not be disjoint. Denote by $I_S(j_1, j_2)$ the intersection of $I_S(j_1)$ and $I_S(j_2)$. Local constraint splitting proceeds as follows. For each $S_i, i \in I_S(j_1, j_2)$, make a copy $S_{ip}$ of $S_i$ and redefine the local constraint incident on $S_i$ (which is not $C_j$) over both $S_i$ and $S_{ip}$. Denote by $I'_S(j_1, j_2)$ an index set for the copied hidden variables. The local constraint $C_j$ is then replaced by $C_{j1}$ and $C_{j2}$ such that $C_{j1}$ is defined over $I_V(j_1) \cup I_S(j_1)$ and $C_{j2}$ is defined over $I_V(j_2) \cup I_S(j_2) \setminus I_S(j_1, j_2) \cup I'_S(j_1, j_2)$. \quad (72)

Following this split procedure, some of the hidden variables in $I_S(j_1, j_2)$ and $I'_S(j_1, j_2)$ may have larger alphabets than necessary. Specifically, if the dimension of the projection of $C_{j1}$ ($C_{j2}$) onto a variable $S_i, i \in I_S(j_1, j_2)$ ($i \in I'_S(j_1, j_2)$), is smaller than the alphabet index set size of $S_i$, then $S_i$ can be redefined with an alphabet index set size equal to that dimension.

The merged code in the example of the previous section $C_{25}$ can be split into two codes: $C_{14}$ defined on $S_{12}, S_{13}$, and $S_{24}$, and $C_{17}$ defined on $S_{15}, S_{16}$, and $S_{24}'$. The projection of $S_{24}$ onto $C_{14}$ has dimension 2 and $S_{24}$ can thus be replaced by the 4-ary hidden variable $S_{14}$. Similarly, the projection of $S_{24}'$ onto $C_{17}$ has dimension 2 and $S_{24}'$ can be replaced by the 4-ary hidden variable $S_{17}$.

3) Insertion/Removal of Degree-2 Repetition Constraints:
Suppose that $S_i$ is a hidden variable involved in the local constraints $C_{i1}$ and $C_{i2}$. A degree-2 repetition constraint is inserted by defining a new hidden variable $S_i$ as a copy of $S_i$, redefining $C_{i2}$ over $S_i$ and defining the repetition constraint $C_j$ which enforces $S_i = S_j$. Degree-2 repetition constraint insertion can be similarly defined for visible variables. Conversely, suppose that $C_j$ is a degree-2 repetition constraint incident on the hidden variables $S_i$ and $S_j$. Since $C_j$ simply enforces $S_i = S_j$, it can be removed and $S_j$ relabeled $S_i$. Degree-2 repetition constraint removal can be similarly defined for visible variables. The insertion and removal of degree-2 repetition constraints is illustrated in Figures 14(a) and 14(b) for hidden and visible variables, respectively.

![Fig. 14. Insertion and removal of degree-2 repetition constraints.](image-url)
4) Insertion/Removal of Trivial Constraints: Trivial constraints are those incident on no hidden or visible variables so that their respective block lengths and dimensions are zero. Trivial constraints can obviously be inserted or removed from graphical models.

5) Insertion/Removal of Isolated Partial Parity-Check Constraints: Suppose that \( C_1, \ldots, C_j \) are \( j \) \( q \)-ary repetition constraints (that is each repetition constraint enforces equality on \( q \)-ary variables) and let \( \beta_{i_1}, \ldots, \beta_{i_j} \in \mathbb{F}_q \) be non-zero. The insertion of an isolated partial parity-check constraint is defined as follows. Define \( j + 1 \) new \( q \)-ary hidden variables \( S_{i_1}, \ldots, S_{i_j} \) and \( S_k \), and two new local constraints \( C_p \) and \( C_k \) such that \( C_p \) enforces the \( q \)-ary single parity-check equation

\[
\sum_{i=1}^{j} \beta_{i_j} S_{i_j} = S_k \quad (73)
\]

and \( C_k \) is a degree-1 constraint incident only on \( S_k \) with dimension 1. The new local constraint \( C_k \) defines the partial parity variable \( S_k \) and is denoted isolated since it is incident on a hidden variable which is involved in a degree-1, dimension 1 local constraint (i.e. \( C_k \) does not constrain the value of \( S_k \)). Since \( C_p \) is isolated, the graphical model that results from its insertion is indeed a valid model for \( C \). Similarly, any such isolated partial parity-check constraint can be removed from a graphical model resulting in a valid model for \( C \).

As an example, Figure 15 illustrates the insertion and removal of an isolated partial-parity check on the binary sum of \( V_7 \) and \( V_8 \) in a Tanner graph for \( C_H \) corresponding to \( \text{[49]} \) (note that \( C_H \) is self-dual so that the generator matrix defined in \( \text{[49]} \) is also a valid parity-check matrix for \( C_H \)).

Fig. 15. The insertion/removal of an isolated partial-parity check on \( V_7 \) and \( V_8 \) in a Tanner graph for \( C_H \).

REFERENCES

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” in Proc. International Conf. Communications, Geneva, Switzerland, May 1993, pp. 1064–1070.
[2] R. G. Gallager, “Low density parity check codes,” IEEE Trans. Information Theory, vol. 8, pp. 21–28, January 1962.
[3] M. Sipser and D. A. Spielman, “Expander codes,” IEEE Trans. Information Theory, vol. 42, pp. 1660–1686, November 1996.
[4] D. J. C. MacKay, “Good error-correcting codes based on very sparse matrices,” IEEE Electronics Letters, vol. 33, pp. 457–458, March 1997.
[5] N. Wiberg, H.-A. Loeliger, and R. Kötter, “Codes and iterative decoding on general graphs,” European Trans. Telecommun., vol. 6, no. 5, pp. 513–525, Sept-Oct. 1995.
[6] N. Wiberg, “Codes and decoding on general graphs,” Ph.D. dissertation, Linköping University (Sweden), 1996.
[7] S. M. Aji and R. J. McEliece, “The generalized distributive law,” IEEE Trans. Information Theory, vol. 46, no. 2, pp. 325–343, March 2000.
[8] K. M. Chugg, A. Anastasopoulos, and X. Chen, Iterative Detection: Adaptivity, Complexity Reduction, and Applications. Kluwer Academic Publishers, 2001.
[9] F. Kschischang, B. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” IEEE Trans. Information Theory, vol. 47, pp. 498–519, Feb. 2001.
[10] G. D. Forney, Jr., “Codes on graphs: Normal realizations,” IEEE Trans. Information Theory, pp. 520–548, February 2001.
[11] R. M. Tanner, “A recursive approach to low complexity codes,” IEEE Trans. Information Theory, vol. IT-27, pp. 533–547, September 1981.
[12] Y. Mao and A. H. Banaihashemi, “A heuristic search for good low-density parity-check codes at short block lengths,” in Proc. International Conf. Communications, vol. 1, Helsinki, Finland, June 2001, pp. 41–44.
[13] D. M. Arnold, E. Eleftheriou, and X. Y. Hu, “Progressive edge-growth Tanner graphs,” in Proc. Globecom Conf., vol. 2, San Antonio, TX, November 2001, pp. 997–1001.
[14] Y. X. Ge, D. Eppstein, and P. Smyth, “The distribution of loop lengths in graphical models for turbo decoding,” IEEE Trans. Information Theory, vol. 47, no. 6, pp. 2549–2553, September 2001.
[15] T. Tian, C. R. Jones, J. D. Villasenor, and R. D. Wesel, “Selective avoidance of cycles in irregular LDPC code construction,” IEEE Trans. Information Theory, vol. 52, no. 8, pp. 1242–1247, August 2004.
[16] T. R. Halfond and K. M. Chugg, “An algorithm for counting short cycles in bipartite graphs,” IEEE Trans. Information Theory, vol. 52, no. 1, pp. 287–292, January 2006.
[17] S. Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke, “On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit,” IEEE Communications Letters, vol. 5, no. 2, pp. 58–60, February 2001.
[18] T. Richardson, M. Shokrollahi, and R. Urbanke, “Design of capacity-approaching irregular low-density parity-check codes,” IEEE Trans. Information Theory, vol. 47, no. 2, pp. 619–673, February 2001.
[19] P. Oswald and A. Shokrollahi, “Capacity-achieving sequences for the erasure channel,” IEEE Trans. Information Theory, vol. 48, no. 12, pp. 3017–3028, December 2002.
[20] H. D. Pfister, I. Sason, and R. Urbanke, “Capacity-achieving ensembles for the binary erasure channel with bounded complexity,” IEEE Trans. Information Theory, vol. 51, no. 7, pp. 2352–2379, July 2005.
[21] S. Benedetto and G. Montorsi, “Design of parallel concatenated convolutional codes,” IEEE Trans. Communications, vol. 44, pp. 591–600, May 1996.
[22] S. Dolinar, D. Divsalar, and F. Pollara, “Code performance as a function of block size,” Jet Propulsion Labs., Pasadena, CA, Tech. Rep. TDA Progress Report 42-133, May 1998.
[23] C. Berrou, “The ten-year-old turbo codes are entering into service,” IEEE Communications Mag., vol. 41, no. 8, pp. 110–116, August 2003.
[24] S. ten Brink, “Convergence behavior of iteratively decoded parallel concatenated codes,” IEEE Trans. Communications, vol. 49, no. 10, pp. 1727–1737, October 2001.
[25] K. M. Chugg, P. Thienniboon, G. D. Dimou, P. Gray, and J. Melzer, “A new class of turbo-like codes with universally good performance and high-speed decoding,” in Proc. IEEE Military Comm. Conf., Atlantic City, NJ, October 2005.
[26] T. R. Halford, A. J. Grant, and K. M. Chugg, “Which codes have 4-cycle-free Tanner graphs?” IEEE Trans. Information Theory, vol. 52, no. 9, pp. 4219–4223, September 2006.
[27] A. Vardy, Handbook of Coding Theory. Amsterdam, The Netherlands: Elsevier, 1999, ch. Trellis structure of codes, eds.: V. S. Pless and W. C. Huffman.
[28] A. R. Calderbank, G. D. Forney, Jr., and A. Vardy, “Minimal tail-hitting trellises: The Golay code and more,” IEEE Trans. Information Theory, vol. 45, no. 5, pp. 1435–1455, July 1999.
