CONSTRAINT COMPLEXITY OF REALIZATIONS OF LINEAR CODES ON ARBITRARY GRAPHS

NAVIN KASHYAP

ABSTRACT. A graphical realization of a linear code $\mathcal{C}$ consists of an assignment of the coordinates of $\mathcal{C}$ to the vertices of a graph, along with a specification of linear state spaces and linear "local constraint" codes to be associated with the edges and vertices, respectively, of the graph. The κ-complexity of a graphical realization is defined to be the largest dimension of any of its local constraint codes. κ-complexity is a reasonable measure of the computational complexity of a sum-product decoding algorithm specified by a graphical realization. The main focus of this paper is on the following problem: given a linear code $\mathcal{C}$ and a graph $\mathcal{G}$, how small can the κ-complexity of a realization of $\mathcal{C}$ on $\mathcal{G}$ be? As useful tools for attacking this problem, we introduce the Vertex-Cut Bound, and the notion of "vc-treewidth" for a graph, which is closely related to the well-known graph-theoretic notion of treewidth. Using these tools, we derive tight lower bounds on the κ-complexity of any realization of $\mathcal{C}$ on $\mathcal{G}$. Our bounds enable us to conclude that good error-correcting codes can have low-complexity realizations only on graphs with large vc-treewidth. Along the way, we also prove the interesting result that the ratio of the κ-complexity of the best conventional trellis realization of a length-n code $\mathcal{C}$ to the κ-complexity of the best cycle-free realization of $\mathcal{C}$ grows at most logarithmically with codelength n. Such a logarithmic growth rate is, in fact, achievable.

1. INTRODUCTION

The study of graphical models of codes and the associated message-passing decoding algorithms is a major focus of current research in coding theory. This is attributable to the fact that coding schemes using graph-based iterative decoding strategies — *e.g.*, turbo codes and low-density parity check (LDPC) codes — have low implementation complexity, while their performance is close to the optimum predicted by theory. A unified treatment of graphical models and the associated decoding algorithms began with the work of Wiberg, Loeliger and Koetter [18],[19], and has since been abstracted and refined under the framework of the generalized distributive law [1], factor graphs [12], and normal realizations [6],[7]. In fact, the study of cycle-free graphical models of codes (*i.e.*, models in which the underlying graphs are cycle-free) can be traced back to the introduction of the Viterbi decoding algorithm in the 1960's, which led to the study of trellis representations of codes. A comprehensive account of the history and development of trellis representations can be found in [17].

In this work, we will follow the approach of Forney [6],[7], and Halford and Chugg [8] in studying the general "extractive" problem of constructing low-complexity graphical models for a given linear code. Roughly speaking, a low-complexity graphical model is one that implies a low-complexity decoding algorithm. In particular, in this paper, we investigate the question of how small the complexity of an arbitrary graphical model for a given code can be.

We briefly introduce graphical models here; a detailed description can be found in Section 2. A graph decomposition of a code $\mathcal{C}$ is a mapping of the set of coordinates of $\mathcal{C}$ to the set of vertices of a graph. A graph decomposition may be viewed as an assignment of symbol variables to the vertices of the graph. A graph decomposition can be extended to a graphical model which additionally

Date: May 14, 2008.
This work was supported by a Discovery Grant from the Natural Sciences and Engineering Research Council (NSERC), Canada.
The author is with the Department of Mathematics and Statistics, Queen's University, Kingston, ON K7L 3N6, Canada. Email: nkashyap@mast.queensu.ca.

assigns state variables to the edges of the graph, and specifies a local constraint code at each vertex of the graph. The full behavior of the model is the set of all configurations of symbol and state variables that satisfy all the local constraints. Such a model is called a graphical realization of $\mathcal{C}$ if the restriction of the full behavior to the set of symbol variables is precisely $\mathcal{C}$. The realization is said to be cycle-free if the underlying graph in the model has no cycles. A trellis representation of a code can be viewed as a cycle-free realization in which the underlying graph is a simple path. In contrast, a *tailbiting* trellis representation [13],[14] is a graphical realization in which the underlying graph consists of a single cycle.

We will focus our attention on the case of realizations of linear codes on *connected* graphs only. Indeed, there is no loss of generality in doing so, since a linear code $\mathcal{C}$ has a realization on a graph $\mathcal{G}$ that is not connected if and only if $\mathcal{C}$ can be expressed as the direct sum of codes that may be individually realized on the connected components of $\mathcal{G}$ [6]. In this context, we will refer to cycle-free graphical realizations simply as tree realizations, as the underlying graph is a connected, cycle-free graph, *i.e.*, a tree.

It is by now well known that any graphical realization of a code specifies a canonical iterative message-passing decoding algorithm, namely, the sum-product algorithm, on the underlying graph [1],[6],[12],[18]. When the underlying graph is a tree, the sum-product algorithm provides an exact implementation of maximum-likelihood (ML) decoding. Even when the underlying graph contains cycles, empirical evidence suggests that, in many cases, the sum-product algorithm continues to be a good approximation to ML decoding.

The computational complexity of the sum-product algorithm associated with a graphical realization of a code is largely determined by the sizes of the local constraint codes in the realization. In Section 3 of this paper, we define various measures of "constraint complexity" of a graphical realization that can be used as estimates of the computational complexity of sum-product decoding. These complexity measures may be viewed as generalizations of previously proposed measures of trellis complexity [17],[14], and tree complexity [7],[8]. However, for the most part, we focus on the *κ-complexity* of a graphical realization, which we define to be the maximum of the dimensions of the local constraint codes in the realization.

In the restricted context of tree realizations, it has previously been established that certain "minimal" tree realizations can be canonically defined. Let the term *tree decomposition* denote a graph decomposition in which the graph is a tree. It is known that among all tree realizations of a code $\mathcal{C}$ that extend a given tree decomposition, there is one that minimizes the dimension of the state space at each edge of the underlying tree, and this minimal tree realization is unique [6]. It has further been shown [11] that this unique minimal tree realization also minimizes (among all tree realizations extending the given tree decompositions) the dimension of the local constraint code at each vertex of the tree. In particular, it has the least κ-complexity among all such tree realizations.

In contrast, there is very little known about the general case of realizations of a code on an arbitrary (not necessarily cycle-free) graph. For instance, there appear to be no "canonical" minimal realizations that can be defined in this situation. The only systematic study in this direction remains that of Koetter and Vardy [13],[14], who studied minimal tailbiting trellis representations of codes, which, as already mentioned, are graphical realizations in which the underlying graph consists of exactly one cycle. Beyond this basic (though by no means easy) case, there is little of interest in the literature on the complexity of realizations of codes on arbitrary graphs, the notable exception to this being the work of Halford and Chugg [8].

In their work, Halford and Chugg lay the foundations for a systematic study of complexity of graphical realizations. Their main result is the "Forest-Inducing Cut-Set Bound", which gives a lower bound on the constraint complexity of a graphical realization in terms of its minimal tree complexity. However, this bound does not appear be user-friendly in practice. The main limitation of their approach is that they rely on the Edge-Cut Bound of Wiberg *et al.* [18],[19] to derive their results. While the Edge-Cut Bound has been put to good use in the study of "state complexity" of graphical realizations [5],[6], it is of limited value in the analysis of constraint complexity.

The main aim of our paper is to present useful and tight lower bounds on the constraint complexity of a graphical realization. Our bounds also provide considerable insight into the problem of finding low-complexity graphical realizations. The fundamental tool in our analysis is the Vertex-Cut Bound, which we state and prove in Section 4. The Vertex-Cut Bound is a natural analogue of the Edge-Cut Bound, but as we shall see, it is more suitable for use in the analysis of constraint complexity.

In Section 5, we define a data structure called *vertex-cut tree* that stores the information necessary about a graph to effectively apply the Vertex-Cut Bound. Vertex-cut trees are similar in structure to the junction trees associated with belief propagation algorithms [10],[1]. The *vc-width* of a vertex-cut tree is a measure of the size of the vertex-cut tree, and the *vc-treewidth* of a graph is the least vc-width among all its vertex-cut trees. The vc-treewidth of a graph is very closely related to the notion of treewidth of graphs much studied in graph theory [16],[3].

Using the Vertex-Cut Bound and the notion of vertex-cut trees, we derive, in Section 6, a suite of lower bounds on the κ-complexity of graphical realizations of a linear code $\mathcal{C}$. We state one of these bounds here as an illustrative example. Let $\kappa_{\text{tree}}(\mathcal{C})$ be the least κ-complexity among all tree realizations of $\mathcal{C}$. Consider an arbitrary graph $\mathcal{G}$, and let $\kappa_{\text{vc-tree}}(\mathcal{G})$ denote its vc-treewidth. Then, the κ-complexity of any realization of $\mathcal{C}$ on $\mathcal{G}$ is bounded from below by the ratio $\kappa_{\text{tree}}(\mathcal{C})/\kappa_{\text{vc-tree}}(\mathcal{G})$.

We further apply our methods to answer certain questions raised in [11]. Borrowing terminology from [17], for a code $\mathcal{C}$, let $b(\mathcal{C})$ denote the least edge-complexity of any trellis representation of $\mathcal{C}$ or any of its coordinate permutations. In the language of our paper, $b(\mathcal{C})$ is the least κ-complexity of any conventional trellis realization of $\mathcal{C}$. We show that for any linear code of length n, we have $b(\mathcal{C})/\kappa_{\text{tree}}(\mathcal{C}) = O(\log_2 n)$, and that this is the best possible estimate of the ratio, up to the constant implicit in the O-notation. This is used to extend a known lower bound [15] on $b(\mathcal{C})$ in terms of the length n, dimension k and minimum distance d of $\mathcal{C}$, to a lower bound on $\kappa_{\text{tree}}(\mathcal{C})$.

Our lower bound on $\kappa_{\text{tree}}(\mathcal{C})$ has an important implication. It shows that if $\mathfrak{C}$ is a code family with the property that, for each $\mathcal{C} \in \mathfrak{C}$, $\kappa_{\text{tree}}(\mathcal{C})$ is bounded from above by a fixed constant, then either the dimension or the minimum distance of the codes in $\mathfrak{C}$ grows sub-linearly with codelength. Thus, such code families are not good from a coding-theoretic perspective. We also prove a slightly more general result, which can be roughly interpreted as saying that a good error-correcting code cannot have a low-complexity realization on a graph with small vc-treewidth. So, for good codes, if low-complexity graphical realizations exist, then they must necessarily exist on graphs with large vc-treewidth.

Some concluding remarks are made in Section 7, and an example in support of a statement in Section 3 is given in an appendix.

2. Background and Notation

In this section, we provide the necessary background, and define the notation we use in the paper. We take $\mathbb{F}$ to be an arbitrary finite field. Given a finite index set I, we have the vector space $\mathbb{F}^I = \{\mathbf{x} = (x_i \in \mathbb{F},\ i \in I)\}$. For $\mathbf{x} \in \mathbb{F}^I$ and $J \subseteq I$, the notation $\mathbf{x}|_J$ will denote the *projection* $(x_i,\ i \in J)$. Also, for $J \subseteq I$, we will find it convenient to reserve the use of $\overline{J}$ to denote the set $\{i \in I:\ i \notin J\}$.

2.1. Codes.

A *linear code* over $\mathbb{F}$, defined on the index set I, is a subspace $\mathcal{C} \subseteq \mathbb{F}^I$. In this paper, the terms "code" and "linear code" will be used interchangeably to mean a linear code over an arbitrary finite field $\mathbb{F}$, unless explicitly specified otherwise. The dimension, over $\mathbb{F}$, of $\mathcal{C}$ will be denoted by $\dim(\mathcal{C})$. An $[n,k]$ code is a code of length n and dimension k. If, additionally, the code has minimum distance d, then the code is an $[n,k,d]$ code.

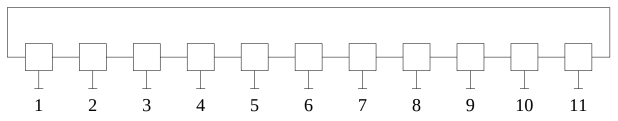


FIGURE 1. A graph decomposition of $I = \{1, 2, 3, \ldots, 11\}$.

Let J be a subset of the index set I. The *projection* of $\mathcal{C}$ onto J is the code $\mathcal{C}|_J = \{\mathbf{c}|_J : \mathbf{c} \in \mathcal{C}\}$, which is a subspace of $\mathbb{F}^J$. We will use $\mathcal{C}_J$ to denote the *cross-section* of $\mathcal{C}$ consisting of all projections $\mathbf{c}|_J$ of codewords $\mathbf{c} \in \mathcal{C}$ that satisfy $\mathbf{c}|_{\overline{J}} = \mathbf{0}$. To be precise, $\mathcal{C}_J = \{\mathbf{c}|_J : \mathbf{c} \in \mathcal{C}, \mathbf{c}|_{\overline{J}} = \mathbf{0}\}$. Note that $\mathcal{C}_J \subseteq \mathcal{C}|_J$. Also, since $\mathcal{C}_J$ is isomorphic to the kernel of the projection map $\pi : \mathcal{C} \to \mathcal{C}|_{\overline{J}}$ defined by $\pi(\mathbf{c}) = \mathbf{c}|_{\overline{J}}$, we have that $\dim(\mathcal{C}_J) = \dim(\mathcal{C}) - \dim(\mathcal{C}|_{\overline{J}})$. As a consequence, we see that if $J \subseteq K \subseteq I$, then $\dim(\mathcal{C}_J) \leq \dim(\mathcal{C}_K)$.

If $\mathcal{C}_1$ and $\mathcal{C}_2$ are codes over $\mathbb{F}$ defined on mutually disjoint index sets I_1 and I_2, respectively, then their *direct sum* is the code $\mathcal{C} = \mathcal{C}_1 \oplus \mathcal{C}_2$ defined on the index set $I_1 \cup I_2$, such that $\mathcal{C}_{I_1} = \mathcal{C}|_{I_1} = \mathcal{C}_1$ and $\mathcal{C}_{I_2} = \mathcal{C}|_{I_2} = \mathcal{C}_2$. This definition naturally extends to multiple codes (or subspaces) $\mathcal{C}_\alpha$, where α is a code identifier that takes values in some set A. Again, it must be assumed that the codes $\mathcal{C}_\alpha$ are defined on mutually disjoint index sets $I_\alpha,\ \alpha \in A$. The direct sum in this situation is denoted by $\bigoplus_{\alpha \in A} \mathcal{C}_\alpha$.

2.2. **Graphs.**

In this paper, we are primarily interested in graphs that are connected, so any unqualifed use of the term "graph" should be taken to mean "connected graph". Let $\mathcal{G} = (V, E)$ be a graph, where V and E denote its vertex and edge sets, respectively. To resolve ambiguity, we will sometimes denote the vertex and edge sets of $\mathcal{G}$ by $V(\mathcal{G})$ and $E(\mathcal{G})$, respectively. Given a $v \in V$, the set of edges incident with v will be denoted by $E(v)$.

For $X \subseteq E$, we define $\mathcal{G} \setminus X$ to be the subgraph of $\mathcal{G}$ obtained by deleting all the edges in X. If X consists of a single edge e, then we will write $\mathcal{G} \setminus e$ instead of $G \setminus \{e\}$. If $\mathcal{G} \setminus X$ is disconnected, then X is called an *edge cut* of $\mathcal{G}$. Similarly, for $W \subseteq V$, we define $\mathcal{G} - W$ to be the subgraph of $\mathcal{G}$ obtained by deleting all the vertices in W along with all incident edges. If W consists of a single vertex v, then we will write $\mathcal{G} - v$ instead of $G - \{v\}$. If $\mathcal{G} - W$ is disconnected, then W is called a *vertex cut* of $\mathcal{G}$.

A tree is a connected graph without cycles. Vertices of degree one in a tree are called *leaves*, and all other vertices are called *internal nodes*. Note that any subset of the edges of a tree constitutes an edge cut, and any subset of internal nodes constitutes a vertex cut of the tree. If e is an edge in a tree T, then we will denote by $T^{(e)}$ and $\overline{T}^{(e)}$ the two components of $T \setminus e$. If v is a vertex of degree δ in a tree T, then we will use $T_1^{(v)}, T_2^{(v)}, \ldots, T_\delta^{(v)}$ to denote the components of $T - v$.

A *path* is a tree with exactly two leaves (the end-points of the path). All internal nodes in a path have degree two. A *simple cycle* is a connected graph in which all vertices have degree two. An *n-cycle*, $n \geq 3$, is a simple cycle with n vertices.

2.3. **Graphical Realizations of Codes.**

The development in this section is based on the exposition of Forney [6],[7]; see also [8],[11]. Let I be a finite index set. A *graph decomposition* of I is a pair $(\mathcal{G}, \omega)$, where $\mathcal{G} = (V, E)$ is a graph, and $\omega : I \to V$ is an *index mapping*. For a code $\mathcal{C}$, we will usually write "graph decomposition of $\mathcal{C}$" as shorthand for "graph decomposition of the index set of $\mathcal{C}$". We again wish to emphasize that, unless explicitly stated otherwise, we will take $\mathcal{G}$ to be a connected graph. When $\mathcal{G}$ is a tree, $(\mathcal{G}, \omega)$ will be called a *tree decomposition*. Pictorially, a graph decomposition $(\mathcal{G}, \omega)$ is depicted as a graph with an additional feature: at each vertex v such that $\omega^{-1}(v)$ is non-empty, we attach special "half-edges", one for each index in $\omega^{-1}(v)$. Figure 1 depicts a graph decomposition of $I = \{1, 2, 3, \ldots, 11\}$.

For a graph $\mathcal{G} = (V, E)$, recall that $E(v)$, $v \in V$, denotes the set of edges incident with v in $\mathcal{G}$. Consider a tuple of the form[1] $(\mathcal{G}, \omega, (\mathcal{S}_e,\ e \in E), (C_v,\ v \in V))$, where

- $(\mathcal{G}, \omega)$ is a graph decomposition of I;
- for each $e \in E$, $\mathcal{S}_e$ is a vector space over $\mathbb{F}$ called a *state space*;
- for each $v \in V$, C_v is a subspace of $\mathbb{F}^{\omega^{-1}(v)} \oplus \left(\bigoplus_{e \in E(v)} \mathcal{S}_e\right)$, called a *local constraint code*, or simply, a *local constraint*.

Such a tuple will be called a *graphical model*. A graphical model in which the underlying graph $\mathcal{G}$ is a tree will be called a *tree model*. The elements of any state space $\mathcal{S}_e$ are called *states*. The index sets of the state spaces $\mathcal{S}_e$, $e \in E$, are taken to be mutually disjoint, and are also taken to be disjoint from the index set I corresponding to the symbol variables.

A *global configuration* of a graphical model as above is an assignment of values to each of the symbol and state variables. In other words, it is a vector of the form $((x_i \in \mathbb{F},\ i \in I), (\mathbf{s}_e \in \mathcal{S}_e,\ e \in E))$. A global configuration is said to be *valid* if it satisfies all the local constraints. Thus, $((x_i \in \mathbb{F},\ i \in I), (\mathbf{s}_e \in \mathcal{S}_e,\ e \in E))$ is a valid global configuration if for each $v \in V$, $((x_i,\ i \in \omega^{-1}(v)), (\mathbf{s}_e,\ e \in E(v))) \in C_v$. The set of all valid global configurations of a graphical model is called the *full behavior* of the model.

Note that the full behavior is a subspace $\mathfrak{B} \subseteq \mathbb{F}^I \oplus \left(\bigoplus_{e \in E} \mathcal{S}_e\right)$. As usual, for $J \subseteq I$, $\mathfrak{B}|_J$ denotes the projection of $\mathfrak{B}$ onto the index set J. For future convenience, we also define certain other projections of $\mathfrak{B}$. Let $\mathbf{b} = ((x_i,\ i \in I),\ (\mathbf{s}_e,\ e \in E))$ be a global configuration in $\mathfrak{B}$. At any given $v \in V$, the *local configuration* of $\mathbf{b}$ at v is defined as

$$\mathbf{b}|_v = ((x_i,\ i \in \omega^{-1}(v)),\ (\mathbf{s}_e,\ e \in E(v))).$$

The set of all local configurations of $\mathfrak{B}$ at v is then defined as $\mathfrak{B}|_v = \{\mathbf{b}|_v :\ \mathbf{b} \in \mathfrak{B}\}$. By definition, $\mathfrak{B}|_v \subseteq C_v$. Similarly, given any $e \in E$, if $\mathbf{b}$ is a global configuration as above, then we define $\mathbf{b}|_e = \mathbf{s}_e$; we further define $\mathfrak{B}|_e = \{\mathbf{b}|_e :\ \mathbf{b} \in \mathfrak{B}\}$. Clearly, $\mathfrak{B}|_e$ is a subspace of $\mathcal{S}_e$.

A graphical model $(\mathcal{G}, \omega, (\mathcal{S}_e,\ e \in E), (C_v,\ v \in V))$ is defined to be *essential* if $\mathfrak{B}|_v = C_v$ for all $v \in V$ and $\mathfrak{B}|_e = \mathcal{S}_e$ for all $e \in E$. When $\mathcal{G}$ is a tree, there is some redundancy in the above definition, as the condition $\mathfrak{B}|_e = \mathcal{S}_e$ for all $e \in E$ actually implies that $\mathfrak{B}|_v = C_v$ for all $v \in V$ [11, Lemma 2.2]. It is worth noting that, in an essential graphical model, at any edge $e = \{u, v\}$, the state space $\mathcal{S}_e$, may be viewed as a projection of each of the local constraint codes C_u and C_v. Thus, for any $e \in E$, $\dim(\mathcal{S}_e) \leq \dim(C_v)$, where v is any vertex incident with e.

An arbitrary graphical model Γ with full behavior $\mathfrak{B}$ can always be "essentialized" by simply replacing each local constraint C_v in Γ with the projection $\mathfrak{B}|_v$, and replacing each state space $\mathcal{S}_e$ with the projection $\mathfrak{B}|_e$. The resulting "essentialization" of Γ still has full behavior $\mathfrak{B}$.

An essential graphical model $(\mathcal{G}, \omega, (\mathcal{S}_e,\ e \in E), (C_v,\ v \in V))$ is defined to be a *graphical realization* of a code $\mathcal{C}$, or simply a *realization of $\mathcal{C}$ on $\mathcal{G}$*, if $\mathfrak{B}|_I = \mathcal{C}$. A graphical realization of $\mathcal{C}$ in which the underlying graph $\mathcal{G}$ is a tree, is called a *tree realization* of $\mathcal{C}$. Our definition of a graphical (and tree) realization differs slightly from the prior definitions in [6],[7],[8],[11], in that we require the underlying graphical model to be essential. As explained above, any graph model can be essentialized, so there is no loss of generality in this definition.

A graphical (resp. tree) realization $(\mathcal{G}, \omega, (\mathcal{S}_e,\ e \in E), (C_v,\ v \in V))$ of $\mathcal{C}$ is said to *extend*, or be an extension of, the graph (resp. tree) decomposition $(\mathcal{G}, \omega)$ of $\mathcal{C}$. We will denote by $\mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)$ the set of all graphical realizations of $\mathcal{C}$ that extend the graph decomposition $(\mathcal{G}, \omega)$ of $\mathcal{C}$.

Now, it is an easily verifiable fact that any tree decomposition of a code can always be extended to a tree realization of the code [7],[11]. Such an extension is not unique in general, but we will describe a canonical "minimal" extension a little later. More generally, any graph decomposition of a code $\mathcal{C}$ can always be extended to a graphical realization of $\mathcal{C}$, as we now explain. Let $\mathcal{G} =$

[1] In referring to a tuple of the form $(\mathcal{G}, \omega, (\mathcal{S}_e,\ e \in E), (C_v,\ v \in V))$, we will implicitly assume that V and E denote the vertex and edge sets, respectively, of the graph $\mathcal{G}$.

(V, E) be a connected graph, and suppose that $(\mathcal{G}, \omega)$ is a graph decomposition of $\mathcal{C}$. Take T to be any spanning tree of $\mathcal{G}$, and let E_T denote its edge set. Then, (T, ω) is a tree decomposition of $\mathcal{C}$. As noted above, this tree decomposition can be extended to a tree realization $(T, \omega, (\mathcal{S}_e,\ e \in E_T), (C_v,\ v \in V))$ of $\mathcal{C}$. We further extend this to a realization of $\mathcal{C}$ on $\mathcal{G}$ as follows. Define the state spaces $\overline{\mathcal{S}}_e$, $e \in E$, as

$$\overline{\mathcal{S}}_e = \begin{cases} \mathcal{S}_e & \text{if } e \in E_T \\ \{0\} & \text{if } e \in E \setminus E_T, \end{cases}$$

and for each $v \in V$, define the local constraint $\overline{C}_v = C_v \oplus (\bigoplus_{e \in E(v) \setminus E_T} \{0\})$. It should be clear that $(\mathcal{G}, \omega, (\overline{\mathcal{S}}_e,\ e \in E),\ (\overline{C}_v,\ v \in V))$ is a graphical realization of $\mathcal{C}$.

Graphical realizations of codes in which the underlying graph is a path or a simple cycle have received considerable prior attention in the literature. Such realizations were called "conventional state realizations" (when the underlying graph is a path) and "tail-biting state realizations" (when the underlying graph is a simple cycle) in [6]. We will call them "trellis realizations". Briefly, a *trellis realization* of a code $\mathcal{C}$ (defined on the index set I) is any extension of a graph decomposition of $\mathcal{C}$ of the form $(\mathcal{G}, \omega)$, where $\mathcal{G}$ is either a path or a simple cycle, and ω is a surjective map $\omega : I \to V(\mathcal{G})$. A trellis realization in which the surjective map $\omega : I \to V(\mathcal{G})$ is not injective (so that ω is not a bijection), is usually called a *sectionalized* trellis realization. A *conventional* trellis realization is one in which the underlying graph $\mathcal{G}$ is a path. When the underlying graph $\mathcal{G}$ is a simple cycle, the trellis realization is said to be *tailbiting*. The theory of conventional trellis realizations is well established; see, for example, [17]. On the other hand, tailbiting trellis realizations are less well understood; the principal systematic study of these remains that of Koetter and Vardy [13],[14]. We remark that our requirement that graphical realizations have underlying graph models that are essential corresponds to the requirement in [13],[14] that "linear trellises" be "reduced".

3. Complexity Measures for Graphical Realizations

As observed in [6], any graphical realization of a code specifies a class of associated graph-based decoding algorithms, namely, the sum-product algorithm and its variants. Thus, ideally, any definition of a complexity measure for a graphical realization should try to capture the computational complexity of the associated decoding algorithms. The analysis in [6, Section V] shows that the computational complexity of the sum-product algorithm specified by a given graphical realization of a code is determined in large part by the cardinalities, or equivalently dimensions, of the local constraint codes in the realization. Thus, as a simple measure of the complexity of a graphical realization, which roughly reflects the complexity of sum-product decoding, we will consider the maximum of the dimensions of the local constraint codes in the realization.

Let $\Gamma = (\mathcal{G}, \omega, (C_v, v \in V), (\mathcal{S}_e, e \in E))$ be a graphical realization of a code $\mathcal{C}$. The *constraint max-complexity*, or simply *κ-complexity*, of Γ is defined to be $\kappa(\Gamma) = \max_{v \in V} \dim(C_v)$. Now, recall that if $(\mathcal{G}, \omega)$ is a graph decomposition of $\mathcal{C}$, then $\mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)$ denotes the set of all graphical realizations of $\mathcal{C}$ that extend $(\mathcal{G}, \omega)$. We further define

$$\kappa(\mathcal{C}; \mathcal{G}, \omega) = \min_{\Gamma \in \mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)} \kappa(\Gamma) \tag{1}$$

We add another level of minimization by defining, for a given graph $\mathcal{G}$, the *$\mathcal{G}$-width* of a code $\mathcal{C}$ to be

$$\kappa(\mathcal{C}; \mathcal{G}) = \min_{\omega} \kappa(\mathcal{C}; \mathcal{G}, \omega), \tag{2}$$

where the minimum is taken over all possible index mappings $\omega : I \to V(\mathcal{G})$, where I is the index set of $\mathcal{C}$. Thus, the $\mathcal{G}$-width of $\mathcal{C}$ is the least κ-complexity of any realization of $\mathcal{C}$ on $\mathcal{G}$, and may be taken to be a measure of the least computational complexity of any sum-product-type decoding algorithm for $\mathcal{C}$ implemented on the graph $\mathcal{G}$.

A broader optimization problem of considerable interest is the following: given a code $\mathcal{C}$ and a family of graphs $\mathfrak{G}$, identify a $\mathcal{G} \in \mathfrak{G}$ on which $\mathcal{C}$ can be realized with the least possible κ-complexity. We thus define

$$\kappa(\mathcal{C}; \mathfrak{G}) = \min_{\mathcal{G} \in \mathfrak{G}} \kappa(\mathcal{C}; \mathcal{G}). \tag{3}$$

Two special cases of this definition — treewidth and pathwidth (or trellis-width) — are particularly of interest. We define treewidth first, and pathwidth a little further below.

If we let $\mathfrak{T}$ denote the set of all trees, then $\kappa(\mathcal{C}; \mathfrak{T})$ is called the *treewidth* of the code $\mathcal{C}$ [11], which we will denote by $\kappa_{\text{tree}}(\mathcal{C})$. The notion of treewidth (*i.e.*, minimal κ-complexity among tree realizations) of a code was first considered by Forney [7], and an analogous notion has been defined for matroids in [9]. The arguments in [7, Section V] (and also in [9]) show that $\kappa_{\text{tree}}(\mathcal{C})$ can always be obtained by minimizing $\kappa(\mathcal{C}; T, \omega)$ over tree decompositions (T, ω) in which T is a cubic tree (*i.e.*, a tree in which all internal nodes have degree 3), and ω is a bijection between the index set of $\mathcal{C}$ and the set of leaves of T.

A complexity measure related to treewidth, termed minimal tree complexity, was defined and studied by Halford and Chugg [8]. Treewidth, as we have defined above, is an upper bound on the minimal tree complexity of Halford and Chugg.

The *pathwidth*, $\kappa_{\text{path}}(\mathcal{C})$, of a code $\mathcal{C}$ is defined to be the quantity $\kappa(\mathcal{C}; \mathfrak{P})$, where $\mathfrak{P}$ denotes the sub-family of $\mathfrak{T}$ consisting of all paths. We will find it convenient to refer to tree decompositions (P, ω), with $P \in \mathfrak{P}$, as *path decompositions*. Thus, $\kappa(\mathcal{C}; \mathfrak{P})$ is the minimum value of $\kappa(\mathcal{C}; P, \omega)$ as (P, ω) ranges over all path decompositions of $\mathcal{C}$. In fact, by the argument of [7, Section V.B], the minimizing path decomposition (P, ω) may be taken to be one in which the index mapping ω is surjective. Thus, $\kappa_{\text{path}}(\mathcal{C})$ is the least κ-complexity of any conventional trellis realization of $\mathcal{C}$, and so we may also call it the *(conventional) trellis-width*[2] of $\mathcal{C}$. It is also known that sectionalization cannot reduce the κ-complexity[3] of a trellis realization [17, Theorem 6.3], and hence, $\kappa_{\text{path}}(\mathcal{C})$ is the minimum value of $\kappa(\mathcal{C}; P, \omega)$ over all path decompositions (P, ω) in which the index mapping ω is a bijection between the index set of $\mathcal{C}$ and the vertices of P.

Measures of constraint complexity other than κ-complexity have been proposed in the previous literature, especially in the context of trellis realizations [17],[14]. The *κ^+-complexity* of a graphical realization $\Gamma = (\mathcal{G}, \omega, (C_v, v \in V), (\mathcal{S}_e, e \in E))$ is defined to be $\kappa^+(\Gamma) = \sum_{v \in V} \dim(C_v)$. Note that $\frac{1}{|V|}\, \kappa^+(\Gamma)$ is the average local constraint code dimension in Γ. On the other hand, it has been suggested [7] that the sum of the constraint code cardinalities "may be a better guide to decoding complexity" than κ-complexity or κ^+-complexity. Thus, we define $\kappa^{\text{tot}}(\Gamma) = \sum_{v \in V} |C_v| = \sum_{v \in V} |\mathbb{F}|^{\dim(C_v)}$. Analogous to (1)–(3), we may define $\kappa^+(\mathcal{C}, \mathcal{G}, \omega)$, $\kappa^{\text{tot}}(\mathcal{C}; \mathcal{G}, \omega)$, etc., but we will only touch upon these briefly in this paper.

We remark that while we have used constraint code dimensions to define our complexity measures for graphical realizations, one could also define measures of complexity based on state-space dimensions. For example, we could define the state max-complexity, $\sigma(\Gamma)$, of a graphical realization Γ to be the maximum of the dimensions of the state spaces in Γ. Similarly, we may consider the complexity measures $\sigma^+(\Gamma)$ and $\sigma^{\text{tot}}(\Gamma)$ analogous to $\kappa^+(\Gamma)$ and $\kappa^{\text{tot}}(\Gamma)$. These measures are especially relevant and have been well studied in the context of trellis realizations; again, see [17],[14]. However, as noted by Forney [7], measures of state-space complexity become less appropriate in the context of realizations on arbitrary graphs or trees. For example, for any code $\mathcal{C} \subseteq \mathbb{F}^n$, one can always find a tree on which $\mathcal{C}$ can be realized in such a way that all state-spaces have dimension at most 1. This would be the "star-shaped" tree T consisting of n leaves connected to a single internal node v of degree n. Take ω to be any bijection between the index set of $\mathcal{C}$ and the leaves of T; set $\mathcal{S}_e = \mathbb{F}$ at each edge e of T; and finally, take the local constraint code C_v at the internal node to be $\mathcal{C}$ itself, and take the local constraint codes at the leaves to be $[2, 1]$ repetition codes. Clearly, the

[2]For this reason, what we have called $\kappa_{\text{path}}(\mathcal{C})$ here was called $\kappa_{\text{trellis}}(\mathcal{C})$ in [11].

[3]Our notion of κ-complexity corresponds to the notion of "edge-complexity" in [17].

resulting tree model (after essentialization) is a tree realization of $\mathcal{C}$. Thus, it makes little sense to define a state-space complexity measure analogous to treewidth, unless we restrict the kind of trees on which we are allowed to realize the given code[4].

The astute reader may point out that in the trivial tree realization above, the sum of the state-space dimensions is non-trivial, and so a state-space analogue to treewidth could potentially be defined in terms of σ^+ or σ^{tot}. This may be true, but we do not pursue this further, since, as already observed previously, complexity measures based on constraint code dimensions are a better guide to decoding complexity. But, while on this topic, we mention in passing that for any *tree* realization Γ of a code $\mathcal{C}$, it turns out that

$$\kappa^+(\Gamma) = \dim(\mathcal{C}) + \sigma^+(\Gamma). \tag{4}$$

Thus, the problem of minimizing $\sigma^+(\Gamma)$ among tree realizations Γ of a given code $\mathcal{C}$ is equivalent to the problem of minimizing $\kappa^+(\Gamma)$. The identity in (4), which may be viewed as a generalization of the statement of Theorem 4.6 in [14] for conventional trellis realizations, will not be proved here as it would be an unnecessary deviation from the main line of our development. It suffices to say that (4) follows from Theorem 3.4 in [11] by first verifying that it indeed holds for any minimal tree realization $\mathcal{M}(\mathcal{C};T,\omega)$, and then observing that the difference between $\kappa^+(\Gamma)$ and $\sigma^+(\Gamma)$ is preserved by the state-merging process mentioned in the statement of that theorem.

Finally, we remark that the state max-complexity of a graphical realization cannot exceed the constraint max-complexity of the realization. This is because, as observed in Section 2.3, in any essential graphical model $(\mathcal{G},\omega,(\mathcal{S}_e, e \in E),(C_v, v \in V))$, for each edge $e \in E$, we have $\dim(\mathcal{S}_e) \le \dim(C_v)$, where v is any vertex incident with e.

3.1. Minimal Realizations.

Given a code $\mathcal{C}$ and a tree decomposition (T,ω) of $\mathcal{C}$, there exists a tree realization, $(T,\omega,(\mathcal{S}_e^*,\ e \in E),(C_v^*,\ v \in V))$, of $\mathcal{C}$ with the following property [6],[7]:

> if $(T,\omega,(\mathcal{S}_e,\ e \in E),(C_v,\ v \in V))$ is a tree realization of $\mathcal{C}$ that extends (T,ω),
> then for all $e \in E$, $\dim(\mathcal{S}_e^*) \le \dim(\mathcal{S}_e)$.

This *minimal* tree realization, which we henceforth denote by $\mathcal{M}(\mathcal{C};T,\omega)$, is unique up to isomorphism[5]. Constructions of $\mathcal{M}(\mathcal{C};T,\omega)$ can be found in [6],[7],[11].

It has further been shown [11] that not only does $\mathcal{M}(\mathcal{C};T,\omega)$ minimize (among realizations in $\mathfrak{R}(\mathcal{C};T,\omega)$) the state space dimension at each edge of T, but it also minimizes the local constraint code dimension at each vertex of T. More precisely, $\mathcal{M}(\mathcal{C};T,\omega)$ also has the following property:

> if $(T,\omega,(\mathcal{S}_e,\ e \in E),(C_v,\ v \in V))$ is a tree realization of $\mathcal{C}$ that extends (T,ω),
> then for all $v \in V$, $\dim(C_v^*) \le \dim(C_v)$.

Consequently, we have that $\kappa(\mathcal{C};T,\omega) = \kappa(\mathcal{M}(\mathcal{C};T,\omega))$, $\kappa^+(\mathcal{C};T,\omega) = \kappa^+(\mathcal{M}(\mathcal{C};T,\omega))$, and $\kappa^{\text{tot}}(\mathcal{C};T,\omega) = \kappa^{\text{tot}}(\mathcal{M}(\mathcal{C};T,\omega))$. The fact that $\mathcal{M}(\mathcal{C};T,\omega)$ minimizes local constraint code dimension at each vertex of T will be central to the derivation of our results in the sections to follow.

We will henceforth consistently use the notation $\mathcal{S}_e^*$ and C_v^* to denote state spaces and local constraint codes in a minimal tree realization $\mathcal{M}(\mathcal{C};T,\omega)$. Exact expressions for the dimensions of $\mathcal{S}_e^*$ and C_v^* in $\mathcal{M}(\mathcal{C};T,\omega)$ are known [6],[7]. Recall that for an edge e of T, we denote by $T^{(e)}$ and $\overline{T}^{(e)}$ the two components of $T \setminus e$. Let us further define $J(e) = \omega^{-1}(V(T^{(e)}))$ and $\overline{J}(e) = \omega^{-1}(V(\overline{T}^{(e)}))$. We then have

$$\dim(\mathcal{S}_e^*) = \dim(\mathcal{C}) - \dim(\mathcal{C}_{J(e)}) - \dim(\mathcal{C}_{\overline{J}(e)}). \tag{5}$$

[4]We do get a reasonable state-space analogue to treewidth if we restrict the class of trees over which we attempt to minimize state-space complexity to the class of cubic trees only; see [11].

[5]Graphical realizations $(\mathcal{G},\omega,(C_v, v \in V),(\mathcal{S}_e, e \in E))$ and $(\mathcal{G},\omega,(C_v', v \in V),(\mathcal{S}_e', e \in E))$ of a code $\mathcal{C}$ are said to be *isomorphic* if, for each $v \in V$, C_v and C_v' are isomorphic as vector spaces, and for each $e \in E$, $\mathcal{S}_e$ and $\mathcal{S}_e'$ are isomorphic as vector spaces. We do not distinguish between isomorphic graphical realizations.

Next, consider any vertex v in T. If v has degree δ, then $T-v$ has components $T_i^{(v)}$, $i = 1, 2, \ldots, \delta$. Define $J_i = \omega^{-1}(V(T_i^{(v)}))$, for $i = 1, 2, \ldots, \delta$. Then [7, Theorem 1],

$$\dim(C_v^*) = \dim(\mathcal{C}) - \sum_{i=1}^{\delta} \dim(\mathcal{C}_{J_i}). \tag{6}$$

In summary, the minimal tree realization $\mathcal{M}(\mathcal{C}; T, \omega)$ is an exact solution to the problem of determining the minimum-complexity extension of a tree decomposition (T, ω) of a code $\mathcal{C}$. Moreover, $\mathcal{M}(\mathcal{C}; T, \omega)$ minimizes, among realizations in $\mathfrak{R}(\mathcal{C}; T, \omega)$, any reasonable measure of complexity, be it state-space complexity or constraint complexity. Unfortunately, when we move to realizations on graphs with cycles, there appear to be no "canonical" minimal realizations with properties similar to those of minimal tree realizations. In fact, if $(\mathcal{G}, \omega)$ is a graph decomposition of a code $\mathcal{C}$, where $\mathcal{G}$ is a graph with cycles, there need not even be a realization Γ that simultaneously achieves $\min_{\Gamma \in \mathfrak{R}(\mathcal{C};\mathcal{G},\omega)} \kappa(\Gamma)$ and $\min_{\Gamma \in \mathfrak{R}(\mathcal{C};\mathcal{G},\omega)} \kappa^+(\Gamma)$. An example of such a graph decomposition is given in Appendix A.

Thus, given a code $\mathcal{C}$ and a graph $\mathcal{G}$ containing cycles, the problem of finding realizations of $\mathcal{C}$ on $\mathcal{G}$ with the least possible κ-complexity, κ^+-complexity, or κ^{tot}-complexity (within some interesting sub-class of realizations of $\mathcal{C}$ on $\mathcal{G}$) is much harder to solve than the corresponding problem for cycle-free graphs. In the next section, we present a simple but valuable tool that will enable us to derive non-trivial lower bounds on the constraint complexity of realizations of a code on an arbitrary graph. These bounds could be used, for example, to determine whether or not the complexity of a given realization is close to the least possible.

4. Cut-Set Bounds

Let $\mathcal{G} = (V, E)$ be a connected graph. A partition (V', V'') of V is said to be *separated* by an edge cut $X \subseteq E$ if, for each pair of vertices $v' \in V'$ and $v'' \in V''$, any path in $\mathcal{G}$ that joins v' to v'' passes through some edge $e \in X$. We remark that if $\mathcal{G} \setminus X$ has more than two components, then there is more than one partition of V that is separated by X.

The Edge-Cut Bound, stated below, is a result of fundamental importance in the study of graphical realizations. This result was originally observed by Wiberg, Loeliger and Koetter [18],[19], but the version we give here is due to Forney [6, Corollary 4.4].

Theorem 4.1 (The Edge-Cut Bound). *Let $\Gamma = (\mathcal{G}, \omega, (C_v, v \in V), (\mathcal{S}_e, e \in E))$ be a realization of a code $\mathcal{C}$ on a connected graph $\mathcal{G} = (V, E)$. If (V', V'') is a partition of V separated by an edge cut $X \subseteq E$, then, defining $J' = \omega^{-1}(V')$ and $J'' = \omega^{-1}(V'')$, we have*

$$\sum_{e \in X} \dim(\mathcal{S}_e) \geq \dim(\mathcal{C}) - \dim(\mathcal{C}_{J'}) - \dim(\mathcal{C}_{J''}).$$

The edge-cut bound can be used to derive useful lower bounds on the state-space complexity of a graphical realization; see, for example, [5]. To deal with constraint complexity, however, we will need a closely-related bound that uses vertex cuts instead of edge cuts.

We introduce here some terminology that we will use to state our vertex-cut bound. For $v \in V$, let $N(v) = \{u \in V : \{u, v\} \in E\}$ denote the set of neighbours of v in $\mathcal{G}$. Furthermore, for $W \subseteq V$, define $N(W) = \bigcup_{v \in W} N(v)$.

Definition 4.1. *An ordered collection $(V_0, V_1, \ldots, V_\delta)$, $\delta \geq 0$, of subsets of V is said to be a* star partition *of V, if the V_i's form a partition of V (*i.e.*, the V_i's are pairwise disjoint, and $\bigcup_{i=0}^{\delta} V_i = V$), and for each $i \in \{1, 2, \ldots, \delta\}$, we have $N(V_i) \subseteq V_i \cup V_0$.*

The definition has been worded so as to allow some of the V_i's to be empty sets. When V_i is non-empty for at most one $i \geq 1$, a star partition is simply a partition. When at least two V_i's

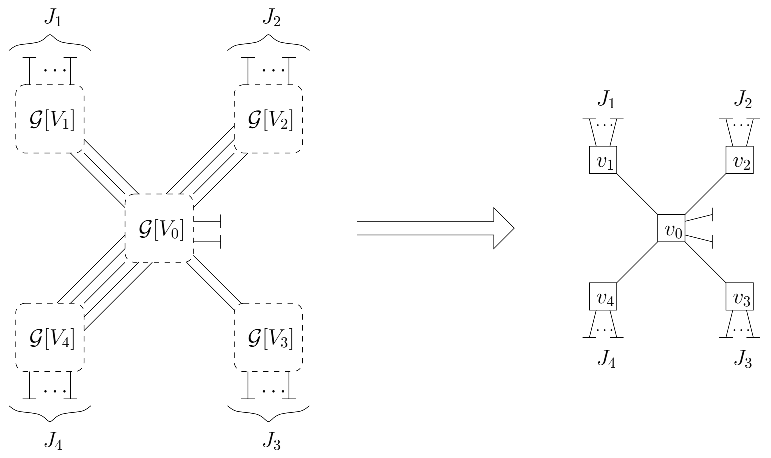


FIGURE 2. A depiction of the construction in the proof of the Vertex-Cut Bound. $\mathcal{G}[V_i]$ denotes the subgraph of $\mathcal{G}$ induced by the vertices in V_i.

other than V_0 are non-empty, then a star partition is a partition that arises from a vertex cut of $\mathcal{G}$, as we now explain. For any $i > j \geq 1$, if V_i and V_j are both non-empty, then the above definition simply says that any path between a vertex in V_i and a vertex in V_j must pass through V_0. Thus, if at least two V_i's other than V_0 are non-empty, then V_0 is a vertex cut of $\mathcal{G}$. Conversely, if V_0 is a vertex cut of $\mathcal{G}$, and $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_\delta$, $\delta \geq 2$, are the (non-empty) components of $\mathcal{G} - V_0$, then, setting $V_i = V(\mathcal{G}_i)$ for $i = 1, 2, \ldots, \delta$, we see that $(V_0, V_1, \ldots, V_\delta)$ is a star partition of V. The graph on the left in Figure 2, which depicts a typical situation covered by the definition, should also explain the nomenclature.

Theorem 4.2 (The Vertex-Cut Bound). *Let $\Gamma = (\mathcal{G}, \omega, (C_v, v \in V), (\mathcal{S}_e, e \in E))$ be a realization of a code $\mathcal{C}$ on a connected graph $\mathcal{G} = (V, E)$. If $(V_0, V_1, \ldots, V_\delta)$ is a star partition of V, then, defining $J_i = \omega^{-1}(V_i)$ for $i = 1, 2, \ldots, \delta$, we have*

$$\sum_{v \in V_0} \dim(C_v) \geq \dim(\mathcal{C}) - \sum_{i=1}^{\delta} \dim(\mathcal{C}_{J_i}).$$

Proof. Let $\mathfrak{B}$ denote the full behaviour of Γ. If $\mathbf{b} = ((x_i, i \in I), (\mathbf{s}_e, e \in E))$ is a global configuration in $\mathfrak{B}$, then given an $X \subseteq E$, we will use $\mathbf{b}|_X$ to denote the projection $(\mathbf{s}_e, e \in X)$. We further set $\mathfrak{B}|_X = \{\mathbf{b}|_X : \mathbf{b} \in \mathfrak{B}\}$.

For $i = 1, 2, \ldots, \delta$, let X_i be the set of edges of $\mathcal{G}$ with exactly one end-point in V_i, so that the other end-point is necessarily in V_0. We then define

$$\mathfrak{B}|_{V_i} = \{(\mathbf{b}|_{J_i}, \mathbf{b}|_{X_i}) : \ \mathbf{b} \in \mathfrak{B}\},$$

for $i = 1, 2, \ldots, \delta$. Furthermore, set $J_0 = \omega^{-1}(V_0)$ and $X_0 = \bigcup_{i=1}^{\delta} X_i$, and define

$$\mathfrak{B}|_{V_0} = \{(\mathbf{b}|_{J_0}, \mathbf{b}|_{X_0}) : \ \mathbf{b} \in \mathfrak{B}\}.$$

Now, consider the "star-shaped" tree $T = (V_T, E_T)$ consisting of a single internal vertex v_0 of degree δ, whose neighbours $v_1, v_2, \ldots, v_\delta$ are all the leaves of T. Thus, $V_T = \{v_0, v_1, \ldots, v_\delta\}$ and $E_T = \{\{v_0, v_i\} : \ i = 1, 2, \ldots, \delta\}$. Define the index mapping $\alpha : I \to V_T$ as follows: $\alpha(j) = v_i$ iff $\omega(j) \in V_i$. Note that, for $i = 0, 1, 2, \ldots, \delta$, we have $\alpha^{-1}(V_i) = \omega^{-1}(V_i) = J_i$. The construction of the tree decomposition (T, α) from $(\mathcal{G}, \omega)$ is depicted in Figure 2.

We next extend the tree decomposition (T,α) to a tree model $\widehat{\Gamma} = ((T,\alpha,(\widehat{\mathcal{S}}_e, e \in E_T),(\widehat{C}_v, v \in V_T))$ by setting $\widehat{C}_{v_i} = \mathfrak{B}|_{V_i}$ for $i = 0,1,2,\ldots,\delta$, and $\widehat{\mathcal{S}}_{\{v_0,v_i\}} = \mathfrak{B}|_{X_i}$ for $i = 1,2,\ldots,\delta$. From the fact that Γ is a realization of $\mathcal{C}$, it readily follows that $\widehat{\Gamma}$ is a tree realization of $\mathcal{C}$.

Recalling that the minimal tree realization $\mathcal{M}(\mathcal{C};T,\alpha)$ minimizes the local constraint code dimension at each vertex of T, we obtain via (6),

$$\dim(\widehat{C}_{v_0}) \ge \dim(\mathcal{C}) - \sum_{i=1}^{\delta} \dim(\mathcal{C}_{J_i}).$$

We complete the proof by observing that

$$\dim(\widehat{C}_{v_0}) = \dim(\mathfrak{B}|_{V_0}) \le \sum_{v \in V_0} \dim(\mathfrak{B}|_v) = \sum_{v \in V_0} \dim(C_v).$$

□

The following useful corollary is an immediate consequence of the Vertex-Cut Bound.

Corollary 4.3. *Let $(\mathcal{G},\omega)$ be a graph decomposition of a code $\mathcal{C}$, where $\mathcal{G}$ is a connected graph. For a vertex cut W of $\mathcal{G}$, if $\mathcal{G}_1,\mathcal{G}_2,\ldots,\mathcal{G}_\delta$ are the components of $\mathcal{G} - W$, then define $\lambda(W) = \dim(\mathcal{C}) - \sum_{i=1}^{\delta}\dim(\mathcal{C}_{J_i})$, where $J_i = \omega^{-1}(V(\mathcal{G}_i))$ for $i = 1,2,\ldots,\delta$. Then, for any realization $\Gamma = (\mathcal{G},\omega,(C_v, v\in V),(\mathcal{S}_e, e \in E))$ of $\mathcal{C}$ that extends $(\mathcal{G},\omega)$, we have*

$$\sum_{v \in W} \dim(C_v) \ge \lambda(W).$$

In its most straightforward application, the Vertex-Cut Bound, via the above corollary, can be used in conjunction with constrained optimization techniques to find lower bounds on $\kappa(\mathcal{C};\mathcal{G},\omega)$, $\kappa^+(\mathcal{C};\mathcal{G},\omega)$ and $\kappa^{\text{tot}}(\mathcal{C};\mathcal{G},\omega)$. Indeed, if $(\mathcal{G},\omega)$ is a graph decomposition of $\mathcal{C}$, and $W_1,W_2,\ldots,W_t$ are vertex cuts of $\mathcal{G}$, then, by Corollary 4.3, the dimensions of the local constraint codes C_v, $v \in V$, in any $\Gamma \in \mathfrak{R}(\mathcal{C};\mathcal{G},\omega)$ must satisfy $\sum_{v\in W_i}\dim(C_v) \ge \lambda(W_i)$, $i = 1,2,\ldots,t$. Thus, for example, $\kappa^+(\mathcal{C};\mathcal{G},\omega)$ is lower bounded by the solution to the following linear programming problem in the variables ξ_v, $v \in V$: given a collection of vertex cuts $W_1,W_2,\ldots,W_t$ of $\mathcal{G}$,

$$\text{minimize } \sum_{v\in V}\xi_v, \quad \text{subject to } \sum_{w \in W_i}\xi_w \ge \lambda(W_i),\ i = 1,2,\ldots,t.$$

However, we do not pursue this angle any further in this paper. Instead, we will henceforth restrict our attention to the κ-complexity measure, for which we will derive a suite of lower bounds, again based on the Vertex-Cut Bound, which unearth some interesting connections with graph theory, and moreover, are amenable to further mathematical analysis. The bounds we derive rely on the notion of vertex-cut trees introduced in the next section.

5. Vertex-Cut Trees

We begin with a simple lemma, which plays a role in our definition of a vertex-cut tree below.

Lemma 5.1. *Suppose that $V_0,V_1,\ldots,V_\delta$ are subsets of V such that $\bigcup_{i=0}^{\delta} V_i = V$. If, for each pair of distinct indices i,j, we have $V_i \cap V_j \subseteq V_0$, then $(V_0, V_1 \setminus V_0,\ldots,V_\delta \setminus V_0)$ is a partition of V.*

Proof. It is evident that $V_0 \cup \bigcup_{i=1}^{\delta}(V_i \setminus V_0) = \bigcup_{i=0}^{\delta} V_i = V$. If $i,j > 0$, $i \ne j$, then $(V_i \setminus V_0) \cap (V_j \setminus V_0) = (V_i \cap V_j) \setminus V_0 = \emptyset$, since $V_i \cap V_j \subseteq V_0$. □

For the main definition of this section, we introduce some convenient notation, which will henceforth be used consistently. If A and B are sets, and $f : A \to 2^B$ is a mapping from A to the power set of B, then for any $X \subseteq A$, we define $f(X) = \bigcup_{x\in X} f(x)$. Also, recall that if z is a vertex of degree δ in a tree T, then the components of $T - z$ are denoted by $T_i^{(z)}$, $i = 1,2,\ldots,\delta$.

Definition 5.1. *Let $\mathcal{G}$ be a connected graph. A* vertex-cut tree *of $\mathcal{G}$ is a data structure (T, β), where T is a tree, and $\beta : V(T) \to 2^{V(\mathcal{G})}$ is a mapping with the following properties:*

(VC1) *$\beta(V(T)) = V(\mathcal{G})$;*

(VC2) *for each pair $x, y \in V(T)$, if $z \in V(T)$ is any vertex that lies on the unique path between x and y in T, then $\beta(x) \cap \beta(y) \subseteq \beta(z)$;*

(VC3) *for each $z \in V(T)$, $(\beta(z), V_1, V_2, \ldots, V_\delta)$ is a star partition of $V(\mathcal{G})$, where δ is the degree of z, and $V_i = \beta(V(T_i^{(z)})) \setminus \beta(z)$ for $i = 1, 2, \ldots, \delta$.*

A vertex-cut tree (T, β) in which T is a path is called a vertex-cut path.

Note that, by Lemma 5.1, conditions (VC1) and (VC2) in the above definition imply that, for each $z \in V(T)$, with δ and V_i, $i = 1, 2, \ldots, \delta$, as in condition (VC3), $(\beta(z), V_1, V_2, \ldots, V_\delta)$ is a partition of $V(\mathcal{G})$. Thus, for (T, β) satisfying conditions (VC1) and (VC2), condition (VC3) is met iff, for each $z \in V(T)$, we have $N(V_i) \subseteq V_i \cup \beta(z)$ for all $i \geq 1$.

Trivial vertex-cut trees (and paths) always exist for a graph $\mathcal{G}$ — given any tree T, pick a vertex $z_0 \in V(T)$, and define a mapping β by setting $\beta(z_0) = V(\mathcal{G})$, and $\beta(z) = \emptyset$ for $z \neq z_0$. This allows us to make the following definition.

Definition 5.2. *Let $\mathcal{G}$ be a connected graph. The* vc-width *of a vertex-cut tree (T, β) of $\mathcal{G}$ is defined as $\max_{z \in V(T)} |\beta(z)|$, and is denoted by vc-width(T, β). The* vc-treewidth *(resp.* vc-pathwidth*) of $\mathcal{G}$ is the least vc-width among all vertex-cut trees (resp. vertex-cut paths) of $\mathcal{G}$, and is denoted by $\kappa_{\text{vc-tree}}(\mathcal{G})$ (resp. $\kappa_{\text{vc-path}}(\mathcal{G})$).*

Thus, for any graph $\mathcal{G}$, we have $0 < \kappa_{\text{vc-tree}}(\mathcal{G}) \leq \kappa_{\text{vc-path}}(\mathcal{G}) \leq |V(\mathcal{G})|$. The vc-treewidth of any tree is equal to one. Indeed, if T is a tree, then (T, β), defined by $\beta(z) = \{z\}$ for all $z \in V(T)$, is a vertex-cut tree of T, with vc-width equal to one.

Example 5.1. *Let $\mathcal{G}$ be an n-cycle with vertices $v_0, v_1, \ldots, v_{n-1}$, labeled in cyclic order. Let P be a path with $n-1$ vertices, which in the linear order defined by the path, are labeled $z_1, z_2, \ldots, z_{n-1}$. To be precise, z_0 is one of the two leaves, and for $i = 1, 2, \ldots, n-1$, z_i is adjacent to z_{i-1} in P. If we define the mapping $\beta : V(P) \to 2^{V(\mathcal{G})}$ as $\beta(z_i) = \{v_0, v_i\}$ for $i = 1, 2, \ldots, n-1$, then (P, β) is a vertex-cut path of $\mathcal{G}$, of vc-width two. It is not difficult to verify that $\mathcal{G}$ has no vertex-cut tree of vc-width one, and hence, $\kappa_{\text{vc-tree}}(\mathcal{G}) = \kappa_{\text{vc-path}}(\mathcal{G}) = 2$.*

Our definition of vertex-cut trees may appear at first to be an artificial construct brought in solely for the purpose of finding applications for the Vertex-Cut Bound. However, this is far from being the case. Vertex-cut trees are very closely related to junction trees, commonly associated with belief propagation algorithms in Bayesian networks [10] and coding theory [1]. Another close relative of vertex-cut trees is the data structure known as tree decomposition (of a graph) [16],[2],[3], which has received considerable attention in the graph theory and computer science literatures.

Definition 5.3. *A* tree decomposition *of a connected graph $\mathcal{G}$ is a data structure (T, β), where T is a tree, and $\beta : V(T) \to 2^{V(\mathcal{G})}$ is a mapping with the following properties:*

(T1) *$\beta(V(T)) = V(\mathcal{G})$;*

(T2) *for each pair $x, y \in V(T)$, if $z \in V(T)$ is any vertex that lies on the unique path between x and y in T, then $\beta(x) \cap \beta(y) \subseteq \beta(z)$;*

(T3) *for each pair of adjacent vertices $u, v \in V(\mathcal{G})$, there exists a $z \in V(T)$ such that $\{u, v\} \subseteq \beta(z)$.*

A tree decomposition (T, β) in which T is a path is called a path decomposition.

The *width* of a tree decomposition as above is defined to be $\max_{z \in V(T)} |\beta(z)| - 1$. The *treewidth* (resp. *pathwidth*) of a graph $\mathcal{G}$, denoted by $\kappa_{\text{tree}}(\mathcal{G})$ (resp. $\kappa_{\text{path}}(\mathcal{G})$), is the minimum among the widths of all its tree (resp. path) decompositions. Note that if $\mathcal{G}$ has at least one non-loop edge, then, because of (T3), any tree decomposition of $\mathcal{G}$ must have width at least one. Thus, for any

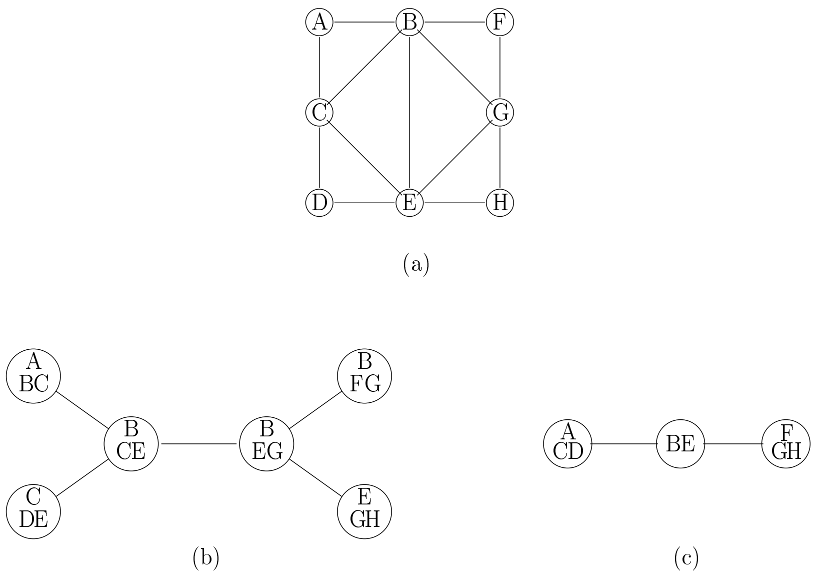


FIGURE 3. (a) A graph $\mathcal{G}$ on eight vertices. (b) A tree decomposition of $\mathcal{G}$ with width 2, and vc-width 3. (c) A vertex-cut path of $\mathcal{G}$ of vc-width 3 that is not a path decomposition of $\mathcal{G}$.

such graph $\mathcal{G}$, we have $0 < \kappa_{\text{tree}}(\mathcal{G}) \le \kappa_{\text{path}}(\mathcal{G}) \le |V(\mathcal{G})| - 1$. It is not hard to check that any tree with at least two vertices has treewidth equal to one[6], and that if $\mathcal{G}$ is an n-cycle, then $\kappa_{\text{tree}}(\mathcal{G}) = \kappa_{\text{path}}(\mathcal{G}) = 2$.

Lemma 5.2. *Any tree decomposition of a connected graph $\mathcal{G}$ is also a vertex-cut tree of $\mathcal{G}$. Hence, $\kappa_{\text{vc-tree}}(\mathcal{G}) \le \kappa_{\text{tree}}(\mathcal{G}) + 1$, and $\kappa_{\text{vc-path}}(\mathcal{G}) \le \kappa_{\text{path}}(\mathcal{G}) + 1$.*

Proof. Let (T, β) be a tree decomposition of $\mathcal{G}$. We only have to show that (T, β) satisfies condition (VC3) of Definition 5.1. Consider any $z \in V(T)$, with degree δ, and let V_i, $i = 1, \ldots, \delta$, be defined as in condition (VC3). By virtue of (T1), (T2) and Lemma 5.1, $(\beta(z), V_1, \ldots, V_\delta)$ is a partition of $V(\mathcal{G})$. Thus, we must show that $N(V_i) \subseteq V_i \cup \beta(z)$ for $i = 1, \ldots, \delta$.

Suppose, to the contrary, that there exists a $v \in V_i$ such that for some $u \in N(v)$, we have $u \notin V_i \cup \beta(z)$. Thus, by definition of V_i, we have $u \notin \beta(V(T_i^{(z)})) \cup \beta(z)$, while $v \in \beta(x) \setminus \beta(z)$ for some $x \in V(T_i^{(z)})$. Now, by condition (T3), we have $\{u, v\} \subseteq \beta(y)$ for some $y \in V(T)$. In particular, $v \in \beta(x) \cap \beta(y)$. On the other hand, by our assumption on u, we must have $y \neq z$ and $y \notin V(T_i^{(z)})$. Thus, $y \in V(T_j^{(z)})$ for some $j \neq i$. This means that the vertex z lies on the unique path in T joining x and y, and hence, by (T2), $v \in \beta(z)$, which contradicts the existence of v as postulated. $\square$

A graph $\mathcal{G}$ can have a vertex-cut tree that is not a tree decomposition. Figure 3 shows an example of a vertex-cut path of a graph $\mathcal{G}$ that is not a path decomposition of $\mathcal{G}$. Also, the inequality $\kappa_{\text{vc-tree}}(\mathcal{G}) \le \kappa_{\text{tree}}(\mathcal{G}) + 1$ in Lemma 5.2 can hold with equality — for the graph $\mathcal{G}$ in Figure 3(a), it is possible to show that $\kappa_{\text{tree}}(\mathcal{G}) = 2$, while $\kappa_{\text{vc-tree}}(\mathcal{G}) = \kappa_{\text{vc-path}}(\mathcal{G}) = 3$.

6. LOWER BOUNDS ON κ-COMPLEXITY

Vertex-cut trees allow us to derive lower bounds on the κ-complexity of a graphical realization of a code, as we now show. Let $(\mathcal{G}, \omega)$ be a graph decomposition of a code $\mathcal{C}$, and let (T, β) be a

[6]Without the '-1' in the definition of width of a tree decomposition, a tree with at least two vertices would have treewidth equal to two.

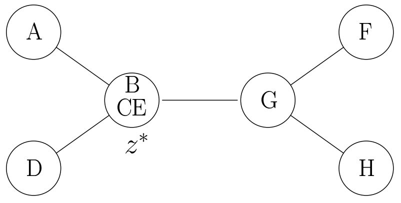


FIGURE 4. The mapping $\alpha : V(T) \to 2^{V(\mathcal{G})}$, with the vertex z^* chosen as shown, constructed from the vertex-cut tree in Figure 3(b).

vertex-cut tree of $\mathcal{G}$. For each $z \in V(T)$, set $J_i = \omega^{-1}(V_i)$ for $i = 1, 2, \ldots, \delta$, where the V_i's are as defined in condition (VC3) of Definition 5.1. Further define

$$m(z) = \dim(\mathcal{C}) - \sum_{i=1}^{\delta} \dim(\mathcal{C}_{J_i}). \tag{7}$$

Now, consider any graphical realization, $\Gamma = (\mathcal{G}, \omega, (\mathcal{S}_e, e \in E), (C_v, v \in V))$, of $\mathcal{C}$ that extends $(\mathcal{G}, \omega)$. By the Vertex-Cut Bound, we have, for each $z \in V(T)$, $\sum_{v \in \beta(z)} \dim(C_v) \geq m(z)$. Since $\sum_{v\in\beta(z)} \dim(C_v) \leq \big(\max_{v \in V(\mathcal{G})} \dim(C_v)\big)\ |\beta(z)| = \kappa(\Gamma)\,|\beta(z)|$, we obtain

$$\kappa(\Gamma)\,|\beta(z)| \geq m(z).$$

Maximizing over all $z \in V(T)$, we get

$$\kappa(\Gamma) \cdot \text{vc-width}(T, \beta) \geq \max_{z \in V(T)} m(z) \overset{\text{def}}{=} \mu(\mathcal{C}; \omega, \beta) \tag{8}$$

We thus have the following proposition.

Proposition 6.1. *Let $(\mathcal{G}, \omega)$ be a graph decomposition of a code $\mathcal{C}$, and let (T, β) be a vertex-cut tree of $\mathcal{G}$. Then, for any graphical realization $\Gamma \in \mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)$, we have*

$$\kappa(\Gamma) \geq \frac{\mu(\mathcal{C}; \omega, \beta)}{\text{vc-width}(T, \beta)}.$$

The lower bound in the above proposition can be brought into a form more convenient for further analysis. To do this, we will construct a tree decomposition (T, γ) of $\mathcal{C}$ such that $\mu(\mathcal{C}; \omega, \beta) = \kappa(\mathcal{C}; T, \gamma)$. So, once again, let $(\mathcal{G}, \omega)$ be a given graph decomposition of $\mathcal{C}$, and (T, β) a given vertex-cut tree of $\mathcal{G}$. We will first give a recipe for constructing tree decompositions of $\mathcal{C}$ from $(\mathcal{G}, \omega)$ and (T, β). Then, we will show how the main ingredient of the recipe may be chosen so that the resulting tree decomposition (T, γ) satisfies $\mu(\mathcal{C}; \omega, \beta) = \kappa(\mathcal{C}; T, \gamma)$.

For any pair of vertices $x, y \in V(T)$, let $(x, y]$ denote the set of vertices on the unique path between x and y in T, including y, but not including x. Also, we will use $\beta(x, y]$, instead of the more cumbersome $\beta((x, y])$, to denote the set $\bigcup_{z \in (x,y]} \beta(z)$. Pick an arbitrary vertex z^* in $V(T)$. Define the mapping $\alpha : V(T) \to 2^{V(\mathcal{G})}$ as follows: $\alpha(z^*) = \beta(z^*)$, and for $z \neq z^*$,

$$\alpha(z) = \beta(z) \setminus \beta(z, z^*].$$

An example of such a mapping α constructed from the vertex-cut tree (T, β) in Figure 3(b) is depicted in Figure 4.

We record in the next two lemmas some properties of the mapping α that we use in the sequel. Recall our convention that for $X \subseteq V(T)$, $\alpha(X) = \bigcup_{x \in X} \alpha(x)$.

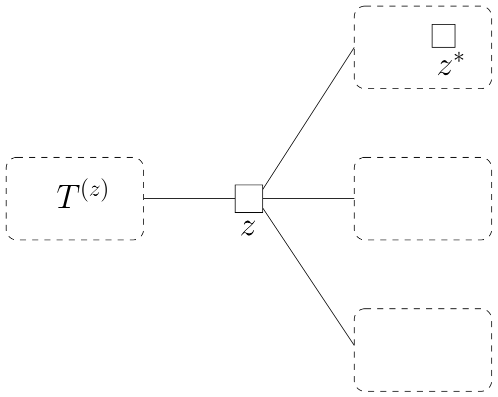


FIGURE 5. A depiction of the situation when $z^* \notin V(T^{(z)})$. Dashed ovals represent components of $T - z$.

Lemma 6.2. *For $z \in V(T)$, if $T^{(z)}$ is any component of $T - z$, then*

$$\alpha(V(T^{(z)})) = \begin{cases} \beta(V(T^{(z)})) & \text{if } z^* \in V(T^{(z)}) \\ \beta(V(T^{(z)})) \setminus \beta(z) & \text{if } z^* \notin V(T^{(z)}). \end{cases}$$

Proof. Suppose first that $z^* \in V(T^{(z)})$, and set $X = V(T^{(z)}) \setminus \{z^*\}$. It then follows from the definition of α that

$$\bigcup_{x\in X} \alpha(x) = \left(\bigcup_{x\in X} \beta(x)\right) \setminus \beta(z^*).$$

Therefore, $\alpha(V(T^{(z)})) = \alpha(X) \cup \alpha(z^*) = (\beta(X) \setminus \beta(z^*)) \cup \beta(z^*) = \beta(V(T^{(z)}))$.

Now, consider the case when $z^* \notin V(T^{(z)})$, as depicted in Figure 5. In this case, the definition of α implies that

$$\bigcup_{x\in V(T^{(z)})} \alpha(x) = \left(\bigcup_{x\in V(T^{(z)})} \beta(x)\right) \setminus \beta[z, z^*],$$

where $[z, z^*]$ denotes the set of vertices on the path between z and z^* in T, including both z and z^*. Note that, as a consequence of condition (VC2) of Definition 5.1, we have, for any $x \in V(T^{(z)})$, $\beta(x) \cap \beta[z, z^*] = \beta(x) \cap \beta(z)$. Hence,

$$\left(\bigcup_{x\in V(T^{(z)})} \beta(x)\right) \setminus \beta[z, z^*] = \left(\bigcup_{x\in V(T^{(z)})} \beta(x)\right) \setminus \beta(z),$$

which proves the desired result. □

Lemma 6.3. *The sets $\alpha(z)$, $z \in V(T)$, form a partition of $V(\mathcal{G})$.*

Proof. Let $T_i^{(z^*)}$, $i = 1, 2, \ldots, \delta$, denote the components of $T - z^*$. Observe that, by Lemma 6.2,

$$\bigcup_{z\neq z^*} \alpha(z) = \bigcup_{i=1}^{\delta} \alpha(V(T_i^{(z^*)})) = \bigcup_{i=1}^{\delta} \beta(V(T_i^{(z^*)})) \setminus \beta(z^*) = \left(\bigcup_{z\neq z^*} \beta(z)\right) \setminus \beta(z^*).$$

Therefore, $\bigcup_{z\in V(T)} \alpha(z) = \bigcup_{z\in V(T)} \beta(z) = V(\mathcal{G})$, by condition (VC1) of Definition 5.1.

Next, we want to show that $\alpha(z) \cap \alpha(z') = \emptyset$ for $z \neq z'$. This is true by definition of α if either $z = z^*$ or $z' = z^*$. So, we henceforth assume that z and z' are distinct vertices in $V(T) \setminus \{z^*\}$.

Suppose that there exists a $v \in \alpha(z) \cap \alpha(z')$. We then have

(1) $v \in \beta(z)$, but $v \notin \beta(y)$ for any $y \in (z, z^*]$; and
(2) $v \in \beta(z')$, but $v \notin \beta(y)$ for any $y \in (z', z^*]$.

In particular, we see that $v \notin \beta(z^*)$. It also follows from (1) and (2) that $z' \notin (z, z^*]$, and $z \notin (z', z^*]$, which together imply that z^* lies on the path between z and z'. However, by (VC2), this means that $v \in \beta(z^*)$, a contradiction. □

We now construct a tree decomposition (T, γ) of the index set, I, of $\mathcal{C}$, by defining an index mapping $\gamma : I \to V(T)$ as follows: for each $i \in I$, set $\gamma(i) = z$ if $\omega(i) \in \alpha(z)$. In other words, for each $z \in V(T)$, $\gamma^{-1}(z) = \omega^{-1}(\alpha(z))$. Since the sets $\alpha(z)$, $z \in V(T)$, form a partition of $V(\mathcal{G})$, the mapping γ is well-defined.

Example 6.1. *Suppose that $\mathcal{C}$ is a code of length 10, defined on the index set $I = \{1, 2, \ldots, 10\}$. For the graph $\mathcal{G}$ shown in Figure 3(a), consider the graph decomposition $(\mathcal{G}, \omega)$ of $\mathcal{C}$ defined by $\omega(1) = \omega(2) = A$, $\omega(3) = \omega(4) = \omega(5) = B$, $\omega(6) = \omega(7) = E$, $\omega(8) = F$, and $\omega(9) = \omega(10) = H$. Let (T, β) be the vertex-cut tree of $\mathcal{G}$ in Figure 3(b), from which we obtain the mapping $\alpha : V(T) \to 2^{V(\mathcal{G})}$ depicted in Figure 4. Based on the last figure, we will use the labels z^*, A, D, G, F and H to identify the vertices of the tree T. Then, the index mapping $\gamma : I \to V(T)$ is given by $\gamma(1) = \gamma(2) = A$, $\gamma(3) = \gamma(4) = \gamma(5) = \gamma(6) = \gamma(7) = z^*$, $\gamma(8) = F$, and $\gamma(9) = \gamma(10) = H$.*

Note that our construction of the tree decomposition (T, γ) depends on the choice of the vertex z^*, via the mapping α. Up to this point, our choice of z^* was arbitrary. We now specify $z^* \in V(T)$ to be such that $m(z^*) = \mu(\mathcal{C}; \omega, \beta)$. We claim that for the tree decomposition (T, γ) arising from such a choice of z^*, we have $\mu(\mathcal{C}; \omega, \beta) = \kappa(\mathcal{C}; T, \gamma)$.

Proposition 6.4. *Given a graph decomposition $(\mathcal{G}, \omega)$ of a code $\mathcal{C}$, and a vertex-cut tree (T, β) of $\mathcal{G}$, there exists a tree decomposition (T, γ) of $\mathcal{C}$ such that $\mu(\mathcal{C}; \omega, \beta) = \kappa(\mathcal{C}; T, \gamma)$.*

Proof. Pick a $z^* \in V(T)$ such that $m(z^*) = \mu(\mathcal{C}; \omega, \beta)$, and construct the tree decomposition (T, γ) as described above. Note that, by (6),

$$\kappa(\mathcal{C}; T, \gamma) = \kappa(\mathcal{M}(\mathcal{C}; T, \gamma)) = \max_{z \in V(T)} k(z),$$

where, for a vertex $z \in V(T)$ of degree δ, $k(z) \overset{\text{def}}{=} \dim(\mathcal{C}) - \sum_{i=1}^{\delta} \dim(\mathcal{C}_{K_i})$, with $K_i = \gamma^{-1}\left(V(T_i^{(z)})\right)$ for $i = 1, 2, \ldots, \delta$. Recall from the definition of γ that $\gamma^{-1}(z) = \omega^{-1}(\alpha(z))$ for any $z \in V(T)$. Hence, $K_i = \omega^{-1}\left(\alpha(V(T_i^{(z)}))\right)$, $i = 1, 2, \ldots, \delta$.

To prove the proposition, we show that $k(z) \le m(z)$ for all $z \in V(T)$, with equality holding if $z = z^*$. From (7), we see that $m(z) = \dim(\mathcal{C}) - \sum_{i=1}^{\delta} \dim(\mathcal{C}_{J_i})$, where, for $i = 1, 2, \ldots, \delta$. $J_i = \omega^{-1}\left(\beta(V(T_i^{(z)})) \setminus \beta(z)\right)$. We must therefore show that, for each $z \in V(T)$, we have $J_i \subseteq K_i$ for all i, which would prove that $k(z) \le m(z)$; and furthermore, when $z = z^*$, we have $J_i = K_i$ for all i, which would prove that $k(z^*) = m(z^*)$.

So, consider any $z \in V(T)$. From Lemma 6.2, we see that if $T_i^{(z)}$ is any component of $T - z$, then

$$\beta(V(T_i^{(z)})) \setminus \beta(z) \subseteq \alpha(V(T_i^{(z)})) \tag{9}$$

Hence, $J_i \subseteq K_i$. Moreover, when $z = z^*$, we always have $z^* \notin V(T_i^{(z)})$, and so, again by Lemma 6.2, equality holds in (9), implying that $J_i = K_i$. □

From Propositions 6.1 and 6.4, we obtain the following theorem.

Theorem 6.5. *Let $(\mathcal{G}, \omega)$ be a graph decomposition of a code $\mathcal{C}$, and let (T, β) be a vertex-cut tree of $\mathcal{G}$. Then, there exists a tree decomposition (T, γ) of $\mathcal{C}$ such that, for any graphical realization*

$\Gamma \in \mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)$, *we have*

$$\kappa(\Gamma) \geq \frac{\kappa(\mathcal{C}; T, \gamma)}{\text{vc-width}(T, \beta)}.$$

Hence, $\kappa(\mathcal{C}; \mathcal{G}, \omega) \geq \frac{\kappa(\mathcal{C}; T, \gamma)}{\text{vc-width}(T, \beta)}$.

The theorem above is a fundamental result with several important consequences, some of which we present here. The first of these is a corollary that gives a lower bound on the κ-complexity of *any* realization of a given code $\mathcal{C}$ on a graph $\mathcal{G}$.

Corollary 6.6. *For a code $\mathcal{C}$ and a connected graph $\mathcal{G}$, we have*

$$\kappa(\mathcal{C}; \mathcal{G}) \geq \frac{\kappa_{\text{tree}}(\mathcal{C})}{\kappa_{\text{vc-tree}}(\mathcal{G})} \geq \frac{\kappa_{\text{tree}}(\mathcal{C})}{\kappa_{\text{tree}}(\mathcal{G}) + 1}.$$

Proof. Consider an arbitrary index mapping $\omega : I \to V(\mathcal{G})$, where I is the index set of $\mathcal{C}$. Let (T, β) be an *optimal* vertex-cut tree of $\mathcal{G}$, by which we mean that vc-width$(T, \beta) = \kappa_{\text{vc-tree}}(\mathcal{G})$. By Theorem 6.5, there exists a tree decomposition (T, γ) of $\mathcal{C}$ such that

$$\kappa(\mathcal{C}; \mathcal{G}, \omega) \geq \frac{\kappa(\mathcal{C}; T, \gamma)}{\text{vc-width}(T, \beta)} \geq \frac{\kappa_{\text{tree}}(\mathcal{C})}{\kappa_{\text{vc-tree}}(\mathcal{G})}.$$

Minimizing over all possible mappings $\omega : I \to V(\mathcal{G})$, we obtain

$$\kappa(\mathcal{C}; \mathcal{G}) \geq \frac{\kappa_{\text{tree}}(\mathcal{C})}{\kappa_{\text{vc-tree}}(\mathcal{G})}.$$

From Lemma 5.2, we also have $\kappa_{\text{vc-tree}}(\mathcal{G}) \leq \kappa_{\text{tree}}(\mathcal{G}) + 1$. □

If, in the above proof, we take (T, β) to be an optimal vertex-cut path instead, *i.e.*, take (T, β) to be a vertex-cut path such that vc-width$(T, \beta) = \kappa_{\text{vc-path}}(\mathcal{G})$, then we obtain the next corollary.

Corollary 6.7. *For a code $\mathcal{C}$ and a connected graph $\mathcal{G}$, we have*

$$\kappa(\mathcal{C}; \mathcal{G}) \geq \frac{\kappa_{\text{path}}(\mathcal{C})}{\kappa_{\text{vc-path}}(\mathcal{G})} \geq \frac{\kappa_{\text{path}}(\mathcal{C})}{\kappa_{\text{path}}(\mathcal{G}) + 1}.$$

The (first) inequality above can be tight, in the sense that there are examples of codes $\mathcal{C}$ and graphs $\mathcal{G}$ for which $\kappa(\mathcal{C}; \mathcal{G}) = \lceil \frac{\kappa_{\text{path}}(\mathcal{C})}{\kappa_{\text{vc-path}}(\mathcal{G})} \rceil$.

Example 6.2. *Take $\mathcal{C}$ to be the $[24, 12, 8]$ binary Golay code, and let $\mathfrak{G}$ be the family of graphs consisting of all n-cycles. From Example 5.1, we know that $\kappa_{\text{vc-path}}(\mathcal{G}) = 2$ for any $\mathcal{G} \in \mathfrak{G}$. It is also known that $\kappa_{\text{path}}(\mathcal{C}) = 9$* [7, Example 1],[17, Section 5]. *Hence, by the bound of Corollary 6.7, noting that $\kappa(\mathcal{C}; \mathcal{G})$ must be an integer, we have $\kappa(\mathcal{C}; \mathcal{G}) \geq 5$ for any $\mathcal{G} \in \mathfrak{G}$. Thus, $\kappa(\mathcal{C}; \mathfrak{G}) \geq 5$. The tailbiting trellis realization of the Golay code given in* [5] *has κ-complexity equal to 5, from which we conclude that $\kappa(\mathcal{C}; \mathfrak{G}) = 5$.*

Using Corollary 6.7 as a starting point, we derive a lower bound on the treewidth of an $[n, k, d]$ linear code. For this, we will also need the following result [4, Theorem 7.1]: if $\mathcal{G}$ is a graph with treewidth at most ℓ, then the pathwidth of $\mathcal{G}$ is at most $(\ell + 1) \log_2 |V(\mathcal{G})|$.

Proposition 6.8. *For an $[n, k, d]$ linear code $\mathcal{C}$, with $n > 1$,*

$$\kappa_{\text{tree}}(\mathcal{C}) \geq \frac{\kappa_{\text{path}}(\mathcal{C})}{3 + 2\log_2(n-1)} \geq \frac{k(d-1)}{n(3 + 2\log_2(n-1))}.$$

Proof. The second inequality above is due to the fact, shown in [15], that[7] for an $[n,k,d]$ linear code $\mathcal{C}$, $\kappa_{\text{path}}(\mathcal{C}) \geq k(d-1)/n$.

The first inequality is proved as follows. Let (T,ω) be an optimal tree decomposition of $\mathcal{C}$, so that $\kappa(\mathcal{C};T,\omega) = \kappa_{\text{tree}}(\mathcal{C})$. As mentioned in Section 3, (T,ω) may be chosen so that T is a cubic tree, and ω is a bijection between the index set of $\mathcal{C}$ and the leaves of T. Thus, T is a cubic tree with n leaves, from which it follows that T has $n-2$ internal nodes. Hence, $|V(T)| = 2n-2$. Since T has treewidth equal to one, by the result from [4] quoted earlier, $\kappa_{\text{path}}(T) \leq 2\log_2(2n-2)$.

We now have, via Corollary 6.7,

$$\kappa(\mathcal{C};T,\omega) \geq \kappa(\mathcal{C};T) \geq \frac{\kappa_{\text{path}}(\mathcal{C})}{\kappa_{\text{path}}(T)+1} \geq \frac{\kappa_{\text{path}}(\mathcal{C})}{2\log_2(2n-2)+1},$$

which proves the proposition. □

In particular, Proposition 6.8 shows that for any linear code $\mathcal{C}$ of length n,

$$\frac{\kappa_{\text{path}}(\mathcal{C})}{\kappa_{\text{tree}}(\mathcal{C})} = O(\log_2 n).$$

This estimate of the ratio $\frac{\kappa_{\text{path}}(\mathcal{C})}{\kappa_{\text{tree}}(\mathcal{C})}$ is the best possible[8], up to the constant implicit in the O-notation. Indeed, it was shown in [11] that a sequence of codes $\mathcal{C}^{(i)}$ $(i = 1,2,3,\ldots)$, with length $n_i = 12(2^i-1)+2$, $\kappa_{\text{tree}}(\mathcal{C}^{(i)}) = 2$ and $\kappa_{\text{path}}(\mathcal{C}^{(i)}) \geq \frac{1}{2}(i+3)$, can be constructed over any finite field $\mathbb{F}$. It is clear that $\frac{\kappa_{\text{path}}(\mathcal{C}^{(i)})}{\kappa_{\text{tree}}(\mathcal{C}^{(i)})}$ grows logarithmically with codelength n_i.

A less precise formulation of the second inequality in Proposition 6.8 is also instructive: there exists a constant $c_0 > 0$ such that, for any $[n,k,d]$ linear code $\mathcal{C}$, with $n > 1$,

$$\kappa_{\text{tree}}(\mathcal{C}) \geq c_0 \frac{k\,d}{n\,\log_2 n}. \tag{10}$$

A noteworthy implication of the above inequality is that code families of bounded treewidth are not very good from an error-correcting perspective. Given an integer $t > 0$, denote by $\mathsf{TW}(t)$ the family of all codes over $\mathbb{F}$ of treewidth at most t. A code family $\mathfrak{C}$ is called *asymptotically good* if there exists a sequence of $[n_i,k_i,d_i]$ codes $\mathcal{C}^{(i)} \in \mathfrak{C}$, with $\lim_i n_i = \infty$, such that $\liminf_i k_i/n_i$ and $\liminf_i d_i/n_i$ are both strictly positive. The following result, an easy consequence of (10), resolves a conjecture in [11].

Corollary 6.9. *Let $\mathcal{C}^{(i)}$, $i = 1,2,3$, be any sequence of $[n_i,k_i,d_i]$ codes such that*

$$\lim_{i\to\infty} \kappa_{\text{tree}}(\mathcal{C}^{(i)}) \frac{\log n_i}{n_i} = 0.$$

Then, either $\lim_{i\to\infty} k_i/n_i = 0$ or $\lim_{i\to\infty} d_i/n_i = 0$. In particular, for any $t > 0$, the code family $\mathsf{TW}(t)$ is not asymptotically good.

In fact, a more general result is true. For a fixed integer $\ell > 0$, let $\mathfrak{G}_\ell$ denote the family of all graphs with vc-treewidth at most ℓ. In particular, note that, by Lemma 5.2, $\mathfrak{G}_\ell$ contains all graphs with treewidth at most $\ell - 1$. Then, for an $[n,k,d]$ linear code $\mathcal{C}$ with $n > 1$, we have, via (3), Corollary 6.6 and Proposition 6.8,

$$\kappa(\mathcal{C};\mathfrak{G}_\ell) \geq \frac{\kappa_{\text{tree}}(\mathcal{C})}{\ell} \geq \frac{k(d-1)}{\ell\, n\,(3+2\log_2(n-1))}.$$

[7]The result in [15] is only explicitly stated as a lower bound on the state max-complexity of any conventional trellis realization of $\mathcal{C}$. However, the state max-complexity of a graphical realization can never exceed the constraint max-complexity of the realization, as noted in Section 3.

[8]It was conjectured in [11] that for codes $\mathcal{C}$ of length n, $\kappa_{\text{path}}(\mathcal{C}) - \kappa_{\text{tree}}(\mathcal{C}) = O(\log n)$, but we now do not believe this to be true.

We thus have the following corollary to Proposition 6.8, which extends Corollary 6.9.

Corollary 6.10. *Given an integer $t > 0$, if $\mathfrak{C}$ is a family of codes over $\mathbb{F}$ with the property that $\kappa(\mathcal{C}; \mathfrak{G}_\ell) \leq t$ for all $\mathcal{C} \in \mathfrak{C}$, then $\mathfrak{C}$ is not asymptotically good.*

A rough interpretation of the last result is that a "good" error-correcting code cannot have a low-complexity realization on a graph with small (vc-)treewidth. In other words, codes that are good from an error-correcting standpoint can have low-complexity realizations only on graphs with large (vc-)treewidth.

7. Concluding Remarks

In this paper, we demonstrated at length the use of the Vertex-Cut Bound in finding lower bounds on the κ-complexity of graphical realizations. As suggested at the end of Section 4, a natural application of the Vertex-Cut Bound is to formulate constrained optimization problems whose solutions are lower bounds to various measures of constraint complexity. This method could be used, for instance, to determine lower bounds on the true computational complexity of sum-product decoding for a given code. This can be done by choosing a constraint complexity measure that accurately reflects the cost of sum-product decoding. Such a complexity measure can be chosen based on detailed counts of the number of arithmetic operations required in the sum-product algorithm; see, for example, [1],[6],[7].

It is now probably fair to say that the main open problem in the area of graphical realizations of codes is to explicitly construct, for a given code $\mathcal{C}$ and an arbitrary graph $\mathcal{G}$, realizations of $\mathcal{C}$ on $\mathcal{G}$ whose constraint complexity is within striking distance of the lower bounds found using the methods of this paper. While we have shown that our bounds can be tight for specific examples of codes $\mathcal{C}$ and graphs $\mathcal{G}$, their tightness in the generic instance remains to be investigated.

Appendix A. An Example

We provide here an example of a code $\mathcal{C}$ and a graph decomposition $(\mathcal{G}, \omega)$ of $\mathcal{C}$, for which there is no realization $\Gamma \in \mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)$ such that $\kappa(\Gamma) = \kappa(\mathcal{C}; \mathcal{G}, \omega)$ and $\kappa^+(\Gamma) = \kappa^+(\mathcal{C}; \mathcal{G}, \omega)$. The example we give is based on Example 3.1 in [14].

Consider the $[11, 3, 3]$ binary linear code $\mathcal{C}$ generated by the codewords 00011100000, 00000111000 and 00101010100. We take $I = \{1, 2, 3, \ldots, 11\}$ to be the index set of $\mathcal{C}$, identifying the index i with the ith coordinate of $\mathcal{C}$. Now, let $\mathcal{G}$ be the 11-cycle, and $\omega : I \to V(\mathcal{G})$ the index mapping depicted in Figure 1. Figure 6 shows a tailbiting trellis realization, Γ_1, of $\mathcal{C}$ that extends $(\mathcal{G}, \omega)$. Note that $\kappa(\Gamma_1) = 2$, while $\kappa^+(\Gamma_1) = 14$. A second tailbiting trellis realization, Γ_2, in $\mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)$ is shown in Figure 7, with $\kappa(\Gamma_2) = 3$ and $\kappa^+(\Gamma_2) = 13$. It can be shown (using arguments similar to those given in Example 3.1 in [14]) that $\kappa(\Gamma_1) = \kappa(\mathcal{C}; \mathcal{G}, \omega)$, while $\kappa^+(\Gamma_2) = \kappa^+(\mathcal{C}; \mathcal{G}, \omega)$, and that there is no realization $\Gamma \in \mathfrak{R}(\mathcal{C}; \mathcal{G}, \omega)$ that simultaneously achieves $\kappa(\mathcal{C}; \mathcal{G}, \omega)$ and $\kappa^+(\mathcal{C}; \mathcal{G}, \omega)$.

References

[1] S.M. Aji and R.J. McEliece, "The generalized distributive law," *IEEE Trans. Inform. Theory*, vol. 46, no. 2, pp. 325–343, 2000.

[2] S. Arnborg, D.G. Corneil and A. Proskurowski, "Complexity of finding embeddings in a k-tree," *SIAM J. Alg. Disc. Meth.*, vol. 8, pp. 277–284, 1987.

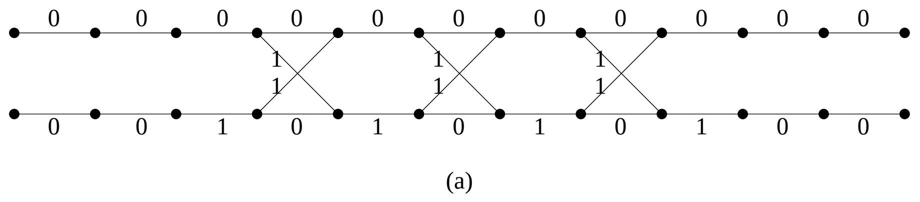
[3] H.L. Bodlaender, "A tourist guide through treewidth," *Acta Cybernetica*, vol. 11, pp. 1–23, 1993.

[4] H.L. Bodlaender, T. Kloks, "Efficient and constructive algorithms for the pathwidth and treewidth of graphs," *J. Algorithms*, vol. 21, no. 2, pp. 358–402, 1996.

[5] A.R. Calderbank, G.D. Forney Jr., and A. Vardy, "Minimal tail-biting trellises: The Golay code and more," *IEEE Trans. Inform. Theory*, vol. 45, pp. 1435–1455, July 1999.

[6] G.D. Forney Jr., "Codes on graphs: normal realizations," *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.

[7] G.D. Forney Jr., "Codes on graphs: constraint complexity of cycle-free realizations of linear codes," *IEEE Trans. Inform. Theory*, vol. 49, no. 7, pp. 1597–1610, July 2003.



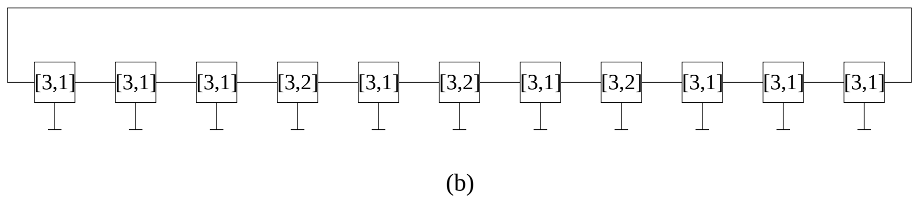
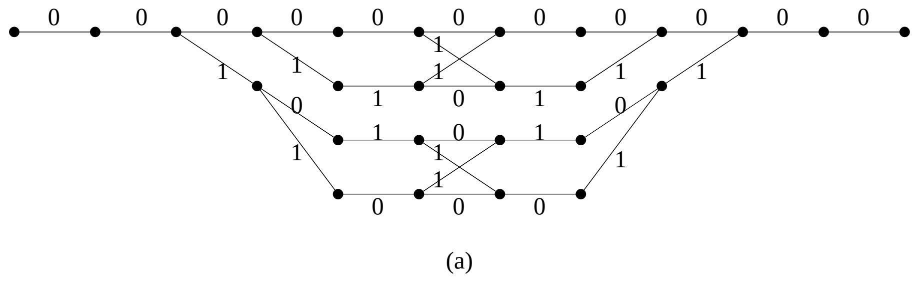


FIGURE 6. A tailbiting trellis realization for an $[11, 3, 3]$ binary linear code. (a) Trellis diagram. (b) Depiction of trellis realization showing the length and dimension of the local constraint codes; all state spaces have dimension 1.



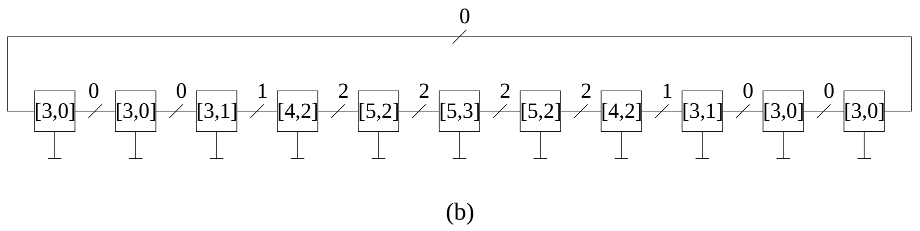


FIGURE 7. A tailbiting trellis realization for an $[11, 3, 3]$ binary linear code. (a) Trellis diagram. (b) Depiction of trellis realization showing the length and dimension of the local constraint codes, and the dimensions of the state spaces.

[8] T.R. Halford and K.M. Chugg, “The extraction and complexity limits of graphical models for linear codes,” *IEEE Trans. Inform. Theory*, to appear.

[9] P. Hliněný and G. Whittle, “Matroid tree-width,” *Europ. J. Combin.*, vol. 27, pp. 1117–1128, 2006.

[10] F.V. Jensen, *An Introduction to Bayesian Networks*, Springer, New York, 1996.

[11] N. Kashyap, “On minimal tree realizations of linear codes,” submitted to *IEEE Trans. Inform. Theory*. ArXiv e-print 0711.1383

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

[13] R. Koetter and A. Vardy, “On the theory of linear trellises,” in *Information, Coding and Mathematics*, M. Blaum, P.G. Farrell and H.C.A. van Tilborg, eds., Kluwer, Boston, Mass., May 2002, pp. 323–354.

[14] R. Koetter and A. Vardy, “The structure of tail-biting trellises: minimality and basic principles,” *IEEE Trans. Inform. Theory*, vol. 49, no. 9, pp. 2081–2105, Sept. 2003.

[15] A. Lafourcade and A. Vardy, “Asymptotically good codes have infinite trellis complexity,” *IEEE. Trans. Inform. Theory*, vol. 41, no. 2, pp. 555–559, March 1995.

[16] N. Robertson and P.D. Seymour, “Graph minors. I. Excluding a forest,” *J. Combin. Theory, Ser. B*, vol. 35, pp. 39–61, 1983.
[17] A. Vardy, “Trellis Structure of Codes,” in *Handbook of Coding Theory*, R. Brualdi, C. Huffman and V. Pless, Eds., Amsterdam, The Netherlands: Elsevier, 1998.
[18] N. Wiberg, *Codes and Decoding on General Graphs*, Ph.D. thesis, Linköping University, Linköping, Sweden, 1996.
[19] N. Wiberg. H.-A. Loeliger and R. Koetter, “Codes and iterative decoding on general graphs,” *Euro. Trans. Telecommun.*, vol. 6, pp. 513–525, Sept./Oct. 1995.