Designing Lexicographic Codes With a Given Trellis Complexity[*]

Ari Trachtenberg[‡§]

October 8, 2002

Abstract

We generalize constructions of lexicographic codes to produce locally optimal codes with a desired trellis decoding complexity. These constructions are efficient for high rate codes and provide a means for automated code design. As a byproduct, we improve known bounds on the parameters of lexicodes.

Keywords: code design, lexicographic codes, minimal trellises, trellis decoding

1 Introduction

Lexicographic codes, or *lexicodes* for short, are greedily constructed codes which were introduced by Levenshtein [9] and again by Conway and Sloane in [4, 5]. Lexicodes have surprisingly good encoding parameters and include, among other famous optimal codes, the Hamming codes, the binary Golay code, and certain quadratic residue codes [5, 10]. Several authors [3, 5, 9] have proved that lexicodes are linear, and comparison with optimal linear codes of the same length and dimension shows that lexicodes are usually within one of the optimal minimum distance [5] and often exhibit the smallest known covering radius [6]. Hence, lexicodes may be regarded as a heuristically good "approximation" to optimal codes. Brualdi and Pless [3] have examined a similar generalization of lexicodes, known as greedy codes, and presented certain bounds on their parameters.

We examine the theoretical underpinnings of lexicodes and investigate various generalizations. Our intent is to graft desired decoding properties onto the heuristically good features of lexicodes. In this way, we develop a means for designing good codes for specific tasks. For example, we can fashion good codes which maintain a desired dimension, error-correction capability, and memory constraint.

Section 2 opens with a definition of the *lexicographic construction*, which iteratively constructs generator matrices for the family of minimum distance d lexicodes, for any $d > 0$. The lexicographic construction is, in turn, a special case of the generalized lexicographic construction, which we abbreviate as the $\mathfrak{G}$-*construction*. This generalized construction encompasses a variety of code families which graft different properties onto lexicode features.

Section 3 is devoted to various instances of the $\mathfrak{G}$-construction. Specifically, Section 3.1 analyzes the role of lexicodes as one of these instances and describes tools for efficiently constructing them. In Section 3.2

[*]A version of this paper appeared in the IEEE Transactions on Information Theory, Vol. 48, No. 1, Jan. 2002, pp 89.

[†]A version of this paper, co-authored with A. Vardy, appeared at the 31st Annual Conference on Information Sciences and Systems.

[‡]The author is with the Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215 (e-mail: trachten@bu.edu).

[§]This work was supported by the Computational Science and Engineering Program at the University of Illinois at Urbana/Champaign.

we demonstrate that a simple modification of the lexicode generator produces "trellis-oriented" lexicodes that locally minimize trellis complexity. These codes exhibit the same "approximately optimal" features of lexicodes but often have a much lower trellis decoding complexity, in some cases reaching the lowest trellis complexity known for their length, rate, and minimum distance. This modification provides a natural heuristic for transforming a given code into a similar code with (often) a better trellis complexity. Finally, in Section 3.3 we examine methods for designing codes with various trellis characteristics, such as constrained state complexity or constrained Viterbi decoding complexity. These methods might be particularly useful for *VLSI*-based decoding, where implementation constraints favor a small trellis state complexity, and for mobile communications, where portability severely limits power consumption.

Section 4 investigates the coset relationship which supports the $\mathfrak{G}$-construction. This relationship establishes bounds on the parameters of the codes produced by the construction and suggests a natural algorithm for computing these codes. The bounds established, though loose, are asymptotically tighter than the lexicode bounds of Brualdi and Pless [3]. We also use the coset relationship to bound the computation time of the $\mathfrak{G}$-construction in Section 5.

Finally, in Section 6 we discuss various applications of the $\mathfrak{G}$-construction and present our conclusions. The appendix lists simulation results for the some of the algorithms described in the paper, extending the list of lexicode parameters originally published in [5].

2 The Lexicographic Construction

A q-ary, length n, minimum distance d lexicode is traditionally defined constructively based on a lexicographic (*i.e.,* dictionary) ordering of vectors in which, for example, 01111 comes before 10000. The construction starts with the set $\mathcal{S} = \{\mathbf{0}\}$ and greedily adds, until exhaustion, the lexicographically earliest vector whose Hamming distance from $\mathcal{S}$ is at least d.

For example, the codewords of the binary lexicode (*i.e.,* $q = 2$) of length $n = 3$ and minimum distance $d = 2$ are marked by a $\bullet$ in Figure 1 and would be computed left-to-right across the figure.

000	001	010	011	100	101	110	111
$\bullet$			$\bullet$		$\bullet$	$\bullet$	

Figure 1: A simple $n = 3, d = 2$ binary lexicode.

This greedy construction always generates a linear code [3, 5, 9]. Thus, we may completely describe a dimension k lexicode by finding k basis vectors (known as "generators") using what we call the *lexicographic construction*. This construction starts with the zero code $\mathcal{L}_0^d = \mathbf{0}$ and greedily adds the lexicographically earliest vector of distance d from the linear space spanned by the previously added vectors; k such iterations form the dimension k code $\mathcal{L}_k^d$. Table 1 demonstrates this construction for $d = 3$; the resulting code is a $(7, 4, 3)$ binary code, meaning that it has length 7, dimension 4, and minimum distance 3.

0000**111**
00**11**001
0**1**01010
1001011

Table 1: Generator matrix for the dimension 4, minimum distance 3 binary code $\mathcal{L}_4^3$. The generator paddings are in bold.

We may understand the lexicographic construction more analytically by making use of the covering radius of each intermediate code $\mathcal{L}_i^d$ in the iteration. The covering radius of a length n code is the smallest integer ρ with the property that Hamming balls of radius ρ centered at codewords of the code will cover every

vector of $\mathbb{F}_q^n$. In other words, ρ is the maximum distance of a vector in $\mathbb{F}_q^n$ from the code. As an example, one can readily see that the binary lexicode in Figure 1 has a covering radius of 1 because every vector in $\mathbb{F}_2^3$ is at most of Hamming distance 1 from a code vector.

An iteration of the lexicographic construction on an intermediate code $\mathbb{C}$ whose covering radius is ρ and minimum distance is d can thus be understood as the addition of a generator vector:

$$\langle 1^{d-\rho} \mid f_{\text{lexi}}(\mathbb{C}) \rangle$$

where $1^{d-\rho}$, known as the *generator padding*, has the usual meaning of $(d-\rho)$ successive 1's, and the function $f_{\text{lexi}}(\mathbb{C})$ returns the lexicographically earliest vector of distance ρ from $\mathbb{C}$:

$$f_{\text{lexi}}(\mathbb{C}) = v \in \mathbb{F}_q^* \text{ s.t. } \forall w \in \mathbb{F}_q^*, \quad v <_{\text{lexi}} w \text{ or } d(w,\mathbb{C}) < d(v,\mathbb{C}), \quad (1)$$

where we define

$$\mathbb{F}_q^* = \bigcup_{i=1}^{\infty} \mathbb{F}_q^i.$$

We use the notation $v <_{\text{lexi}} w$ to denote that v appears before w in a lexicographical ordering of $\mathbb{F}_q^*$. Moreover we use the customary notation $\langle \cdot | \cdot \rangle$ to denote concatenation.

The linear codes generated by the lexicographic construction described above are precisely the lexicodes.

Theorem 1 *Over a finite field $\mathbb{F}_q$, a dimension k, minimum distance d code $\mathbb{C}$ is a lexicode if and only if it is produced by the lexicographic construction, namely $\mathbb{C} = \mathcal{L}_k^d$.*

Proof: We first prove that $\mathcal{L}_k^d$ is always a lexicode, by induction on k for an arbitrary, fixed d. For the base case it is clear that $\mathcal{L}_1^d = \{0^d, 1^d\}$ is a $(d, 1, d)$ lexicode. Now assume as an inductive hypothesis that $\mathcal{L}_k^d$ is a dimension k, minimum distance d lexicode. From the definition of the lexicographic construction, $\mathcal{L}_{k+1}^d$ has parameters $(n+d-\rho, k+1, d)$, where ρ is the covering radius of $\mathcal{L}_k^d$. Consider the $(n+d-\rho, k', d)$ lexicode $\mathbb{C}_{k'}$ constructed by repeatedly choosing the appropriate lexicographically-earliest vectors in $\mathbb{F}_q^{n+d-\rho}$. Clearly, we generate $\mathbb{C}_k$ in the process of this construction, so that $\mathbb{C}_k \subseteq \mathbb{C}_{k'}$ and, by the inductive hypothesis, $\mathcal{L}_k^d \subseteq \mathbb{C}_{k'}$. Moreover, the vector

$$v = \left\langle 1^{d-\rho} \mid f_{\text{lexi}}(\mathbb{C}_k) \right\rangle$$

is the lexicographically earliest vector of distance ρ from $\mathcal{L}_k^d$. Thus, it must be the case that $v \in \mathbb{C}_{k'}$ and, since $\mathbb{C}_{k'}$ is linear, $\mathcal{L}_{k+1}^d \subseteq \mathbb{C}_{k'}$.

In addition, any vector $v \in \mathbb{F}_q^{n+d-\rho}$ that is in $\mathbb{C}_{k'}$ but not in $\mathcal{L}_{k+1}^d$ would necessarily have its n right-most bits at distance $\leq \rho$ from $\mathcal{L}_{k+1}^d$, and its other bits at distance $\leq \lceil (d-\rho)/2 \rceil$ from $\mathcal{L}_{k+1}^d$. Noting that $\rho < d$, we may now use the triangle inequality to see that the distance from v to $\mathcal{L}_{k+1}^d$ (and consequently also $\mathbb{C}_{k'}$) must be at most

$$\rho + \lceil (d-\rho)/2 \rceil \;=\; \lceil (d+\rho)/2 \rceil \;<\; d.$$

This contradicts our constructive definition of $\mathbb{C}_{k'}$, so it must be that $\mathcal{L}_{k+1}^d = \mathbb{C}_{k'}$.

For the converse assertion of the theorem, we need to show that a non-trivial (n, k, d) lexicode $\mathbb{C}$ can be constructed by the lexicographic construction. The code $\mathcal{L}_k^d$ produced by k iterations of the minimum-distance d lexicographic construction is clearly a lexicode, from the first part of the proof. Since all distance d lexicodes are ordered by inclusion, as we saw in the first part of this proof, we may conclude that either $\mathbb{C} \subseteq \mathcal{L}_k^d$ or $\mathcal{L}_k^d \subseteq \mathbb{C}$. However $\mathcal{L}_k^d$ and $\mathbb{C}$ are both non-trivial[1] dimension k codes, so it follows that they must be equal. ∎

Theorem 1 allows us to bypass the codeword-by-codeword lexicode construction, and instead directly compute the generator matrix of a desired lexicode.

[1] A non-trivial code is one whose generator matrix contains no all-zero columns.

2.1 Generalization

The lexicographic construction may be extended to produce codes with desired characteristics using the *generalized lexicographic construction*, which we abbreviate as the $\mathfrak{G}$-*construction*. The $\mathfrak{G}$-construction replaces the "lexicographically earliest" heuristic used in building lexicodes with an arbitrary function. This allows us to generate arbitrary greedy codes in which various properties are grafted upon the good code parameters of the lexicodes.

Definition 1 *The generalized lexicographic construction is initialized with a linear (n, k, d) seed code $\mathbb{C}$ and iteratively constructs the family of codes*

$$\mathfrak{G}(f, \mathbb{C}) \;=\; \{\mathfrak{G}_i(f, \mathbb{C}) : i \in \{0, 1, 2, \ldots\}\}$$

using a mapping from codes to vectors:

$$f(\cdot) : \mathbb{C} \subseteq \mathbb{F}_q^* \longmapsto v \in \mathbb{F}_q^*. \tag{2}$$

The construction follows the scheme:

- $\mathfrak{G}_0(f, \mathbb{C})$ *is trivially the code* $\mathbb{C}$*;*
- $\mathfrak{G}_i(f, \mathbb{C})$ *is computed by adding to* $\mathfrak{G}_{i-1}(f, \mathbb{C})$ *the generator*

$$\left(1^{\Delta_i} \;\middle|\; \lambda_i\right) \tag{3}$$

where $\lambda_i = f(\mathfrak{G}_{i-1}(f, \mathbb{C}))$ and Δ_i is defined to be d minus the Hamming distance from λ_i to the code $\mathfrak{G}_{i-1}(f, \mathbb{C})$.

We will call $f(\cdot)$ the *generating mapping* of the construction. The familiar lexicode family of minimum distance d is thus the simple special case $\mathfrak{G}(f_{\text{lexi}}, \mathbb{S}_d)$, where we use the trivial seed code $\mathbb{S}_d \stackrel{\text{def}}{=} \{0^d, 1^d\}$. The code in Table 1 is $\mathfrak{G}_3(f_{\text{lexi}}, \mathbb{S}_3)$, as can be verified by hand.

Despite its generality, there are many linear codes that cannot be constructed using a non-trivial application of the $\mathfrak{G}$-construction. The code given by the following basis vectors is one such example:

$$\begin{array}{c} 1110000 \\ 0001111 \end{array}$$

For sake of simplicity we shall concern ourselves only with binary codes in the remainder of this paper, though the extension to q-ary codes is fairly straightforward.

3 Instances of the $\mathfrak{G}$-construction

The $\mathfrak{G}$-construction algorithm is technically complicated and will be relegated to Section 4. A generating mapping uniquely determines a corresponding family of codes, known as a $\mathfrak{G}$-family, produced by the $\mathfrak{G}$-construction. In this section we shall analyze various generating mappings that can be used to design these families for specific decoding needs.

3.1 Lexicodes

As mentioned in the previous section, the mapping f_{lexi} generates the lexicodes. The computation of f_{lexi} is handled by a simple greedy algorithm, presented as Method 1, which operates in linear time (in its inputs), subject to appropriate knowledge of the input code. The speed with which we can compute f_{lexi} permits efficient generation of lexicodes, as we shall see later in Section 5.

In order to compute f_{lexi} we will first transform the generator matrix of the code into a minimal span generator matrix (`MSGM`) form, in which the sum of the spans of the generators is minimized. Adapting the notation in [11], the span of a binary n-vector $x = (x_1, x_2, x_3, \ldots, x_n)$ is $\mathrm{R}(x) - \mathrm{L}(x)$, where $\mathrm{R}(\cdot)$ and $\mathrm{L}(\cdot)$ are the rightmost (*i.e.,* largest) and leftmost (*i.e.,* smallest) index i, respectively, such that $x_i \neq 0$. Thus, the span of the vector $x = (0001001100)$ is $\mathrm{R}(x) - \mathrm{L}(x) = 8 - 4 = 4$. We can efficiently transform any matrix into `MSGM` form using the greedy algorithm in [11]. We note that two vectors of an `MSGM` cannot have their leftmost or their rightmost index in common, or else they may be added to produce a generator matrix with shorter span.

Given an `MSGM` for a code and a set of coset representatives $\mathcal{V}$, Method 1 computes the lexicographically earliest vector among the cosets represented in $\mathcal{V}$.

Method 1 *Consider a set of vectors $\mathcal{V}$ representing cosets of a code $\mathbb{C}$ whose length is n. Let G be an `MSGM` for $\mathbb{C}$ whose generators are in lexicographically increasing order (*i.e., $G_1 <_{lexi} G_2 <_{lexi} G_3 \ldots <_{lexi} G_n$*). The following greedy method computes the lexicographically earliest vector among the represented cosets in $O(n|V|)$ time and $O(n)$ space.*

1. **for each** $v = (v_1, v_2, v_3, \ldots, v_{|\mathcal{V}|}) \in \mathcal{V}$ **do**
2. **for** i from n down to 1
3. **if** $v_{\mathrm{L}(G_i)}$ is a 1 then
4. $v \leftarrow v + G_i$
5. **store** the modified v
6. **among** all stored v, return the lexicographically earliest

This method looks for the generators whose left-most 1-bit corresponds to 1-bits in vectors $v \in V$. Figure 2 demonstrates this method with the set $\mathcal{V} = \{1010000\}$ on the $(7, 4, 3)$ code described in Table 1. For this case, the vector 0001011 is the lexicographically earliest vector in the same coset as 1110000. Note that the ordering of the generators in the `MSGM` is significant and that different orderings might not yield the lexicographically earliest vector. For example, if generators G_2 and G_3 are switched, then Method 1 yields 0010010, which is not the lexicographically earliest vector desired.

$G_1 =$	0000111	v:	1110000
$G_2 =$	0011110	$v \leftarrow v + G_4$:	0111000
$G_3 =$	0110100	$v \leftarrow v + G_3$:	0001100
$G_4 =$	1001000	$v \leftarrow v + G_1$:	0001011

Figure 2: Method 1 applied to the code in Table 1 with initial condition $\mathcal{V} = \{1110000\}$. The various values taken on by v during the computation are shown on the right-hand region.

We will now prove the correctness of Method 1. In our applications, $\mathcal{V}$ will typically be the set of leaders of cosets with maximum distance from $\mathbb{C}$. These coset leaders are defined to be the minimum-weight vectors in their corresponding cosets, so that Method 1 computes precisely $f_{\text{lexi}}(\mathbb{C})$.

Proof of Method 1: We first show correctness of the method by proving that, for each $v \in \mathcal{V}$, lines 2-5 compute the lexicographically earliest vector in the same coset as v. This way, line 6 in the method will return the lexicographically earliest vector among all the cosets represented.

We know from [11, Thm 6.11 and Lemma 6.7] that the rows of G have the *predictable support property*:

$$\text{span}\left(\sum_{j \in J} G_j\right) = \bigcup_{j \in J} \text{span}(G_j) \tag{4}$$

for every subset $J \subseteq \{1, 2, 3, ..., n\}$.

Now suppose that v_{best} is the lexicographically earliest vector in the same coset as v, but that instead v_{stored} is the vector stored on line 5 of the method. Our goal will be to show that the difference between these two vectors, denoted v_{diff}, is in fact 0.

Clearly v, v_{best}, and v_{stored} are necessarily in the same coset of $\mathbb{C}$. Moreover, since v and v_{stored} are in the same coset, v_{diff} must be a codeword of $\mathbb{C}$ so that we can write (for an appropriate set J_{diff}):

$$v_{\text{diff}} = \sum_{j \in J_{\text{diff}}} G_j. \tag{5}$$

Then, applying the *predictable span property* of G,

$$\text{span}(v_{\text{diff}}) = \bigcup_{j \in J_{\text{diff}}} \text{span}(G_j). \tag{6}$$

Assume (for sake of contradiction) that $L(v_{\text{diff}}) = k$, meaning that the left-most 1 bit of v_{diff} occurs at the k-th index. This implies that, starting from the left-most bit, v_{best} and v_{stored} are identical until the k-th index, on which they differ. Since, by definition, v_{best} comes lexicographically before v_{stored}, it must be that v_{best} has a 0 and v_{stored} a 1 at the k-th index. Then (6) implies that there must be some generator G_j whose left-index is k; however, the lexicographic ordering of the generators insures that G_j would have eliminated any 1 at index k in v_{stored}, providing a contradiction. Thus, the left-index of v_{diff} could not possibly be at location k, for any k, so that v_{diff} must be 0 and correctness of the method is proved. The time bound follows straightforwardly, and the space bound follows from noting that only the lexicographically earliest vector v needs to be stored in line 5 of the method. ∎

Method 1 provides a fast way of computing the lexicographic generating mapping. Appendix .1 gives computed parameters for many lexicodes that were thus computed. This list is more exhaustive than what is currently available in the literature [3, 5].

3.2 Trellis-oriented lexicodes

We now consider a different generating mapping for the $\mathfrak{G}$-construction which will generate families of codes oriented towards a reduced decoding trellis.

There is a unique, minimal [12] trellis for an arbitrary linear block code. Known as the `BCJR` [1] trellis, it has no more edges or vertices at each time index than any other trellis for the code and can be constructed fairly easily [8, 11]. It is shown in [8, 11] that the minimal span generator matrix (`MSGM`) for a linear code, in which the sum of the spans of the binary generators is minimized, reflects the properties of the corresponding `BCJR` trellis. Namely, if G is an `MSGM` of a code $\mathbb{C}$, comprised of the generators $G_1, \ldots, G_k$, then the number of vertices V_i and edges E_i in the corresponding `BCJR` trellis is given by:

$$\begin{aligned} |V_i| &= 2^{k-p_i-f_i} \\ |E_{i,i+1}| &= 2^{k-p_i-f_{i+1}} \end{aligned} \tag{7}$$

where p_i and f_i are the dimensions of the past and future subcodes for the i-th index of $\mathbb{C}$. These dimensions may be computed for $i = 0, 1, \ldots, n$ as follows [11]:

$$\begin{aligned} p_i &= |\{j : \mathrm{R}(G_j) \leq i\}| \\ f_i &= |\{j : \mathrm{L}(G_j) \geq i+1\}| \end{aligned}$$

As before, R$(\cdot)$ and L$(\cdot)$ refer to the rightmost and leftmost non-zero indices of a vector, respectively. With a minor modification of the lexicographic construction we may exploit the above relations to locally minimize two measures of trellis complexity: state complexity, which is the maximum number of states at any time interval of the trellis, and Viterbi decoding complexity. Specifically, $\mathfrak{G}(f_{\text{trelli}}, \mathbb{C})$ is such a family of codes, and it locally minimizes trellis complexity if $f_{\text{trelli}}(\mathbb{C})$ returns the vector with maximum distance from $\mathbb{C}$ whose bit-wise reverse (REV) is lexicographically earliest:

$$f_{\text{trelli}}(\mathbb{C}) = v \in \mathbb{F}_q^* \text{ s.t. } \forall w \in \mathbb{F}_q^*, \\ \text{REV}(v) <_{\text{lexi}} \text{REV}(w) \text{ or } d(w, \mathbb{C}) < d(v, \mathbb{C}). \quad (8)$$

For example, if the vectors {1011, 1101, 1110} are at maximum distance from a code $\mathbb{C}$, then $f_{\text{trelli}}(\mathbb{C})$ returns the vector 1110. Because of their locally minimal trellis complexity, these codes may be justly called "trellis-oriented."

Theorem 2 establishes that $f_{\text{trelli}}(\cdot)$ locally minimizes trellis complexity among local extensions with optimal code parameters. It is possible to improve Viterbi decoding complexity by using a generating mapping which produces a longer extension, but the information rate of the resulting code will be inferior.

Theorem 2 *Let $\mathbb{C}$ be a linear code. Among those single-iteration extensions $\mathfrak{G}_1(f, \mathbb{C})$ with shortest length, the generator mapping $f(\cdot) = f_{trelli}(\cdot)$ minimizes the Viterbi decoding complexity and trellis state complexity.*

Proof: Suppose that G is an MSGM for the (n, k, d) code $\mathbb{C}$ whose past and future subcodes have dimensions p_i and f_i respectively, and whose trellis has vertices V_i and edges $E_{i,i+1}$ at the i-th time interval. Now, consider appending to G the generator $v = \langle 1^{\Delta} | \lambda \rangle$ as in Equation (3) of the definition of the $\mathfrak{G}$-construction. The generated code $\mathbb{C}' = \mathfrak{G}_1(f, \mathbb{C})$ will have parameters $(n' = n + \Delta, k + 1, d)$. Without loss of generality we may assume that the resulting generator matrix of $\mathbb{C}'$ is an MSGM. If we denote the difference in lengths between $\mathbb{C}'$ and $\mathbb{C}$ by $\nabla n = n' - n$, then a simple analysis shows that $\mathbb{C}'$ will have past and future subcode dimensions:

$$p'_i = \begin{cases} 0 & \text{if } 0 \le i \le \nabla n \\ p_{i-\nabla n} & \text{if } \nabla n < i < R(v) \\ p_{i-\nabla n} + 1 & \text{if } R(v) \le i \le n' \end{cases}$$
$$f'_i = \begin{cases} k+1 & \text{if } i = 0 \\ k & \text{if } 0 < i \le \nabla n \\ f_{i-\nabla n} & \text{if } \nabla n < i \le n' \end{cases} \quad (9)$$

We may then use (7) and (9) at each time unit i to compute the number of vertices and edges $(|V'_i|, |E'_i|)$ in the BCJR trellis for $\mathbb{C}'$ based on the vertices and edges $(|V_i|, |E_i|)$ in the BCJR trellis for $\mathbb{C}$:

$$(|V'_i|, |E'_i|) = \begin{cases} (1, 2) & \text{if } i = 0 \\ (2, 2) & \text{if } 0 < i < \nabla n \\ (2|V_{i-\nabla n}|, 2|E_{i-\nabla n}|) & \text{if } \nabla n \le i < R(v) \\ (|V_{i-\nabla n}|, |E_{i-\nabla n}|) & \text{if } R(v) \le i \le n' \end{cases} \quad (10)$$

It is now a simple matter of algebra to compute the difference in Viterbi decoding complexities between the trellises of $\mathbb{C}'$ and $\mathbb{C}$:

$$\Delta(\text{Viterbi complexity}) = (2|E'| - |V'| + 1) - (2|E| - |V| + 1) \\ = 3 + 2(\nabla n - 1) + 2 \left[\sum_{i=0}^{\text{R}(v)-\nabla n-1} |E_i| \right] - \left[\sum_{i=0}^{\text{R}(v)-\nabla n-1} |V_i| \right] \quad (11)$$

Since $|E_i| \geq |V_i|$ for all i, minimizing the change in Viterbi decoding complexity depends on minimizing $R(v) = R\left(1^{\Delta}|\lambda\right)$, which in turn depends on having the 1 bits of λ as far to the left as possible. On the other hand, λ cannot have weight less than the covering radius ρ of $\mathbb{C}$ if it is to produce an extension of shortest length. Thus, the mapping $f_{\text{trelli}}(\cdot)$ is one of several mappings which meet both criteria for local optimality: minimizing $R(v)$ and having $\text{wt}(\lambda) = \rho$. Other intuitive generating mappings, such as picking lexicographically *latest* vectors at maximum distance from the code, are not always locally optimal because the lexicographically latest vector need not (and generally does not) have the minimum rightmost index.

Equation 10 also proves that minimizing $\text{R}(v)$ is the appropriate criterion for minimizing state complexity. This is because larger values of $\text{R}(v)$ correspond to doubling more vertices $|V_{i-\nabla n}|$ when generating V_i'. Thus, $f_{\text{trelli}}(\cdot)$ locally minimizes state complexity as well. ∎

0000**111**
00**11**100
0**11**0010
1111000

Table 2: Generator matrix for the dimension 4 minimum distance 3 trellis-oriented $\mathfrak{G}$-code. The generator padding for each iteration is marked in bold.

Table 2 depicts the $(7, 4, 3)$ *trellis-oriented* $\mathfrak{G}$-code that was generated similarly to the code in Table 1. In fact, Method 1 can be trivially reversed to compute $f_{\text{trelli}}(\cdot)$, without affecting the running time or space:

Method 2 *Consider a set of vectors $\mathcal{V}$, and a code $\mathbb{C}$ with MSGM G whose generators are in lexicographically increasing order. The following greedy method computes the lexicographically earliest bitwise reversed vector among the represented cosets in $O(n|V|)$ time and $O(n)$ space.*

1. **for each** $v = (v_1, v_2, v_3, \ldots, v_{|\mathcal{V}|}) \in \mathcal{V}$ **do**
2. **for** i from 1 to n
3. **if** $v_{\text{R}(G_i)}$ is a 1 then
4. $v \leftarrow v + G_i$
5. **store** the modified v;
6. **among** all stored v, return the lexicographically earliest

The proof of correctness and complexity for Method 2 follows trivially from the proof of Method 1.

3.3 Trellis-bounded lexicodes

It is often useful to bound various parameters of a trellis so that decoding can be efficiently handled by a system with complexity constraints, such as a *VLSI* chip with certain implementation constraints or a mobile device with certain power limits. We consider generating mappings which bound Viterbi decoding complexity and trellis-state complexity. The trellis-state complexity is a strong indicator of the decoding complexity of a code because it asymptotically restricts memory-usage and the number of bifurcations in the trellis.

We may produce $\mathfrak{G}$-families which constrain state complexity by simply restricting the generating mapping to returning only vectors which maintain state complexity bounds. In order to duplicate the nice locally optimal properties of f_{trelli} we similarly design the state-constrained generating mapping f_{state} to return, among all candidate vectors which maintain the state complexity bound, the vector of maximum distance

from the given code whose reverse is lexicographically earliest. Formally, $f_{\text{state}}(s, \mathbb{C})$ is the same as $f_{\text{trelli}}(\mathbb{C})$ with the additional constraint that the code formed by adding the return vector v to $\mathbb{C}$ has a trellis state complexity bounded by s.

We may similarly produce $\mathfrak{G}$-families which constrain Viterbi decoding complexity by using the generating mapping f_{decoding}, which like f_{state} restricts its output to producing codes whose Viterbi decoding complexity is some function $g(k)$ of the dimension k of the underlying code.

The computations of f_{state} and f_{decoding} are straightforward by-products of the mechanism behind Theorem 3 (in the next section), which drives the computation of codes in an arbitrary $\mathfrak{G}$-family. Specifically, for any such $\mathfrak{G}$-code, we may quickly derive an `MSGM` from which all the necessary trellis properties may be easily gleaned and suitably constrained. Unfortunately, the worst case (extreme bounding) computation of either of these generating mappings involves exhaustively searching through every coset leader in an attempt to meet the constraint. Appendices .2 and .3 demonstrate the effects of bounding various trellis properties.

4 Relation Between Cosets

So far we have addressed the computation of various generating mappings. In this and subsequent sections we will complete our analysis by addressing the theoretical mechanism which supports the $\mathfrak{G}$-construction. Specifically, the $\mathfrak{G}$-construction establishes an important relationship between the coset leaders of the codes produced at successive iterations. This relationship allows us to efficiently compute the coset leaders of many $\mathfrak{G}$-codes (and hence their generator matrices) as well as to determine bounds on their code parameters. We describe this relationship by first associating a *companion* with each coset leader. Recall that a coset leader is a designated vector of minimum weight in a given coset.

Definition 2 *The* companion *of a coset leader l, with respect to a vector $v \in \mathbb{F}_2^n$, is the leader of the coset containing $l + v$. It is denoted $\kappa_v(l)$, or $\kappa(l)$ in context.*

We show that one iteration of the lexicographic construction produces a code whose coset leaders are either $\langle a|u\rangle$ or $\langle \bar{a}|\kappa(u)\rangle$ (depending on which has lower Hamming weight), for each vector a of appropriate size and coset leader u in the original code. Here $\bar{a}$ refers to the complement of a.

Theorem 3 *Consider an (n, k, d) code $\mathbb{C}$ with a fixed set $\mathcal{S}$ of coset leaders, and the linear code $\mathbb{C}'$ spanned by $\mathbb{C} \cup \left\{(1^\Delta|\lambda)\right\}$ for some $\lambda \in \mathbb{F}_2^n$ and $\Delta \in \mathbb{Z}$. Then the set $\mathcal{S}'$ of coset leaders of $\mathbb{C}'$ can be obtained from $\mathcal{S}$ according to the following bijective correspondence:*

$$\phi : (a,\, l) \in \{\langle 0|\mathbb{F}_2^{\Delta-1}\rangle\} \times \mathcal{S} \longmapsto l' \in \mathcal{S}'$$

where l' is defined by the property

$$l' = \begin{cases} \langle a|l\rangle & \text{if } \mathrm{wt}(\langle a|l\rangle) \leq \mathrm{wt}\left(\langle \bar{a}|\kappa_\lambda(l)\rangle\right), \\ \langle \bar{a}|\kappa_\lambda(l)\rangle & \text{if } \mathrm{wt}(\langle a|l\rangle) > \mathrm{wt}(\langle \bar{a}|\kappa_\lambda(l)\rangle). \end{cases}$$

In other words, every coset of leader l in S, when paired with a vector a, produces a coset leader l' in S', and vice versa.

Proof: We introduce the following notation to simplify the proof:

$$\mathrm{g}(a, l) \stackrel{\text{def}}{=} \langle a \mid l\rangle \text{ and } \mathrm{h}(a, l) \stackrel{\text{def}}{=} \langle \bar{a} \mid \kappa_\lambda(l)\rangle\,. \tag{12}$$

This proof then rests upon two observations. The first observation is that $\mathrm{g}(a,l)$ and $\mathrm{h}(a,l)$ are always in the same coset of the new code $\mathbb{C}'$. This is clear because:

$$\begin{aligned}\mathrm{g}(a,l) \ + \ \mathrm{h}(a,l) &= \langle [a+\bar{a}] \ | \ [l+\kappa(l)] \rangle \\ &= \langle 1^{\Delta} \ | \ [l+(l+\lambda+c)] \rangle, \text{ for } c \in \mathbb{C} \\ &= \langle 1^{\Delta} \mid \lambda \rangle + \langle 0^{\Delta} \mid c \rangle \\ &\in \ \mathbb{C}'.\end{aligned}$$

Here Δ is the padding value, as presented in the statement of the theorem.

The second observation of this proof is that $\mathrm{g}(a,l)$ and $\mathrm{g}(a',l')$ are generally not in the same coset of $\mathbb{C}'$. More specifically, these two vectors are in the same coset precisely when l and l' are in distinct cosets of $\mathbb{C}$ and $\bar{a} \neq a'$. To see this we analyze the two possibilities for the sum $\mathrm{g}(a,l)+\mathrm{g}(a',l')$. In the first case, its leading bits are neither 0^{Δ} nor 1^{Δ}. Alternatively, its leading bits are 0^{Δ} but its trailing bits are the sum of distinct coset leaders $l+l' \notin \mathbb{C}$. Both possibilities preclude $\mathrm{g}(a,l)+\mathrm{g}(a',l')$ from being a codeword of $\mathbb{C}'$, proving the observation.

Based on the above two observations, $\mathrm{g}(\cdot,\cdot)$ and $\mathrm{h}(\cdot,\cdot)$ represent the same two-to-one correspondence between (vector, coset-leader) pairs (a,l) and the cosets of $\mathbb{C}'$. In fact, for each $a \in \mathbb{F}_2^n$ and coset leader l of $\mathbb{C}$, the *coset* of $\mathrm{g}(a,l)$ contains only the vectors:

$$\begin{cases} \langle a \mid (l+c) \rangle \\ \langle \bar{a} \mid (l+\lambda+c) \rangle \end{cases}$$

for any $c \in \mathbb{C}$.

In addition, for any $c \in \mathbb{C}$, $\mathrm{g}(a,l)$ cannot have weight greater than the weight of $\langle a|(l+c)\rangle$, because l is a coset leader, and $\mathrm{h}(a,l)$ cannot have weight greater than $\langle \bar{a}|(l+\lambda+c)\rangle$. Therefore, all the vectors in the coset containing $\mathrm{g}(a,l)$ will have weight not less than $\min\{\mathrm{wt}(\mathrm{g}(a,l)), \mathrm{wt}(\mathrm{h}(a,l))\}$, so that either $\mathrm{g}(a,l)$ or $\mathrm{h}(a,l)$ must be a coset leader in $\mathbb{C}'$. Since $\mathrm{g}(a,l)$ and $\mathrm{h}(a,l)$ are correspondences, $\mathcal{S}'$ is the complete set of coset leaders of $\mathbb{C}'$. ∎

Table 3 demonstrates the use of the ϕ correspondence of Theorem 3 to generate the coset leaders of the $(6,2,4)$ binary lexicode $\mathcal{L}_4^2$ from the coset leaders of the $(4,1,4)$ binary lexicode $\mathcal{L}_4^1$. In essence, the ϕ correspondence indicates that the coset leaders of a $\mathfrak{G}$-code are augmented permutations of the coset leaders of its predecessor in the $\mathfrak{G}$-construction. One such permutation produces the lexicodes, whereas other permutations produce the various other codes described in this paper.

$a \mid l$	$\bar{a} \mid \kappa_\lambda(l)$	*corresponding* $l' \in \mathcal{S}'$
00 0000	11 0101	00 0000
01 0000	10 0101	01 0000
00 0001	11 0100	00 0001
01 0001	10 0100	01 0001
00 0010	11 1000	00 0010
01 0010	10 1000	01 0010
00 0011	11 0110	00 0011
01 0011	10 0110	01 0011
$\ddots$	$\ddots$	$\ddots$

Table 3: A partial table of coset leaders of the $(6,2,4)$ lexicode. It is derived from the (4,1,4) lexicode, using Theorem 3 with $\lambda = 0101$ and $\Delta = 2$. Spaces between bits are used to highlight which vectors are being concatenated.

4.1 Bounds on code parameters

Theorem 3 yields a description of the coset leaders of a code $\mathfrak{G}_k$ in terms of the coset leaders of its predecessor $\mathfrak{G}_{k-1}$ in the $\mathfrak{G}$-construction. The maximum weight coset leader of $\mathfrak{G}_k$, in turn, determines the covering radius of this $\mathfrak{G}$-code and, hence, the code parameters of the subsequent codes in the construction. This insight allows us to improve the known bounds on the parameters of lexicodes.

4.1.1 Bounding the Covering Radius

Our first bound is a recursive *upper* bound on the covering radius ρ_m of the m-th code of an arbitrary $\mathfrak{G}$-family. We shall use the term Δ_m to refer to the corresponding term in Definition 1 for this m-th code.

Lemma 1 *For any $\mathfrak{G}$-family of codes and $m > 1$,*

$$\rho_m \;\leq\; \left\lfloor \frac{d}{2} \right\rfloor + \rho_{m-1}.$$

Proof: Consider constructing $\mathfrak{G}_m$ from $\mathfrak{G}_{m-1}$ using Theorem 3. Any two coset leaders l and $\kappa(l)$ of $\mathfrak{G}_{m-1}$ must each have weight at most ρ_{m-1}, by the definition of the covering radius. Consider the weight of a corresponding coset leader of $\mathfrak{G}_m$, based on Theorem 3:

$$\begin{aligned} &\min\left\{\mathrm{wt}(a \mid l), \mathrm{wt}\left(\bar{a} \mid \kappa_v(l)\right)\right\} \\ &\quad \leq \min\left\{\mathrm{wt}(a) + \rho_{m-1}, \mathrm{wt}(\bar{a}) + \rho_{m-1}\right\} \\ &\quad \leq \lfloor \Delta_m/2 \rfloor + \rho_{m-1} \end{aligned}$$

for all $a \in \mathbb{F}_2^{\Delta_m}$. Since the $\mathfrak{G}$-construction implicitly demands that $\Delta_m \leq d$, we may conclude that any coset leader of $\mathfrak{G}_m$ must have weight no greater than

$$\left\lfloor \frac{d}{2} \right\rfloor + \rho_{m-1}$$

which bounds the covering radius of $\mathfrak{G}_m$ and proves the lemma. ∎

With some combinatorial analysis, we can also establish a recursive *lower* bound on the covering radius of $\mathfrak{G}$-codes. Specifically, we shall make use of the following simple combinatorial lemma, which we provide without proof.

Lemma 2 *If $a > 3b > 0$ then*

$$\binom{a}{b} > \sum_{i=0}^{b-1} \binom{a}{i}. \tag{13}$$

Theorem 4 *For any $\mathfrak{G}$-family of codes whose seed code has length*

$$n_0 > 3\left\lfloor \frac{d-1}{2} \right\rfloor,$$

it necessarily follows that

$$\rho_m \;\geq\; \left\lfloor \frac{d-1}{2} \right\rfloor + \left\lfloor \frac{d-\rho_{m-1}}{2} \right\rfloor.$$

Proof: Following convention, we let $t = \lfloor (d-1)/2 \rfloor$ denote the number of errors that the codes of a $\mathfrak{G}$-family can correct. It is well-known that each vector in $\mathbb{F}_2^{n_{m-1}}$ of weight $\leq t$ must be a unique coset

leader for $\mathfrak{G}_{m-1}$. Moreover, using our assumption about n_0 (which also holds for n_m, since $n_m \geq n_0$), Lemma 2 implies that

$$\binom{n_{m-1}}{t} > \sum_{i=0}^{t-1} \binom{n_{m-1}}{i}. \tag{14}$$

The left-hand side of (14) is the number of vectors of weight t, while the right-hand side is the number of vectors of weight $< t$. Thus, by the pigeon-hole principle, there must be at least one coset leader $l \in \mathfrak{G}_{m-1}$ of weight t whose companion $\kappa(l)$ has weight at least t. Since $\Delta_m \geq d - \rho_{m-1}$ for any $\mathfrak{G}$-code, we may choose $a = 0^{\lfloor \Delta_m/2 \rfloor} 1^{\lceil \Delta_m/2 \rceil}$ to get:

$$\begin{aligned} \rho_m &\geq \min\{wt(\langle a \mid l \rangle), \mathrm{wt}(\langle \bar{a} \mid \kappa_v(l) \rangle)\} \\ &= \left\lfloor \frac{\Delta_m}{2} \right\rfloor + t \\ &\geq \left\lfloor \frac{d - \rho_{m-1}}{2} \right\rfloor + \left\lfloor \frac{d-1}{2} \right\rfloor. \end{aligned}$$

∎

The result in Theorem 4 also applies to all $\mathfrak{G}$-families trivially seeded with the code $\mathbb{S}_d$.

Lemma 3 *For any trivially seeded $\mathfrak{G}$-family of codes,*

$$\rho_m \geq \left\lfloor \frac{d-1}{2} \right\rfloor + \left\lfloor \frac{d - \rho_{m-1}}{2} \right\rfloor.$$

If a $\mathfrak{G}$-family is trivially seeded, then, necessarily, $n_0 = d$ and $\rho_0 = \lfloor \frac{d}{2} \rfloor$. Thus,

$$n_1 = n_0 + (d - \rho_0) = d + \left\lceil \frac{d}{2} \right\rceil > 3 \left\lfloor \frac{d-1}{2} \right\rfloor$$

so that Theorem 4 applies to the remaining codes in the family.

We now turn our attention to generating mappings which produce codes whose information rate is locally maximized. More specifically, for any code $\mathbb{C}$ with covering radius $\rho_{\mathbb{C}}$, we will consider only generating mappings f with the property that the Hamming distance from $\mathbb{C}$ to $f(\mathbb{C})$ is exactly $\rho_{\mathbb{C}}$. We will call such mappings *minimal generating mappings*, and the corresponding family of codes *minimal $\mathfrak{G}$-codes*, because they locally minimize length (and hence locally maximize information rate) for a given dimension and minimum-distance.

As an example, the generating mappings for the traditional lexicodes and the trellis-oriented lexicodes are both minimal. We may now easily strengthen Lemma 1 by observing in its proof that $\Delta_m = d - \rho_{m-1}$ for *minimal $\mathfrak{G}$-codes*.

Corollary 1 *For any minimal $\mathfrak{G}$-family of codes,*

$$\rho_m \leq \left\lfloor \frac{d + \rho_{m-1}}{2} \right\rfloor.$$

The covering radius bounds we have developed for $\mathfrak{G}$-codes translate naturally to length bounds.

4.1.2 Bounding Length

By summing Theorem 4 over many iterations, we may obtain a lower bound on the length n_m of the m-th code of any minimal $\mathfrak{G}$-family.

Corollary 2 *For any trivially seeded, minimal $\mathfrak{G}$-code,*

$$n_m \leq \left(m + \frac{1}{2}\right)\frac{d+4}{3} - \frac{2}{3}.$$

Proof: Consider summing a weaker inequality obtained from Theorem 4 by eliminating the floor function. Summing over iterations $1 \ldots m$ of the $\mathfrak{G}$-construction we get:

$$\begin{aligned}\sum_{i=1}^{m} \rho_i &\geq \sum_{i=1}^{m} \left(d - \frac{\rho_{i-1}}{2} - 2\right) \\ &\geq m(d-2) - \frac{1}{2}\sum_{i=0}^{m-1} \rho_i.\end{aligned}$$

Noting that for trivially seeded codes, $\rho_0 = \lfloor \frac{d}{2} \rfloor$ and $\rho_m \geq 0$, we may reduce this to:

$$\frac{3}{2}\sum_{i=1}^{m} \rho_i \geq m(d-2) - \frac{1}{2}(\rho_0 - \rho_m) \geq m(d-2) - \frac{d}{4}.$$

Furthermore, since we are dealing with minimal $\mathfrak{G}$-codes,

$$n_m = \sum_{i=0}^{m} (d - \rho_i) = md - \sum_{i=1}^{m} \rho_i$$

so that we may now conclude the proof:

$$\begin{aligned}n_m &= md - \sum_{i=1}^{m} \rho_i \\ &\leq md - \frac{2}{3}\left(m(d-2) - \frac{d}{4}\right) \\ &\leq \frac{m}{3}(d+4) + \frac{d}{6}.\end{aligned}$$

∎

In the case of a $\mathfrak{G}$-family seeded by a non-trivial code of length n_0 and covering radius ρ_0, Corollary 2 may be easily generalized to

$$n_m \leq n_0 + \frac{m(d+4) + \rho_0}{3} + \frac{d}{6}.$$

Clearly, Corollary 2 applies to all lexicodes and trellis-oriented lexicodes. It is a asymptotically tighter than the similar bound given by Brualdi and Pless [3, Theorem 3.5]:

$$k \geq \begin{cases} n - 2 - \lfloor \log_2(n-1) \rfloor & \text{if } d = 4, \\ \left\lfloor \frac{4n-d-12}{2d-4} \right\rfloor & \text{if } d \equiv 0 \pmod 4, d \neq 4, 8, \\ \lfloor \frac{n}{3} \rfloor & \text{if } d = 8,\ n > 18, \\ \left\lfloor \frac{4n-d-14}{2d-4} \right\rfloor & \text{if } d \equiv 2 \pmod 4. \end{cases} \tag{15}$$

Specifically, Corollary 2 asymptotically binds $k \geq 3\frac{n}{d}$ whereas Equation (15) binds $k \geq 2\frac{n}{d}$. Nevertheless, both of these bounds are weak as illustrated by Appendix .1.

5 Computations

Theorem 3 can be directly translated into an algorithm that computes $\mathfrak{G}$-families. The algorithm simply computes the coset leaders of $\mathfrak{G}$-codes in the family, from which the remaining parameters may be easily deduced. This computation scheme has the advantage of being exponential in the co-dimension (*i.e.,* the dimension of the dual code) rather than the dimension or the length. Thus, this algorithm is efficient for high-rate codes, and it is presented in Algorithm 1.

Algorithm 1 *Given an iteration m, a generating mapping $f(\cdot)$, and a seed code $\mathbb{C}$, the following algorithm will compute the code $\mathfrak{G}_m(f, \mathbb{C})$:*

1. Record the cosets of the seed code $\mathfrak{G}_0$
2. **for** i from 1 to m **do**
3. compute $v = f(\mathfrak{G}_{i-1})$
4. **for** each coset leader λ of $\mathfrak{G}_{i-1}$ **do**
5. compute $\kappa_v(\lambda)$, which is the companion of λ
6. **for** each $a \in \langle 0 | \mathbb{F}_2^{\Delta_i - 1} \rangle$ **do**
7. **if** $\mathrm{wt}(\langle a|\lambda \rangle) < \mathrm{wt}(\langle 1^{\Delta_i} + a | \kappa(\lambda) \rangle)$ **then**
8. $[a|\lambda]$ is a coset leader of $\mathfrak{G}_i$
9. **else**
10. $\langle (1^{\Delta_i} + a) | \kappa_v(\lambda) \rangle$ is a coset leader of $\mathfrak{G}_i$
11. Record the covering radius ρ_i of $\mathfrak{G}_i$ from the newly computed coset leaders.

In Algorithm 1 we reuse our earlier notation that $\Delta_i = d - \mathrm{wt}(f(\mathfrak{G}_{i-1}))$. Note that we assume that the coset leaders of the seed code are known or trivially derivable, as is often the case. The correctness of the algorithm follows immediately from Theorem 3. Moreover, we can also compute the running time and space of this algorithm in terms of $\widehat{m} \stackrel{\mathrm{def}}{=} n_m - m$, the co-dimension of the m-th $\mathfrak{G}$-code,

Lemma 4 *Using an oracle for computing the generating mapping, Algorithm 1 runs in space $O(2^{\widehat{m}})$ and time $O(n_m m 2^{\widehat{m}})$.*

An oracle is simply a black box that computes a given function in constant time. In this case, we assume the existence of an oracle for computing the generating mapping because the complexity of such a computation can vary greatly among mappings. The proof of Lemma 4 follows from a straightforward analysis of the pseudo-code for Algorithm 1.

For the specific case of distance 4 lexicodes, this algorithm is particularly fast, since Brualdi and Pless [3, Thm.3.5] show that, under these circumstances:

$$m = n_m - 2 - \lfloor \log_2(n_m - 1) \rfloor \tag{16}$$

so that Lemma 4 reduces to time $O(n_m m \log(n_m))$ and space $O(n_m)$. This is not surprising given that the distance 4 lexicodes are simply shortenings of the extended Hamming codes [5].

The analysis of Algorithm 1 assumes an oracle computation of the generating mapping. For the case of lexicodes and trellis-oriented lexicodes, however, Method 1 provides an efficient means of computing this mapping in time $O(n_m \times \#\mathtt{COSET}_m)$ and $O(n_m)$ space, where $\#\mathtt{COSET}_m$ represents the number of cosets in the m-th corresponding $\mathfrak{G}$-code. As in the case of lexicodes and trellis-oriented codes, the overhead of computing $f(\mathfrak{G}_{i-1})$ is usually eclipsed by the running time of the remainder of the algorithm.

Corollary 3 *Algorithm 1 requires time $O(n_m m 2^{\widehat{m}})$ and space $O(2^{\widehat{m}})$ to compute lexicodes and trellis-oriented lexicodes.*

The complexity of Algorithm 1 is thus bounded by the co-dimension of the code and by the difficulty of computing the generating mapping. Under practical conditions, we were able to compute lexicodes well beyond length 44 initially reported in [5].

In addition, we were able to construct trellis-oriented codes with code parameters rivaling those of lexicodes, but with much better trellis state complexities. As an example, we generated a trellis-oriented $\mathfrak{G}$-code with code parameters similar to lexicodes, but with better state complexity than the corresponding BCH codes heuristically minimized in [7]. These result were predicted by [7] and are demonstrated in Table 4.

TRE:	0:1:2:3:4:5:6:6:7:8:9:8:9:8:7:6:7:6:6:6:5:5:4:3:4:4:4:3:3:2:1:0
BCH:	0:1:2:3:4:5:6:6:7:8:9:8:9:8:7:6:7:8:9:8:9:8:7:6:7:6:5:4:3:2:1:0

Table 4: A comparison of the trellis state complexities of the $(31, 16, 7)$ trellis-oriented lexicode (TRE) and a $(31, 16, 7)$ extended BCH code (BCH) heuristically minimized in [7] .

Finally, the trellis state bounded codes that we have computed show improvements, for various code parameters, over the state-bounded codes computed in [15] using different techniques.

6 Applications and Conclusions

The $\mathfrak{G}$-construction lends itself to several applications in code design, some of which we describe in this section.

6.1 Code engineering

The $\mathfrak{G}$-families produced under various generating mappings lend themselves to engineering a code for a specific purpose under specific constraints. For example, if given a dimension k and a desired capability of correcting t errors, the mapping f_{lexi} can be used to generate an appropriate, approximately optimal code.

For a heuristic decoding optimization, the mapping f_{trelli} may be used to generate an approximately optimal "trellis-oriented" code. These codes may be constrained further, however, if there is a particular memory limit m on the decoding space. In this case, the mapping f_{state} may be used to get a reasonably short code satisfying dimension, error-correcting capability, and decoding memory constraints. Alternatively, one might wish to find a family of good codes with a bounded decoding time, in which case the mapping f_{decoding} would be appropriate. In each of these cases, the $\mathfrak{G}$-construction often provides a code that meets the desired constraints and which, empirically, tends to have good parameters.

The appendix lists various $\mathfrak{G}$-codes, demonstrating their use in this application. For example we see in Appendix .1 that though the trellis-oriented codes have almost identical code parameters to their lexicode counterparts, they tend to have a much better trellis complexity. As another example, we can see in Appendix .2 that by sacrificing about 12% of the information rate, we can reduce decoding complexity from the 1024 states in the $(38, 21, 8)$ lexicode to a mere 64 states in the $(43, 21, 8)$ state-bounded code. Such a low memory requirement would be ideal for a *VLSI* chip or a smart card.

6.2 Code improvement

The $\mathfrak{G}$-construction also enables us to transform a given code into a similar code with (often) a better trellis complexity. Theorem 2 assures us that $f_{\text{trelli}}(\cdot)$ is a locally optimal method of extending an arbitrary subcode if we seek to maximize information rate and minimize decoding complexity. Thus, we have a simple, greedy method in which we may attempt to improve a given code $\mathbb{C}$.

Specifically, we may arbitrarily delete a generator of $\mathbb{C}$ and replace it with a trellis-oriented generator. Because of Theorem 2 we know that the resulting code will have trellis complexity and code parameters (e.g., length, minimum distance) no worse than the original code.[2]

In fact, this method may be applied to several generators: shorten $\mathbb{C}$ by deleting k generators; replace the generators with k trellis-oriented generators. This allows us to create an amalgam of the original code and a trellis-oriented code. Though the resulting code is not guaranteed to match the decoding performance of our original code, it often performs much better as we see in the following example.

Example 1 *Consider the length 31, dimension 16, triple error-correcting BCH code generated by the polynomial* $g(x) = 1 + x + x^2 + x^3 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{15}$. *It requires* $2^{15} = 32,768$ *trellis states and* $262,139$ *steps for Viterbi decoding. Table 5 shows the dramatic improvement of this code as more generator vectors are replaced with trellis-oriented generators.*

k	State complexity	Decoding complexity	Generation time
1	2^{15}	262,139	26:36
2	2^{15}	262,139	22:11
3	2^{13}	94,203	18:30
4	2^{13}	54,659	24:45
5	2^{11}	36,355	20:25
6	2^{12}	43,779	17:17
7	2^{12}	37,251	14:59
8	2^{11}	32,333	14:49
9	2^{12}	39,373	12:50
10	2^{12}	40,159	11:41
11	2^{12}	37,647	10:34
12	2^{11}	26,303	11:22
13	2^{10}	16,611	10:12
14	2^{10}	16,147	9:39
15	2^{10}	13,603	9:41
16	2^{9}	4,907	1:07

Table 5: Improving a $(31, 16, 7)$ BCH code by replacing k of its generators with trellis-oriented generators. The trellis state complexity and Viterbi decoding complexity of the intermediate $(31, 16, 7)$ codes is noted, as is the time (in minutes) needed to generate them on a Sun Ultra-Sparc 1.

We note that both the state and decoding complexities generally decrease as k increases. Moreover, the computation time also generally decreases because of the decreasing size of the seed code.

6.3 Practical considerations

In practice, computing the $\mathfrak{G}$-construction using Algorithm 1 is only feasible for high-rate codes because the algorithm needs to maintain a list of coset leaders at each iteration.

Another problem with the $\mathfrak{G}$-construction is that it is theoretically possible for bounding to be so severe as to halt the construction when, for example, $f(\mathbb{C})$ does not return a vector for a particular code $\mathbb{C}$. Though we have not seen this with trellis state bounding (*i.e.,* using the mapping f_{state}), we have seen indications that this can occur with f_{decoding}. Of course, certain constraints will simply not correspond to any existing linear code, so this problem is inherent to code design.

[2]We do have to take some care that when we delete the generator the resulting code is non-trivial. If the generator matrix of the resulting code has an all-zero column, making the code trivial, we may simply delete the offending column.

6.4 Future directions

Many questions remain unanswered in this work. First, it is still not clear why lexicodes, or even trellis-oriented $\mathfrak{G}$-codes, have such good code parameters. Second, we suspect that the bounds on parameters of lexicodes can be improved by a more sophisticated count of the worst-case companion pairings. Furthermore, one should note that the exponential time and space bound (with respect to co-dimension) of Algorithm 1 may be improved by various approximation techniques. The actual continuation of the algorithm depends only on the properties of a few coset leaders, and a heuristic approach to choosing these leaders might work well. Speeding up Algorithm 1 would allow for the design of lower rate codes with good trellis decoding properties. Finally, the iterated method in which lexicodes are constructed is reminiscent of concatenated codes, and the connection to concatenated code design deserves further attention.

Acknowledgments

We are grateful to G. David Forney, Jr. for suggesting some of the problems addressed in this work. We would also like to thank Frank Kschischang for sending us preprints of his papers and Alexander Vardy for motivating this work, providing important feedback, and suggesting Methods 1 and 2. Finally, we would like to thank the anonymous referees for their valuable suggestions for tightening the presentation of this paper.

References

[1] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, 20:284–287, 1974.

[2] A.E. Brouwer and T. Verhoeff. An updated table of minimum-distance bounds for binary linear codes. *IEEE Transactions on Information Theory*, 39:662–677, 1993.

[3] R.A. Brualdi and V.S. Pless. Greedy codes. *Journal of Comb. Th. (A)*, 64:10–30, September 1993.

[4] J.H. Conway. Integral lexicographic codes. *Discrete Math*, 83:219 – 235, 1990.

[5] J.H. Conway and N.J.A. Sloane. Lexicographic codes: error-correcting codes from game theory. *IEEE Transactions on Information Theory*, 32:337–348, 1986.

[6] R.L. Graham and N.J.A. Sloane. On the covering radius of codes. *IEEE Trans. Inf. Theory*, 31:385–401, 1985.

[7] F.R. Kschischang and G.B. Horn. A heuristic for ordering a linear block code to minimize trellis state complexity. In *Proc. 32-nd Annual Allerton Conference on Communications, Control and Computing*, pages 75–84, Monticello, IL, September 1994.

[8] F.R. Kschischang and V. Sorokine. On the trellis structure of block codes. *IEEE Transactions on Information Theory*, 41:1924–1937, 1995.

[9] V.I. Levenšteĭn. A class of systematic codes. *Soviet Math. Dokl.*, 1(1):368–371, 1960.

[10] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, New York, 1977.

[11] R.J. McEliece. On the BCJR trellis for linear block codes. *IEEE Transactions on Information Theory*, 42:1072–1092, 1996.

[12] D.J. Muder. Minimal trellises for block codes. *IEEE Transactions on Information Theory*, 34:1049–1053, 1988.

[13] Ari Trachtenberg. *Graph-based decoding of error-correcting codes.* PhD thesis, University of Illinois at Urbana/Champaign, January 2000.

[14] A. Vardy. Trellis structure of codes. In V.S. Pless and W. Cary Huffman, editors, *Handbook of Coding Theory.* Elsevier Science Publishers, Amsterdam, 1998.

[15] S. Zhang. Design of linear block codes with fixed state complexity. Master's thesis, University of Toronto, 1996.

We show data gleaned from the implementation of many of the algorithms in this paper. More complete simulation results are available in [13].

.1 Trellis-Oriented $\mathfrak{G}$-codes

Table 6 lists the properties of some distance-8 lexicodes and trellis-oriented $\mathfrak{G}$-codes. Over a binary field, lexicodes with odd minimum distance are simply punctured lexicodes with even minimum distance, so it is redundant to list their properties [5]. The minimum distance 4 lexicodes are all extended Hamming codes or shortenings thereof [5]. The minimum distance 6 and 8 codes we have computed also have optimal error-correction capability in the sense that each has the best known minimum distance for its length and dimension as compared to [2]. We can see that the trellis-oriented codes almost always have lower decoding complexity than the corresponding lexicodes; however, there are small pockets, where the trellis-oriented codes have higher state complexity and/or decoding complexity than lexicodes.

.2 State Bounded $\mathfrak{G}$-Codes

We depict the parameters of distance 6 $\mathfrak{G}$-codes with bounded trellis state complexities.

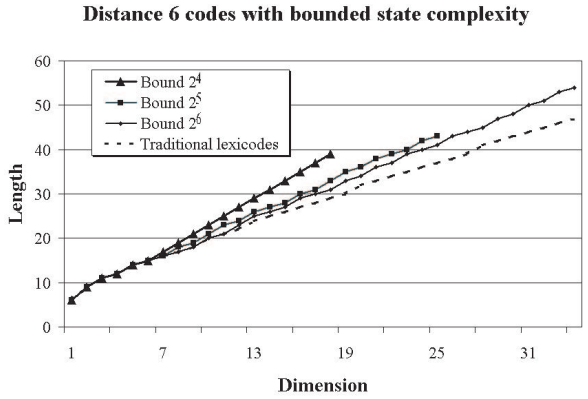

Figure 3: The effects of bounding the *state complexity* of distance 6 $\mathfrak{G}$-codes with various constants.

.3 Bounding Decoding Complexity

Figure 4 shows the results of applying Viterbi decoding constraints to $\mathfrak{G}$-codes. Specifically, it shows the decoding complexity achieved under linear and quadratic decoding bounds. The degradation in code length compared to the standard lexicodes is modest for these examples.

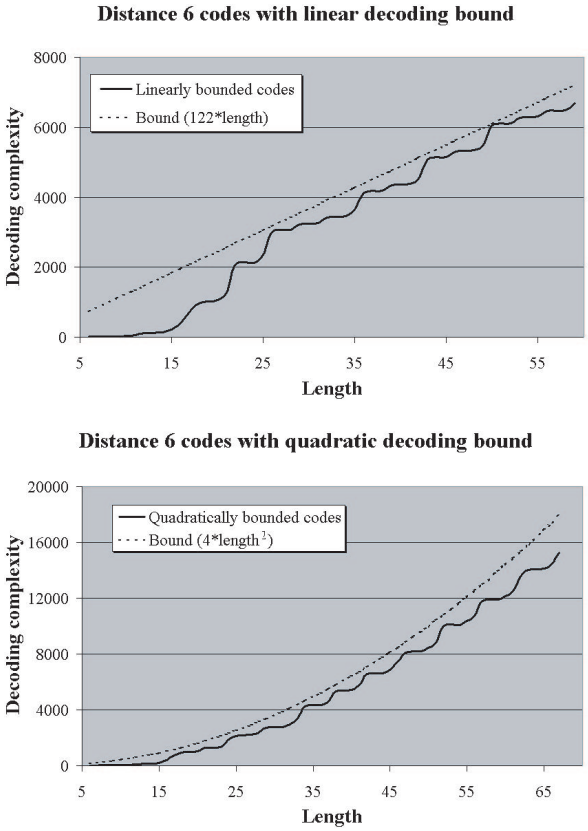

Figure 4: The effects of bounding the decoding complexity of distance 6 $\mathfrak{G}$-codes with a *linear* function (top)and a *quadratic* function (bottom).

Table 6: Parameters of $\mathbf{d = 8}$ codes. We display the following parameters for lexicodes and trellis-oriented $\mathfrak{G}$-codes: length, dimension, state complexity and Viterbi decoding complexity with the BCJR trellis.

Dimension	*Standard*	*Trellis-oriented*	*Standard*	*Trellis-oriented*	*Standard*	*Trellis-oriented*
	Length	*Length*	*States*	*States*	$2\|E\| - \|V\| + 1$	$2\|E\| - \|V\| + 1$
1	8		1		17	
2	12		2		35	
3	14		3		71	
4	15		4		143	
5	16		4		195	
6	18		5		341	
7	19		6		647	
8	20		6		779	
9	21		7		1,547	
10	22		8		2,395	
11	23		9		4,219	
12	24		9		4,475	
13	28		9		4,529	
14	30		9		4,777	
15	31		9		5,463	
16	32		9		5,515	
17	34		9		7,645	*5,693*
18	35		9		12,671	*6,143*
19	36		9		12,803	*6,275*
20	37		10	*9*	24,267	*7,691*
21	38		10	*9*	25,115	*8,539*
22	39		10	*9*	26,939	*10,363*
23	40		10	*9*	27,195	*10,619*
24	42		10		31,805	*17,853*
25	43		11		41,791	*33,087*
26	44		11		41,987	*33,283*
27	45		12		65,227	
28	46		12		67,163	
29	47		12		72,571	*78,203*
30	48		12		72,827	*78,459*
31	49	*50*	13	*12*	135,611	*80,317*
32	50	*51*	13	*12*	169,019	*87,103*
33	51	*52*	13	*12*	248,123	*88,643*
34	52	*53*	13		248,507	*137,547*
35	53	*54*	14	*13*	427,579	*138,331*
36	54	*55*	14	*13*	431,419	*142,203*
37	55	*56*	14	*13*	442,171	*142,459*
38	56	*57*	14		442,555	*274,875*
39	58		14		487,997	*308,283*
40	59		14		628,031	*457,019*
41	60		14		628,227	*460,091*
42	62		14		629,197	*460,861*
43	63		14		631,903	*464,703*
44	64		14		632,035	*464,899*
45	65		15	*14*	1,263,819	*581,835*
46	66		15		1,287,195	*1,053,275*