The Trellis Complexity of Block and Convolutional Codes

Thesis by

Wei Lin

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1997

(Submitted February 26, 1997)

**To my parents
and brother Kui**

Acknowledgements

It is my fortune and honor to read the degree of Doctor of Philosophy under the guidance of Professor Robert J. McEliece, whose knowledge and wisdom provided me inspiration and made this research possible. Working with him was a pleasure and a great learning experience. I am greatly indebted to him for his unconditioned support and enthusiastic encouragement, without which I could not overcome all difficulties and finish this thesis. I learned from him not only the ways of doing research but also the spirits of being a scholar. A world-class scholar and philosophically-correct Ph.D. advisor, Bob McEliece deserves my life-long respect and appreciation which cannot be expressed fully in language.

I would like to thank Drs. Aaron B. Kiely, Samuel J. Dolinar and Laura L. Ekroot from JPL/NASA, Pasadena. Many suggestive email communications and discussions with them gave me a lot of insights into the problems in Chapter 1 and 2. I would like to thank Aaron for his careful review and excellent comments, which improved the quality of this thesis much. I would also like to thank Professor Paddy G. Farrell in the Electrical Engineering Department, University of Manchester, U.K., for several discussions from which I benefited a lot. I would also like to thank Ø. Ytrehus from the Department of Informatics, University of Bergen, Norway, for the preprint of his paper and useful discussion regarding the problems in Chapter 5. Private communications with Professor Alex Vardy in the Department of Electrical Engineering, UIUC, were also very helpful.

I would like to thank the members of the dissertation defense committee: Professors Robert J. McEliece, P. P. Vaidyanathan, Richard M. Wilson, and Drs. Aaron B. Kiely and Samuel J. Dolinar.

I also would like to thank Pacific Bell, National Science Foundation (NCR-9505975) and Air Force Office of Scientific Research (AFOSR F4960-94-1-005) for their financial support.

I would like to thank my friends (too many to list) in Caltech. Xin An, Chang Liu, Tao Tan from Electrical Engineering, Sheng Wu from Chemistry, Yang Liu from Biology and Qiao Lin from Civil Engineering have been very good friends and have always offered warm support whenever requested. I would like to thank the friends I lived with in the graduate house where I stayed for four years. I would like to thank my fellow groupmates and good friends: Mohamed-Slim Alouini, Jung-Fu Cheng, Dr. Sanjeev K. Deora, Bahadir Erimli, Masayuki Hattori, Gavin Horn, Hui Jin, Lifang Li, Hongyu Piao, Aji Srinivas, Meina Xu and Zhong Yu. I would also like to thank my colleagues in Tsinghua University, Beijing, China. They made my undergraduate life joyful and colorful.

I would like to thank Lilian Porter, our secretary, for her excellent professional services on office management and letter preparation. I also would like to thank Robert Freeman, our system administrator, for his professional services on keeping the computers working perfectly.

Finally, I would like to thank my mother and father, my aunt and my brother. Without their meticulous early education and guidance, I could not have the chance to be here. It is their unlimited love that keeps me going.

Abstract

This thesis concerns the computational complexity of high performance decoding algorithms. The primary objective is to design the most efficient maximum–likelihood (ML) decoders for both block codes and convolutional codes. By *efficient*, we mean an implementation of ML decoding algorithms on trellises that minimize the computational complexity (the total number of additions and comparisons). Trellises are graph representations of codes. Since decoding complexity is completely determined by the particular trellis employed, the problem is equivalent to constructing the minimal trellis (one that has the minimum number of edges, vertices and bifurcations) for a given code.

There are four parts to this research. The first problem we attacked was to construct the minimal trellises for block codes over the coordinate permutations. The related problem of finding a coordinate permutation that minimizes the number of vertices at a given depth in the minimal trellis for a binary linear block code [36] has been proven to be NP–complete. Our approach was based on the concept of *span* of the generator matrices, which connects the code parameters and the trellis complexity. New bounds on measures of trellis complexity such as $|E|$ (the total number of edges) and $|V|$ (the total number of vertices) were obtained from the analysis of the *span distribution*. Aiming to minimize the total span in a generator matrix, an efficient, effective "divide–and–conquer" algorithm and variants were proposed to search for the optimal or a good trellis structure for any block code. For example, it took about 12 minutes on a Sun Sparc Station 20 to find one optimal permutation for the [48,24,12] QR code from 48! candidates.

By introducing the concept of *trellis–canonical* generator matrices and a simple algorithm to compute one, we developed a general theory of minimal trellises for convolutional codes. In this theory, *punctured convolutional codes* no longer have to be treated as special cases. By then, the minimal trellises for block and convolutional codes were both well–defined. This allowed one to make a direct performance–complexity comparison between block codes and convolutional codes.

The ratio of performance (measured by the asymptotic coding gain-ACG) and complexity (measured by the logarithm trellis edge complexity-LTC) defines the coding efficiency. By means of the span analysis, we also proved a universal lower bound on the complexity to performance ratio. It implies that $\frac{LTC}{ACG}$ can never be smaller than 1 for any code, block or convolutional. In some cases, the bound is optimal or asymptotically optimal. The study suggests that optimal codes in terms of minimum distance or free distance do not necessarily offer the best coding efficiency.

The last problem addressed in this dissertation is the implementation of maximum–likelihood decoding and the computational complexity for convolutional codes. By combining the optimal sectionalization technique [45] with minimal trellis theory, a low complexity hybrid decoding algorithm was developed. For some partial unit memory convolutional codes, its decoding complexity is significantly superior to other known algorithms. There are two components of the computational complexity. One is the edge metric computation cost θ. We proved a lower bound on θ which is independent of the computation mechanism (sequential or parallel). This bound is optimal in some cases. The other is the cost of the state metric updating which is inferable from the trellis structure. This sets a lower bound on the computational complexity for any implementation.

Finally we give a general review of research activities on this subject and present a list of open problems.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Trellis evolution

In this chapter, we give a brief introduction to the minimal trellis theory for block codes. Facts to be used in the following chapters are presented without proofs. For details and proofs, please read the partially tutorial paper by A. Kiely *et al.* [40].

The idea of representing block codes by *trellises* was originally introduced by Bahl *et al.* [7]. Such trellises can be used as templates for encoding and decoding. There are several possible definitions[1], but we shall use the following:

Definition: A *trellis* for a block code C over the symbol set $\mathcal{A}$ of q elements[2] is a triple $\mathcal{T} = (V, E, \rho)$ satisfying:

1. (Decomposition of V) $V = \bigcup_{i=0}^{n} V_i$,

   where $|V_i| > 0$ for any i, $|V_i \cap V_j| = 0$ if $i \neq j$. An element of V is called a *vertex*. We require $|V_0| = |V_n| = 1$. The unique vertex in V_0 is the *root* of the trellis. The unique vertex in V_n is the *sink* of the trellis.

2. (Decomposition of E) $E = \bigcup_{i=1}^{n} E_i$,

   where $|E_i| > 0$ for any i, $|E_i \cap E_j| = 0$ if $i \neq j$. An element of E is called an *edge*.

3. $\rho : U \overset{1\ to\ 1}{\longmapsto} E$, where $U \subseteq V \times V \times \mathcal{A}$ such that $\rho^{-1}(E_i) \subseteq V_{i-1} \times V_i \times \mathcal{A}$ and $|\rho^{-1}(E_i)| = |E_i|$. For any $x \in E$, let $\rho^{-1}(x) = (a, b, c)$, then a, b, c are called the tail, the head and the label of x, respectively.

4. A sequence $d_1 d_2 d_3 \ldots d_n$ over $\mathcal{A}$ is a codeword of C if and only if there is a

[1]For example, the tailbiting trellises [75] are different from our definition.

[2]In this thesis, we assume $\mathcal{A} = GF(2)$, unless otherwise specified. However, many of the generalizations to $GF(q)$ are straitforward.

sequence $x_1x_2x_3\dots x_n$, where $x_i \in E_i$, such that $a_1 \in V_0$, $b_i = a_{i+1}$, $b_n \in V_n$ and $c_i = d_i$, where $\rho^{-1}(x_i) = (a_i, b_i, c_i)$. The sequence $x_1x_2x_3\dots x_n$ is called a path and $c_1c_2c_3\dots c_n$ is the label of the path.

5. No two distinct paths have the same label. For any $x \in E$, there is at least one path containing x. For any $v \in V$, there is at least one path containing v.

Merging is the primitive operation to produce a new trellis from an old one. It includes two processes. *Vertex merging:* For two distinct elements of V_i, u and v, a new triple $\mathcal{T}^*$ is obtainable by deleting v, replacing every v in $\rho^{-1}(E)$ by u. *Edge merging:* For two distinct elements of E_i, if they have identical tail, head and label under ρ, combine them as one edge. Repeat edge merging until $|\rho^{-1}(E_i)| = |E_i|$. Edge merging is only possible after vertex merging.

After merging operations, the new trellis may not be one for the same code. In this thesis, we are only interested in "legal" mergings that do not change the code.

Definition: A trellis $\mathcal{T}$ is said to be *redundant* if there exist vertex or edge mergings such that the merged triple $\mathcal{T}^*$ is a trellis for the same code. On the other hand, an *efficient trellis* is one without redundancy (which is in general not unique).

For an efficient trellis, it is easy to see that $|V_i| \leq \min(q^{n-i}, q^i)$.

If we draw a picture for a trellis, it is a labeled digraph. This justifies our using graph terminologies such as edge, vertex, root, tail. The above definition excludes the tree structure as a legal trellis. However, any trellis is obtainable from a tree by mergings.

Example 1.1 *The [6,3,3] shortened Hamming code*

Consider the [6,3,3] code with generator matrix

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

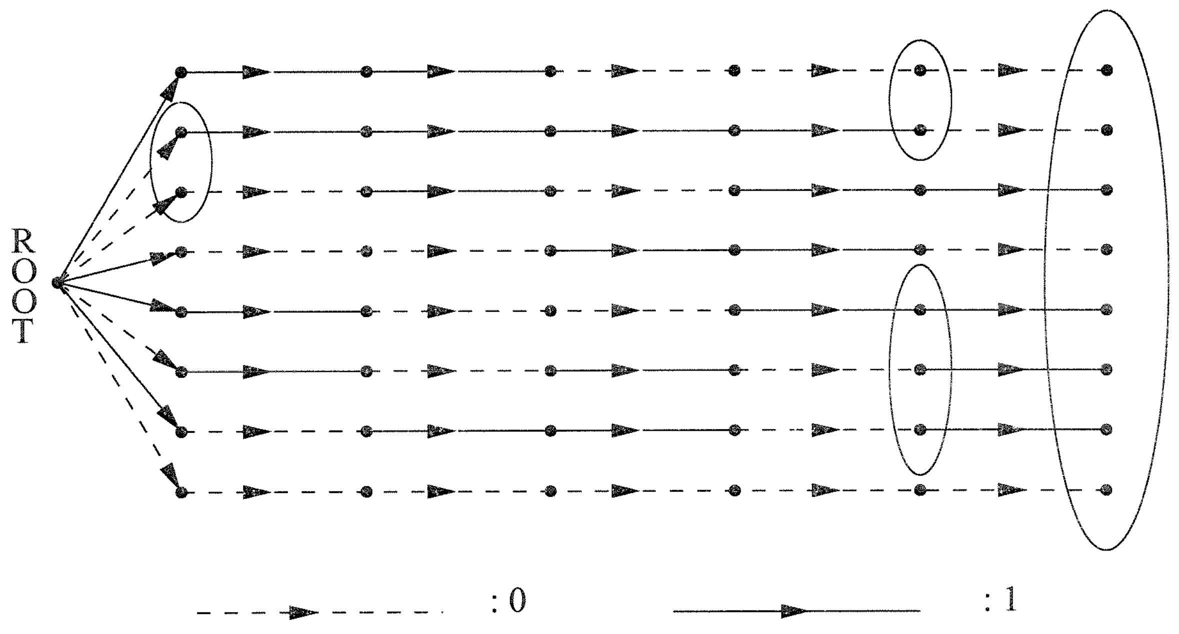

Figure 1.1: The tree structure for the [6,3,3] code. Those vertices in the same ellipse can be merged.

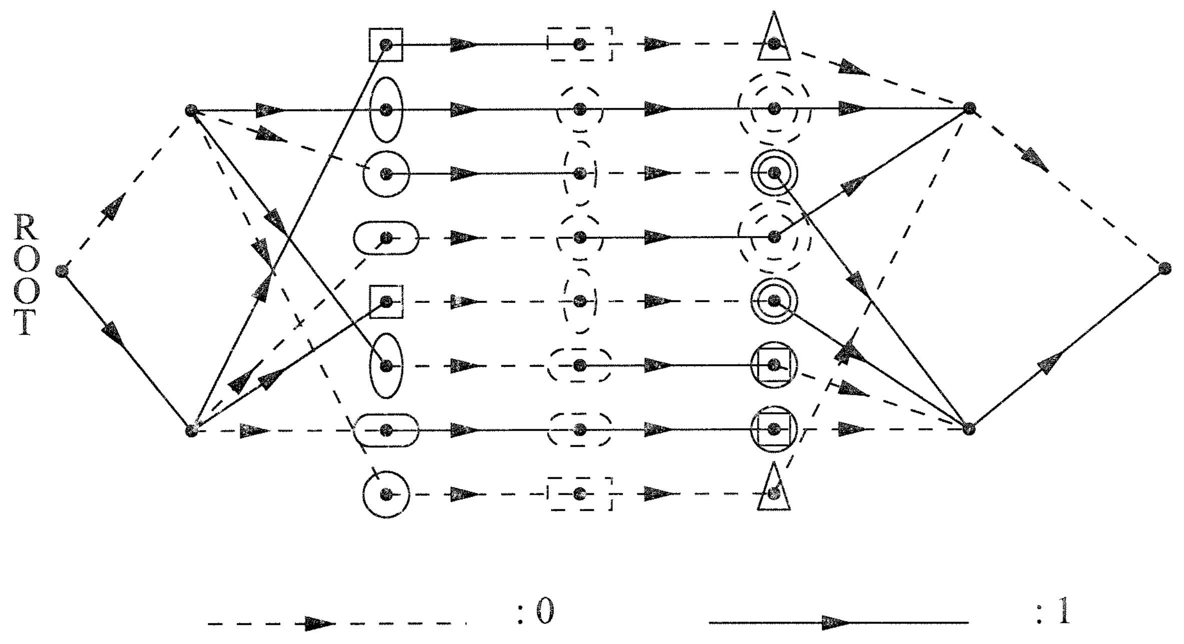

Figure 1.2: One trellis for the [6,3,3] code. Those vertices in identical frame can be merged. For example, the two vertices in squares are to be merged.

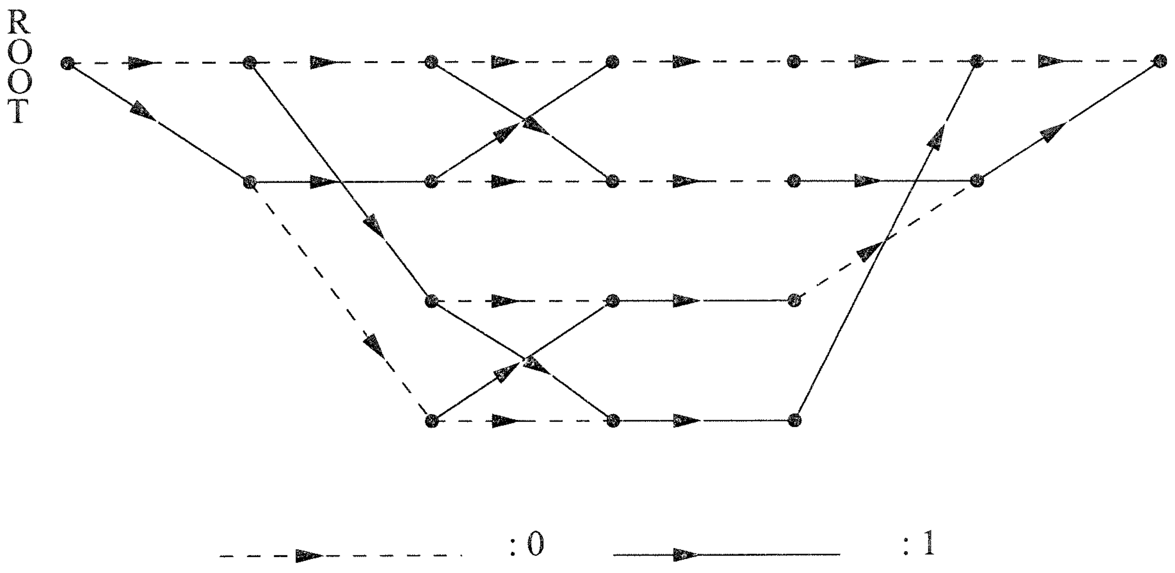

Figure 1.3: An efficient trellis for the [6,3,3] code.

Figure 1.1, 1.2 and 1.3 depict how an efficient trellis is evolved from the tree structure. It is further noted that it contains at most four primitive structures: extension, expansion, merger and butterfly, as shown in Figure 1.4 [40]. (This is true for binary linear block codes, but not true in general for nonlinear codes.)

The evolution algorithm for binary code:

```
Begin:
    Construct the tree structure.
    Merge all terminal vertices and do edge merging.
    WHILE (The trellis has redundancy)
    {
        Do merging operations.
    }
End.
```

The evolution algorithm will find an efficient trellis for any linear or nonlinear binary code. The refined evolution algorithm can find the minimal trellis (to be

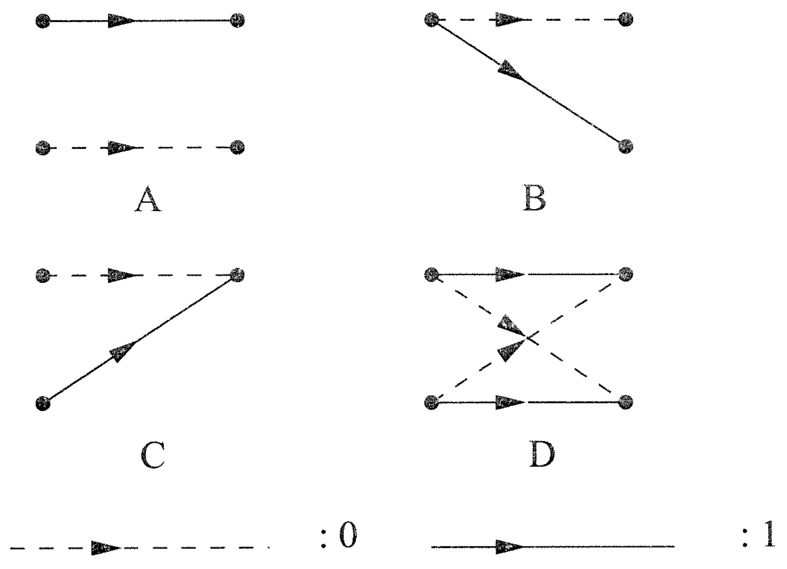

Figure 1.4: The primitive structures of trellises for block codes. A: Extension. B: Expansion. C: Merger. D: Butterfly.

defined) for both linear and nonlinear code[3] [51].

In the following discussion, we assume codes are linear, though some results are applicable to nonlinear codes.

1.2 Trellis complexity measures

There are a number of different ways to measure the complexity of a trellis.

- The total number of edges $|E|$, the total number of vertices $|V|$ and the total number of bifurcations (mergers) M.
- The edge dimension profile $e_i = \log_2 |E_i|$ and the vertex dimension profile $s_i = \log_2 |V_i|$.
- The maximum vertex dimension $s_{max} = \max(s_i)$. The maximum vertex dimension $e_{max} = \max(e_i)$.
- The total span of a MSGM m_G (See Chapter 2).

[3]The minimal trellis for nonlinear codes is not unique.

These complexity measures are related via the following equations.

$$M = |E| - |V| + 1, \tag{1.1}$$

$$|E| = \sum_{i=1}^{n} 2^{e_i}, \tag{1.2}$$

$$|V| = \sum_{i=0}^{n} 2^{s_i}, \tag{1.3}$$

$$m_G = \sum_{i=1}^{n} e_i = \sum_{i=0}^{n} s_i + k. \tag{1.4}$$

1.3 The minimal trellis

Among all possible trellis representations for a code, the one minimizing $|E|$ is the minimal trellis.

Theorem 1.1 *A trellis is minimal if and only if it is efficient* [51].

In [58], McEliece showed how to construct a minimal trellis from a generator matrix. This trellis has $n+1$ levels of vertices and n levels of edges. The vertex levels are called depths and are numbered from 0 to n; the edge levels are called stages and are numbered from 1 to n. The ith stage of edges corresponds to the ith stage of encoding or decoding using the trellis as well as the ith column in the generator matrix. It was also shown that the minimal trellis simultaneously minimizes $|E|$, $|V|$, M and s_{max} [58, 80].

Let V_n be the n–dimensional vector space over the finite field $GF(2)$. Given a vector $x = (x_1, x_2, x_3, \ldots, x_n) \in V_n$, let

$$L_x = \min\{i : x_i = 1\}, \tag{1.5}$$

$$R_x = \max\{i : x_i = 1\}. \tag{1.6}$$

Then $m_x = R_x - L_x + 1$ is called the *span* of x. $[L_x, R_x]$ is the *support* of x. Any entry in the support is *active*. The support of a code is the union of supports of all codewords.

Definition: A generator matrix $G^T = (r_0, r_1, r_2, \dots r_{k-1})^T$ over V_n for a linear $[n,k,d]$ code is a *minimal span generator matrix* (MSGM) if and only if it satisfies the "LR" properties, defined as follows:

$$L_{r_i} \neq L_{r_j} \;\text{ if }\; i \neq j, \tag{1.7}$$
$$R_{r_i} \neq R_{r_j} \;\text{ if }\; i \neq j. \tag{1.8}$$

The number $m_G = \sum_i m_{r_i}$ is the *span* of G.

Definition: At depth i, the ith past code P_i is the set of codewords whose support are in $[1,i]$; the ith future code F_i is the set of codewords whose support are in $[i+1,n]$.

These two sequences of subcodes are nested in the following manner:

$$\{\mathbf{0}\} = P_0 \subseteq P_1 \subseteq P_2 \subseteq \cdots \subseteq P_n = C, \tag{1.9}$$
$$\{\mathbf{0}\} = F_n \subseteq F_{n-1} \subseteq F_{n-2} \subseteq \cdots \subseteq F_0 = C. \tag{1.10}$$

The dimensions of these codes are denoted by:

$$p_i = \dim(P_i), \tag{1.11}$$
$$f_i = \dim(F_i), \tag{1.12}$$

and can be easily read from the MSGM: f_i is the number of rows for which the leftmost nonzero entry lies in column $i+1$ or later, and p_i is the number of rows for which the rightmost nonzero entry lies in column i or earlier. It follows that p_i and f_i have the following properties,

$$0 = p_0 \leq p_1 \leq p_2 \cdots \leq p_n = k, \tag{1.13}$$
$$k = f_0 \geq f_1 \geq f_2 \cdots \geq f_n = 0, \tag{1.14}$$
$$p_i \leq p_{i-1} + 1, \quad i = 1, \cdots\cdots, n. \tag{1.15}$$
$$f_{i-1} \leq f_i + 1, \quad i = 1, \cdots\cdots, n. \tag{1.16}$$

Theorem 1.2 (Forney [27]).

$$s_i = k - f_i - p_i, \tag{1.17}$$

$$e_i = k - f_{i+1} - p_i. \tag{1.18}$$

Example 1.2 *The [6,3,3] shortened Hamming code*

Consider the [6,3,3] code again. The following table demonstrates the connection between the MSGM (shown in Example 1.1) and trellis complexities.

Position	0	1	2	3	4	5	6
Future code f_i	3	2	1	0	0	0	0
Past code p_i	0	0	0	1	1	2	3
Vertex dimension s_i	0	1	2	2	2	1	0
Edge dimension e_i	-	1	2	3	2	2	1
# of active elements	-	1	2	3	2	2	1

1.4 Trellis complexity bounds

1.4.1 The DLP bounds

Motivated by the idea of *Generalized Hamming Weight* (GHW) [84], Forney defined the *dimension/length profile* (DLP) for block codes [28].

Definition: Let $K_i(C)$ be the maximum dimension of any subcode of an (n, k, d) linear block code C whose support size is no greater than i, $0 \leq i \leq n$. The sequence $K_0(C), K_1(C), \cdots, K_n(C)$ is the dimension/length profile.

Lemma 1.1

$$p_i \leq K_i(C), \tag{1.19}$$

$$f_i \leq K_{n-i}(C). \tag{1.20}$$

Combining Theorem 1.2 and Lemma 1.1, trellis complexities are lower bounded by the DLP [28, 37, 40, 43, 84].

Theorem 1.3

$$
\begin{aligned}
s_i &\geq k - K_i(C) - K_{n-i}(C), & (1.21)\\
e_i &\geq k - K_{n-i-1}(C) - K_i(C), & (1.22)\\
|E| &\geq \sum_{i=1}^{n} 2^{k-K_{n-i-1}(C)-K_i(C)}, & (1.23)\\
|V| &\geq \sum_{i=0}^{n} 2^{k-K_{n-i}(C)-K_i(C)}. & (1.24)
\end{aligned}
$$

The DLP bounds are not easy to apply, since the DLP is hard to compute for most codes. One notable exception is that the DLP of the entire family of Reed-Muller codes are completely determined in [37, 38]. It turns out that the DLP bounds hold with equality for every RM code.

1.4.2 Bounds based on tables of known best linear codes

Let $K(n,d)$ be the largest possible dimension of a linear block code of length n and minimum hamming distance d. Then

$$K_i(C) \leq K(i,d). \qquad (1.25)$$

Theorem 1.4 [22, 40]

$$
\begin{aligned}
s_i &\geq k - K(i,d) - K(n-i,d), & (1.26)\\
e_i &\geq k - K(n-i-1,d) - K(i,d), & (1.27)\\
|E| &\geq \sum_{i=1}^{n} 2^{k-K(n-i-1,d)-K(i,d)}, & (1.28)\\
|V| &\geq \sum_{i=0}^{n} 2^{k-K(n-i,d)-K(i,d)}. & (1.29)
\end{aligned}
$$

These bounds are, in general, looser than the DLP bounds, but are, on the other hand, easier to compute since tables of $K(n,d)$ are available [14].

1.4.3 Bounds based on the code's span

Since the sum of edge dimensions equals the total span of the MSGM (see lemma 4.1), an analysis of the spatial distribution of the total span results in new lower bounds on trellis complexities. We present these bounds in section 2.2.

1.4.4 Permutation-invariant bounds

For a fixed coordinate ordering, the minimal trellis structure is unique. However, different permutations of the symbols yield different minimal trellises. It is not known yet if for any code there exists one permutation which simultaneously minimizes all trellis complexities.

Definition: A bound on trellis complexity is permutation-invariant if it holds for any symbol permutation.

The DLP bounds and the bounds based on the code tables are permutation-invariant. The bounds based on the code's span, in contrast, are not permutation-invariant.

1.5 Summary

In this chapter, we defined the minimal trellis structure and associated trellis complexity for block codes. The construction of a minimal trellis based on the minimal span generator matrix was presented in [58]. If the symbol ordering is fixed, the minimal trellis structure is unique and the trellis complexities can be read directly from the MSGM. Lower bounds on trellis complexities were also presented.

The rest of the thesis is organized as follows. In Chapter 2, we investigate the complexity of minimal trellises for block codes over all coordinate permutations. Our approach is based on the concept of the *span* of a generator matrix, which connects the code parameters and the trellis complexity. New bounds on trellis complexities such as E (the total number of edges) and V (the total number of vertices) are obtained from the analysis of the *span distribution.* Aiming to minimize the total span of a generator matrix, an efficient, “divide–and–conquer” algorithm and variants are proposed to search for the optimal trellis structure (corresponding to an optimal coordinate permutation) for any block code.

In Chapter 3, by introducing the concept of *trellis–canonical* generator matrices and a simple algorithm to compute one, we develop a general theory of the minimal trellises for convolutional codes. In this theory, *punctured convolutional codes* no longer have to be treated as special cases. By then, the minimal trellises for block and convolutional codes are both well–defined. This allows one to make a direct performance–complexity comparison between block and convolutional codes.

The tradeoff between performance (measured by the asymptotic coding gain–ACG) and complexity (measured by the logarithm trellis edge complexity–LTC) is the theme of Chapter 4. Via the span analysis, we prove a universal bound on the complexity to performance ratio. It implies that $\frac{LTC}{ACG}$ can never be smaller than 1 for any code, block or convolutional. In some cases, the bound is optimal or asymptotically optimal. The study suggests that optimal codes in terms of minimum distance or free distance do not necessarily offer the best tradeoff.

Chapter 5 deals with the implementation of maximum–likelihood decoding and the computational complexity for convolutional and block codes. By combining the optimal sectionalization technique [45] and the minimal trellis theory, a low complexity hybrid decoding algorithm is developed. For some partial unit memory convolutional codes, the resulting decoding complexity is significantly superior to other known algorithms. There are two parts to the computational complexity. One is the edge metric computation cost θ. We prove a lower bound on θ which is independent of the computation mechanism (sequential or parallel), and which is optimal in some cases. The other is the cost of the state metric updating which is inferable from the trellis structure. This establishes a lower bound on the computational complexity for any implementation.

In the last chapter, we propose some directions for future research.

Chapter 2 Permutation Trellis Complexity of Block Codes

2.1 The problem

If a block code C is being used on a memoryless channel, permuting the columns of its generator matrix will not affect the code's performance. But as Bahl *et al.* [7] were the first to note, a coordinate permutation of C may drastically alter its trellis complexity. In this chapter, we measure the trellis complexity by the total number of edges $E(C)$ in a minimal trellis. However, the discussion extends to other complexity measures.

Given an $[n, k, d]$ linear code C with generator matrix G, let the coordinates of codewords be labeled, from left to right, as $1, 2, 3, \ldots, n$. Let S_n be the permutation group on {1,2,3,. . . n}. For any $\sigma \in S_n$, σC denotes the code with generator matrix σG obtained by permuting the columns of G according to σ. Γ_C is defined as the set of codes equivalent to C, viz. $\Gamma_C = \{\sigma C : \sigma \in S_n\}$. The permutation edge complexity (PEC) of C is defined as

$$E_C = \min_{C^* \in \Gamma_C} \{E(C^*)\}. \tag{2.1}$$

A similar definition applies to other trellis complexity measure.

In this chapter, we consider the problem of finding E_C and a permutation achieving it, **for a fixed code**. In section 2.2, we present a lower bound on the permutation edge complexity via an analysis of the span distribution of a MSGM. It strongly suggests that minimizing the total span will minimize E_C. A heuristic algorithm for finding "good" coordinate permutations for a given code is presented in section 2.3.

2.2 The span distribution and PEC

In Lemma 4.2, we shall prove a relation between the total span and the edge dimension distribution, i.e., the sum of the edge dimensions equals to the total span. It is well known [57, 58] that for a fixed column ordering, the edge complexity is minimized by minimizing the total span in a generator matrix. But it is not clear if the permutation edge complexity can be minimized by minimizing the permutation total span. However, the total span is easier to handle. Many examples show that the permutation total span and the permutation edge complexity can be minimized simultaneously. We believe that this is true for most codes, if not all. In this section, we shall further study the properties of the edge dimension distribution and its connection with the total span and edge complexity.

2.2.1 The A-sequence

Let A_C^i be the number of active elements in the MSGM of a linear code C at column position i. $A_C^1, A_C^2, \cdots, A_C^n$ is a sequence[1] with many interesting properties.

Lemma 2.1 *The above defined A-sequence has the following properties:*

1.

$$\sum_{i=1}^{n} A_C^i = m_G \triangleq \text{the total span}. \tag{2.2}$$

2.

$$|A_C^{i+1} - A_C^i| \leq 1, \quad \textit{where} \;\; 1 \leq i \leq n-1. \tag{2.3}$$

3.

$$1 \leq A_C^i \leq \min(i, n-i+1, k). \tag{2.4}$$

4.

$$E(C) = \sum_{i=1}^{n} 2^{A_C^i}. \tag{2.5}$$

[1]It is the edge dimension profile. But we denote it by A_C^i instead of e_i in order to introduce A-sequences for the code.

5. *For any $i > 1$, compare all pairs A_C^i and A_C^{i-1}. Suppose there are a greater, b equal and c less, then*

$$a + b = b + c = n - 1 - a \geq k, \quad \textit{and} \quad a, c \leq k. \tag{2.6}$$

Proof:

1. See the proof of Lemma 4.2.

2. A row is active at one column position if its entry is active. In a MSGM, at most one row becomes active or inactive at each column position. At column position i, if one row becomes active, A_C^i is increased by 1 from A_C^{i-1}. If one row becomes inactive , A_C^i is decreased by 1. If one row becomes active and another becomes inactive or neither happens, A_C^i stays. A_C^i is necessary positive since we assume no zero columns.

3. A_C^i reaches its maximum if and only if for any $1 < j \leq i$, A_C^j is increased by 1 from its preceding. That means $A_C^i \leq i$. By symmetry, $A_C^i \leq n + 1 - i$. In any case, A_C^i must be smaller than k.

4. This sequence is in fact the edge dimension distribution. See Chapter 1.

5. Each greater means one more active row; each equal may mean one more active row; each less means one less active row. Since there are k rows and $a+b+c = n-1$, all follow. This is implied by conditions 2 and 3. ■

Lemma 2.1 combined with the following lemma set tight conditions on the shape of edge dimension distributions.

Lemma 2.2 *If the code's minimal distance is d, then $kd \leq m_G \leq k(n + 1 - k)$.*

Proof: The lower bound is proved in Lemma 4.1. To prove the upper bound, consider the case when all A_C^i's reach their maxima (See part 3 of Lemma 2.1). ■

Example 2.1 *There are codes that attain the upper and lower bounds in Lemma 2.2.*

For any *Maximum Distance Separable* (MDS) Code, $d = n - k + 1$. The upper bound and the lower bound meet. The total span and edge complexity are constant over all coordinate permutations. Hence, MDS codes are the worst codes for trellis representation.

Definition: An A-sequence for an $[n, k, d]$ code of total span m is a sequence of integers satisfying conditions 1,2,3 in Lemma 2.1. $\mathcal{A}_{m,n,k}$ is the set of all A-sequences of given m, n, k.

2.2.2 The bounds on PEC

Given a MSGM whose span is m, for a fixed code, there is more than one possible A-sequence corresponding to m, n, k. Any true edge dimension distribution achieving total span m is an A-sequence, but the reverse is not true. However, we can check if the given MSGM corresponds to a good coordinate permutation.

Let

$$E(m; n, k)_{A_C} = \sum_{i=1}^{n} 2^{A_C^i}. \tag{2.7}$$

Then, define

$$\overline{E}(m; n, k) = \max_{x \in \mathcal{A}_{m,n,k}} E(m; n, k)_x, \tag{2.8}$$

$$\underline{E}(m; n, k) = \min_{x \in \mathcal{A}_{m,n,k}} E(m; n, k)_x. \tag{2.9}$$

Then the following hold:

$$\overline{E}(m; n, k) \geq \overline{E}(m-1; n, k) \geq \ldots \geq \overline{E}(n; n, k). \tag{2.10}$$

$$\underline{E}(m; n, k) \geq \underline{E}(m-1; n, k) \geq \ldots \geq \underline{E}(n; n, k). \tag{2.11}$$

Theorem 2.1 *(Lower Bound)*

Let

$$s = \left\lfloor \frac{n + 1 - \sqrt{(n+1)^2 - 4m}}{2} \right\rfloor, \tag{2.12}$$

$$t = m + s^2 - ns - s. \tag{2.13}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. Then

$$\underline{E}(m; n, k) = (n - 2s + t + 4)2^s - 4. \tag{2.14}$$

Proof: For the A-sequence attaining $\underline{E}(m; n, k)$, its elements must be as nearly equal as possible. For example, if $A_C^i = a + s$, $A_C^j = a$, and $s \geq 2$, $a \geq 1$, we may increase A_C^j by one and decrease A_C^i by one at the same time. The new sequence generates a smaller $E(m; n, k)$, since $2^{a+s-1} + 2^{a+1} < 2^{a+s} + 2^a$. However, the above procedure fails if A_C^j is saturated (i.e., reaches its maximum, see Lemma 2.1). Hence, for every k, A_C^k either attains its maximum or differs from all other unsaturated elements at most 1. Its shape is depicted in Fig. 2.1.

The total span in the unshaded area is

$$s(n - 2s + 2) + s(s - 1), \tag{2.15}$$

and the shaded area contributes t to the total span, where

$$0 \leq t = m - s(n - 2s + 2) - s(s - 1) < n - 2s, \tag{2.16}$$

$$0 \leq t = m + s^2 - ns - s < n - 2s. \tag{2.17}$$

Solving Eqn. (2.17), we obtain:

$$s = \left\lfloor \frac{n + 1 - \sqrt{(n+1)^2 - 4m}}{2} \right\rfloor .$$

Substituting this A-sequence into (2.7), (2.14) follows. ■

Theorem 2.2 *(Upper Bound, Case I)*

Suppose

$$u = \lfloor \sqrt{m - n} + 1 \rfloor \leq k. \tag{2.18}$$

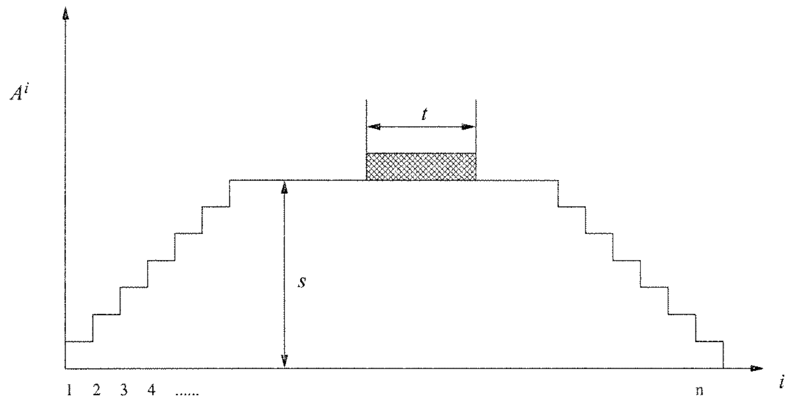

Figure 2.1: An A-sequence achieving $\underline{E}(m; n, k)$, where the total span is m.

Then

$$\overline{E}(m; n, k) = (3 + a)2^u + 2^{b+1} + 2(n - 2u - a) - 4, \tag{2.19}$$

where

$$m - n - (u-1)^2 = a(u-1) + b, \quad \textit{and} \quad \begin{cases} a = 0, & 0 \le b < u - 1. \\ a = 1, & 0 \le b < u - 1. \\ a = 2, & b = 0. \end{cases} \tag{2.20}$$

Proof: In contrast to the proof of Theorem 2.1, elements of this A-sequence must be as polarized as possible. The maximum of $A_i's$ must be as large as possible. Fig. 2.2 portraits its shape where the sequence is not necessarily presented in order. The total span in the unshaded area is:

$$n + (u-1)^2. \tag{2.21}$$

Since u is the maximal number such that the sequence has shape as Fig. 2.2, the shaded area has span

$$0 \le m - n - (u-1)^2 = a(u-1) + b < 2u - 1. \tag{2.22}$$

Solving (2.22), we get

$$u = \lfloor \sqrt{m-n} + 1 \rfloor. \tag{2.23}$$

Substituting this A-sequence into (2.7), the upper bound follows. ∎

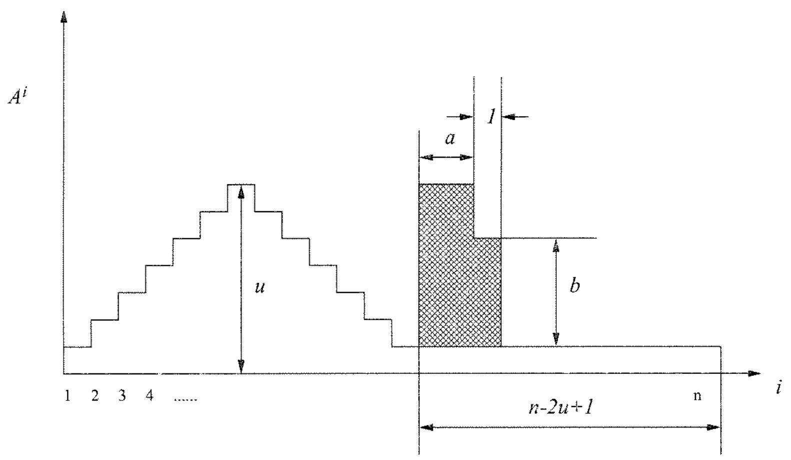

Figure 2.2: An A-sequence achieving $\overline{E}(m; n, k)$, where the total span is m and there is at least most one $A^i = k$. The sequence is not presented in order.

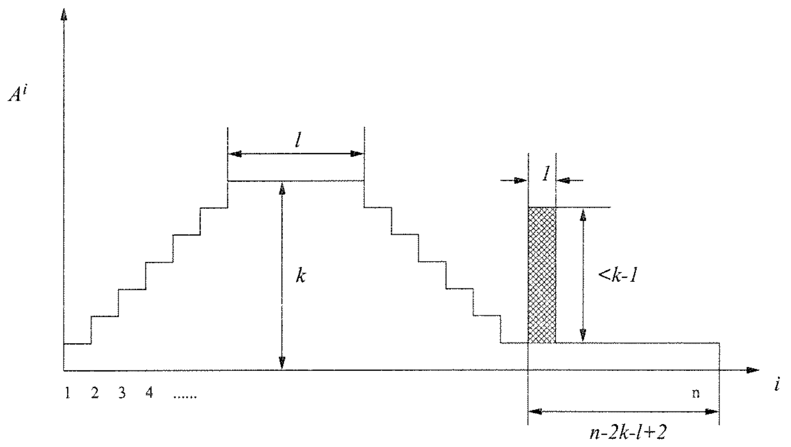

Figure 2.3: An A-sequence achieving $\overline{E}(m; n, k)$, where the total span is m and there is at least one $A^i = k$. The sequence is not presented in order.

Theorem 2.3 *(Upper Bound, Case II)*
If

$$\lfloor \sqrt{m-n}+1 \rfloor \geq k,$$

let

$$l = \left\lfloor \frac{m-n-2-k^2+3k}{k-1} \right\rfloor, \tag{2.24}$$

$$s = m-n-2-k^2+3k-lk+l. \tag{2.25}$$

Then

$$\overline{E}(m;n,k) = (l+2)2^k + 2^{s+1} + 2(n-2k-l) - 2. \tag{2.26}$$

Proof: Fig. 2.3 portraits the shape (the elements are not in order). The total span in the unshaded area is:

$$k^2 - 3k + kl + n - 2 - l. \tag{2.27}$$

The span in the shaded area satisfies

$$0 \leq m-n-2-k^2+3k-lk+l < k-1. \tag{2.28}$$

Solving these inequalities gives the conclusion. ■

Remarks: Theorems 2.1, 2.1, and 2.3 are similar to Theorems 4 and 6 in [40]. These are independent work of Lin and Kiely *et al.*, respectively. While the theorems in [40] are applicable to q-ary codes, the results here give exact value for binary case. Theorem 2.4 gives tighter bounds. The idea of presenting the A-sequence graphically, which makes the proofs shorter and clearer, is from Kiely and Dolinar.

Example 2.2 *The [8,4,4] extended Hamming code*

For this code, one MSGM is

$$\begin{pmatrix} 11110000 \\ 00111100 \\ 00001111 \\ 01100110 \end{pmatrix}$$

with $m = 18$. The corresponding A-sequence is $(1,2,3,3,3,3,2,1)$. $\underline{E}(18;8,4) = 44$, which is equal to the permutation edge complexity. $\overline{E}(18;8,4) = 48$.

Example 2.3 *The [24,12,8] Golay code*

Consider the Golay code with generator matrix:

$$\begin{pmatrix}
1111 & 1111 & 0000 & 0000 & 0000 & 0000 \\
0000 & 1111 & 1111 & 0000 & 0000 & 0000 \\
0000 & 0000 & 1111 & 1111 & 0000 & 0000 \\
0000 & 0000 & 0000 & 1111 & 1111 & 0000 \\
0000 & 0000 & 0000 & 0000 & 1111 & 1111 \\
0011 & 0011 & 1100 & 1100 & 0000 & 0000 \\
0000 & 0000 & 0011 & 0011 & 1100 & 1100 \\
0110 & 0110 & 0110 & 0110 & 0000 & 0000 \\
0000 & 0000 & 0110 & 0110 & 0110 & 0110 \\
0001 & 0001 & 0001 & 1110 & 1000 & 1000 \\
0000 & 0101 & 0011 & 1001 & 1010 & 0000 \\
0000 & 0011 & 0110 & 1010 & 1100 & 0000
\end{pmatrix}$$

The MSGM of the code shown above [27] has total span $m = 136$ and achieves permutation edge complexity 3580. $\underline{E}(136;24,12) = 3068$, which is 512 less. Since $u = 11 < k = 12$, $a = 1$, $b = 2$, Theorem 2.2 tells $\overline{E}(136;24,12) = 8198$, which is much greater than the actual E_C. This shows that the actual edge dimension distribution tends to approach the lower bound.

We can refine the bounds on $E(m;n,k)$ if $A_{max} = \max\{A_C^i : 1 \le i \le n\}$ is known. The set of A-sequence maybe reduced. The proof of the following theorem is similar to that of Theorem 2.1, 2.2, and 2.3.

Theorem 2.4

$$\overline{E}(m, A_{max}; n, k) = 3 \cdot 2^{A_{max}} - 4 + p \cdot 2^{A_{max}} + 2^{q+1} + 2 \cdot (n - p - 2A_{max}) \quad (2.29)$$

where $m - n - (A_{max} - 1)^2 = p \cdot (A_{max} - 1) + q, \quad p \geq 0, \quad 0 \leq q \leq A_{max} - 2.$

$$\underline{E}(m, A_{max}; n, k) = 3 \cdot 2^{A_{max}} - 4 + q \cdot 2^{p+1} + 2^p \cdot (n + 1 - 2A_{max} - q) \qquad (2.30)$$

where $m - A_{max}^2 = p \cdot (n + 1 - 2A_{max}) + q, \quad p \geq 0, \quad 0 \leq q \leq n - 2A_{max}.$

Theorem 2.5 *Suppose for $\sigma \in S_n$, m, the span of σG is minimal over all permutations. Then*

$$\underline{E}(m; n, k) \leq E_C \leq \overline{E}(m; n, k). \qquad (2.31)$$

If the actual edge complexity

$$E(\sigma C) \leq \underline{E}(m + 1; n, k), \qquad (2.32)$$

then there exists a permutation such that the total span and the permutation edge complexity are minimized. If

$$E(\sigma C) = \underline{E}(m; n, k), \qquad (2.33)$$

then σ is such a permutation.

Example 2.4 *The Hamming code in Example 2.3 has the minimal total span and the permutation edge complexity.*

If the generator matrix of a code C is equivalent, under row operations and column permutations, to the form[2]:

$$\begin{pmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{pmatrix} \qquad (2.34)$$

where $\mathbf{A}$, $\mathbf{B}$ are submatrices, $\mathbf{0}$ is zero matrix, then the span of C is the sum of spans of subcodes generated by $\mathbf{A}$ and $\mathbf{B}$. Its permutation edge complexity is the sum of that of the subcodes.

[2]This is called "Direct–Sum" code.

Finally, we show how the discussion here can be extended to other complexity measures. Let $s_G = \sum_{i=0}^{n} s_i$. Then

$$
\begin{aligned}
s_G &= \sum_{i=0}^{n}(k - p_i - f_i) = \sum_{i=0}^{n-1}(k - p_i - f_i) && (2.35)\\
&= \sum_{i=0}^{n-1}(k - p_i - f_{i+1}) - f_0 && (2.36)\\
&= m_G - k. && (2.37)
\end{aligned}
$$

Hence, v_G and m_G are always minimized simultaneously. We thus can develop bounds on the permutation vertex complexity [40].

2.3 The algorithms

In previous sections we have developed a number of lower bounds on the permutation-trellis complexity of a given block code. In this section, we will describe a fast heuristic algorithm we have developed which succeeds in finding a permutation with relatively low edge complexity. The objective function used by the algorithm is the total span of the code, rather than edge count.

2.3.1 The basic algorithm

This algorithm consists of three parts: the *preprocessor*, the *permuter*, and the *postprocessor.* The preprocessor consists of preliminary row operations, which put G into either *RRE* or *MS* form, and then sort the rows so that the weight of row 1 is a minimum, the weight of row 2 in those positions where row 1 is zero is a minimum, the weight of row 3, in those positions where rows 1 and 2 are both zero is a minimum, etc. A generator matrix in this form is denoted as G_1. If the columns of G_1 are viewed as k-bit binary expansions of integers in the range 0 to $2^k - 1$, then the "column permuter" arranges the columns in ascending order, with the resulting matrix denoted as G_2. Finally, the postprocessor converts the matrix G_2 into MS form, G_3, so that the total span can be computed.

Example 2.5 *(Simple Case)*

Consider the cyclic [7,4,3] Hamming code with generator polynomial $g(x) = x^3+x+1$, then the corresponding cyclic generator matrix

$$G_1 = \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix} \end{array} \tag{2.38}$$

which, having the LR property, is a MSGM (span=16), and is also already in "G_1" form. The decimal equivalents of the seven columns are as follows:

$$\begin{array}{rccccccc} \text{column}: & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \text{decimal equivalent}: & 8 & 4 & 10 & 13 & 6 & 3 & 1 \end{array}$$

Thus, sorting the columns into increasing order gives the matrix

$$G_2 = \begin{array}{c} \begin{array}{ccccccc} 7 & 6 & 2 & 5 & 1 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}. \tag{2.39}$$

Using the "minimal-span" algorithm algorithm given in [58], we obtain the following minimal-span form for G_2:

$$G_3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{2.40}$$

The span of G_3 is seen to be 15, one less than that of G_1, and indeed this is the minimal possible span among all column-equivalent versions of the original code C, and the corresponding trellis has 36 edges, which is also minimal.

2.3.2 Relaxation

The basic greedy algorithm is myopic. If it falls into local minimum, it cannot jump out and reach the global minimum.

Example 2.6 *The relaxation on the [5,3,2] code*

Consider the [5,3,2] code with RRE generator matrix:

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \tag{2.41}$$

If we sort the rows according to the basic algorithm, we obtain

$$G_1 = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \tag{2.42}$$

If we reorder the columns into ascending order, we obtain

$$G_2 = \begin{array}{c} \begin{array}{ccccc} 3 & 1 & 4 & 2 & 5 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}. \tag{2.43}$$

The MS form of G_2 is

$$G_3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix} \tag{2.44}$$

which has span 8, and edge count 16, which is not the smallest possible. To overcome this kind of problem, we can modify the algorithm to allow preliminary pairwise column exchanges. For example, if we interchange columns 1 and 4 of the original matrix G, and then apply the basic algorithm, we arrive at

$$G_3' = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

which has smaller span (7) and edge count (14) than the previous permutation, and indeed this permutation is globally minimal.

In this example, the exchange of one column representing an information bit and one column representing a parity check bit is crucial to obtain the global minimum. We call this *relaxation.* There are two kinds of relaxation, i.e., deterministic and stochastic. By deterministic, we mean the two columns to be exchanged are chosen according to some pre–defined rules. By stochastic relaxation, on the other hand, the columns are determined by random numbers generated according to some pre–defined distribution functions. In either cases, the choice could be rejected. For example, to expect the maximal impact on the code trellis structure, we would like to reject parity check bit depending on only 1 or 2 information bits.

Example 2.7 *The [24,12,8] Golay code*

The search algorithm found one optimal permutation for [24,12,8] Golay code in short time. See Example 2.4.

Example 2.8 *The [48,24,12] extended quadratic residue code*

Table 2.1: One optimal coordinate permutation for the [48,24,12]QR code.

$$\begin{pmatrix}
000000000000000000000000000000000000111111111111\\
000000000000000000000000000000111111111111000000\\
000000000000000000000000000111000111111000111000\\
000000000000000000000000011011001001100100110110\\
000000000000000000000000110010011010101001101100\\
000000000000000000000011101110101010110000100000\\
000000000000000000011001100101010001111100000000\\
000000000000000000101011000111001000011001010000\\
000000000000000011000010101100010010011110000000\\
000000000000000011001000001011010111010100000000\\
000000000000001010010110011010001001110000000000\\
000000000000011110111110000001100001000000000000\\
000000000000100001000001111110011110000000000000\\
000000000011010000010001011111100100000000000000\\
000000000110110000101011000011110000000000000000\\
000000001001110111011011001011101000000000000000\\
000000010111000101001110111000000000000000000000\\
000000101110100000101100110101000000000000000000\\
000001100000101000101011101110000000000000000000\\
000011011000110110101011000000000000000000000000\\
000111000111001110100110000000000000000000000000\\
001110110100011100111000000000000000000000000000\\
010101110001111111000000000000000000000000000000\\
111111111111000000000000000000000000000000000000
\end{pmatrix}$$

The [48,24,12] self-dual code is the extended [47,24,11] QR code. One generator polynomial of the [47,24,11] QR code is $1 + x + x^2 + x^3 + x^5 + x^6 + x^7 + x^9 + x^{10} + x^{12} + x^{13} + x^{14} + x^{18} + x^{19} + x^{23}$. The modified algorithm found the following optimal equivalent code in 12 minutes on a Sun Sparc 10 workstation. In 1996, Berger and Be'ery gave another optimal permutation based on the "twisted squaring construction" [9].

2.3.3 One optimal permutation for [48,24,12] QR code

If the dual distance $d^{\perp}$ is known, the bounds in Theorem 1.4 can be refined. Since the dual code of $[n, k, d, d^{\perp}]$ code is an $[n, n-k, d^{\perp}, d]$ code, and since Forney (Theorem 3 in [28] and in many others) has shown that the DLP for the dual code $C^{\perp}$ is related to the DLP for the primal code C by

$$K_i(C) = i - (n-k) + K_{n-i}(C^{\perp}), \quad for\ i = 0, 1, \ldots, n, \tag{2.45}$$

the following theorem follows:

Theorem 2.6

$$p_i \leq \begin{cases} K(i,d) \\ i - (n-k) + K(n-i, d^{\perp}) \end{cases} \tag{2.46}$$

$$f_i \leq \begin{cases} K(n-i,d) \\ k - i + K(i, d^{\perp}) \end{cases} \tag{2.47}$$

■

If we define

$$K_i^*(n,k,d,d^{\perp}) = \min(K(i,d), i - (n-k) + K(n-i, d^{\perp})), \tag{2.48}$$

we also have the lower bound

$$E_C \geq \sum_{i=0}^{n-1} 2^{k - K_i^*(n,k,d,d^{\perp}) - K_{n-i-1}^*(n,k,d,d^{\perp})}. \tag{2.49}$$

Consider the [48,24,12] extended quadratic residue code. Since this code is self-dual, we have $d^{\perp} = 12 = d$ and

$$p_i \leq \begin{cases} K(i,12) & 0 \leq i \leq 24 \\ i - 24 + K(48-i, 12) & 24 \leq i \leq 48 \end{cases} \tag{2.50}$$

$$f_i \leq \begin{cases} K(48-i,12) & 24 \leq i \leq 48 \\ 24 - i + K(i, 12) & 0 \leq i \leq 24 \end{cases}. \tag{2.51}$$

If we look up the tables of $K(i,d)$ in [14], the above inequalities give the following DLP of the code:

i:	0	1	2	3	4	5	6	7	8	9	10	11	12
p$_i$:	0	0	0	0	0	0	0	0	0	0	0	0	1
f$_i$:	24	23	22	21	20	19	18	17	16	15	14	13	13

i:	13	14	15	16	17	18	19	20	21	22	23	24
p$_i$:	1	1	1	1	1	2	2	2	3	3	4	5
f$_i$:	12	11	10	9	8	8	7	6	6	5	5	5

and the values of p_i, f_i with $25 \leq i \leq 48$ are given by the formula $p_i = f_{48-i}$ and $f_i = p_{48-i}$. Substituting the DLP bounds into (2.49), we find that $E_C \geq 860,156$. The code we found achieves the DLP bounds, and so $E_C = 860,156$ for the [48,24,12] extended QR code [22, 9].

2.3.4 Extending and puncturing

The algorithm described in the previous section seems to work quite well for codes of even length, but for odd length codes, its performance is sometimes disappointing. We have discovered a useful trick that overcomes this problem in many cases. If we have an odd-length $[n,k]$ code C, we apply the algorithm to the extended, even length, [n+1, k] code C', obtaining a good permutation for C'. We then delete the overall parity-column from the permuted, extended code.

If the extended code is invariant under a transitive permutation group, then any column of the extended code can be deleted ([54], Corollary 8.15), so that by deleting the column which reduces the span as much as possible, a good permutation for the original code usually results.

Example 2.9 *Consider the [15,5,7] code BCH with generator polynomial* $1+x+x^2+x^4+^5+x^8+x^{10}$.

The optimal generator matrix of the extended code is:

$$\begin{pmatrix} 0000000011111111 \\ 0000111111110000 \\ 0011001111001100 \\ 0101010110101010 \\ 1111111100000000 \end{pmatrix}$$

Deleting column 8 results in one good equivalent code.

2.4 Summary

In this chapter we studied the bounds on PEC given the total span of the generator matrix. We also proposed efficient heuristic algorithms to search for optimal or good permutations for any block codes.

In appendix A, we list computer simulation results of these algorithms on various codes.

List of algorithms

The basic algorithm:

Preprocessor:

- Transform the original matrix into Row-Reduced Echelon form (sometime MS form is enough). Then each column is either an information column or a parity check column.
- Sort the rows: Let one row of least weight as the first row. Suppose the first m rows have been chosen, row $m+1$ is chosen to be one with least weight at positions where previous chosen rows all are 0.

Permuter:

- Permuting the columns such that all 1's in the first row are in one side. For example, (0,0,1,0,1,1,0,1,1,0,0) is permuted to (0,0,0,0,0,0,1,1,1,1,1). At this point, the input matrix can be represented as:

$$\begin{pmatrix} \mathbf{0} & \mathbf{1} \\ G_a & G_b \end{pmatrix}$$

- Record the permutation if necessary. Then send G_a and G_b to lower level permuters to process next row. This can be easily realized by divide–and–conquer programming.

Postprocessor: Apply the "minimal-span" algorithm given in [58].

The modified algorithm:

Step 1: Setup termination conditions (i.e., the maximal number of iterations).

Step 2: Apply the basic greedy algorithm.

Step 3: Termination conditions are met? If yes, go to step 6. If no, go to step 4.

Step 4: Choose one information column and one parity check column.

Step 5: If choice is accepted, exchange them then go to step 2. Otherwise go to step 4.

Step 6: End.

The algorithm for odd length codes:

Step 1: Add a parity check column to the original code.

Step 2: Apply the modified searching algorithm to the extended code.

Step 3: Delete the added column. Check LR conditions if necessary.

The algorithm for codes with transitive permutation group:

Step 1: Add a parity check column to the original code.

Step 2: Apply the modified searching algorithm to the extended code.

Step 3: Delete the column which reduces the span maximally. Check LR conditions if necessary.

Chapter 3 The Trellis Complexity of Convolutional Codes

3.1 Introduction

Since its introduction for convolutional codes, the trellis has proven to be a powerful tool for understanding and analyzing the dynamics of convolutional codes and their decoding algorithms. This success spurred extensive research on minimal trellis representation for block codes. For a long time, ironically, it seemed that people understood the minimal trellis structures of block codes very well but knew relatively little about the minimal trellis structures of convolutional codes. In the recent work of [61] and [73, 72], theories of minimal trellis representation for convolutional codes were first presented. These advancements were motivated by the research on block codes partially presented in previous chapters.

In this chapter, we shall answer the following questions concerning a convolutional code:

- What is the minimal trellis structure?
- What is the trellis-canonical PGM?
- How can we obtain the minimal trellis structure and the trellis-canonical PGM?
- What properties should the trellis-canonical PGM have?

The rest of this chapter is organized as follows. In section 3.2, we briefly review the fundamentals of algebraic theory of convolutional codes which is to be used throughout the rest of this chapter. A complete treatment of this subject can be found in [59]. In section 3.3, we elaborate the idea of minimal trellis structure of convolutional codes and define the trellis-canonical PGM. Section 3.4 presents a general algorithm for finding the minimal trellis structure, given any PGM as input. While the algorithm always terminates, it does not guarantee the output to be trellis-canonical. However, as proved in section 3.5, for the class of most useful PGMs, viz, the canonical PGMs, there exists a simpler algorithm which always terminates and guarantees a trellis-canonical output. In section 3.6, we show that permutations can reduce the trellis complexity of a convolutional code substantially. The last section is the conclusion.

3.2 The basic theory of convolutional codes

An (n, k) binary convolutional code C is a k-dimensional subspace of $F(D)^n$, where $F(D)$ is the field of rational functions in D over $GF(2)$. A generator matrix $G(D)$ for C is a $k \times n$ matrix over $F(D)$ whose rows form a basis for C. It follows that any convolutional code has a polynomial generator matrix (PGM) whose entries are all polynomials. Hereafter, we only consider PGMs.

Let a PGM for C be

$$G(D) = G_0 + G_1 D + G_2 D^2 + \cdots + G_L D^L = (G_{i,j}(D)), \tag{3.1}$$

where G_i is a $k \times n$ binary scalar matrix, and $G_{i,j}(D)$ is a polynomial. Without loss of generality, we assume G_0 and G_L be non–zero. Denote the degree of a polynomial $P(D) = \sum_{i=0}^{L} a_i D^i$ be L if $a_L \neq 0$ or $P(D) = 0$. Then, for each row i of $G(D)$, define its degree $deg_i = \max_j\{\text{degree of } G_{i,j}(D)\}$. The degree of a polynomial matrix is the maximum degree of all its entries. The internal degree intdeg $G(D)$ of $G(D)$ is the maximum degree of all its $k \times k$ minors. The external degree extdeg $G(D)$ of $G(D)$ is the sum of its row degrees.

A $k \times n$ polynomial matrix $G(D)$ is called *basic* if, among all polynomial matrices of the form $T(D)G(D)$, where $T(D)$ is a nonsingular $k \times k$ matrix over $F(D)$, it has the minimum possible internal degree, i.e.,

$$\text{intdeg } G(D) \leq \text{intdeg } T(D)G(D).$$

A $k \times n$ polynomial matrix $G(D)$ is called *reduced* if its external degree cannot be reduced by a sequence of elementary row operations, i.e.,

$$\text{extdeg } G(D) \leq \text{extdeg } T(D)G(D).$$

where $T(D)$ is a square polynomial matrix whose determinant is a nonzero scalar. A PGM $G(D)$ for the convolutional code C is *canonical* if and only if it is both reduced and basic. It turns out that every PGM is equivalent to a canonical PGM.

Suppose $G(D)$ is canonical. Then, deg_i is called the *Forney index* of row i. Let the Forney indices be denoted by $f_1 \leq f_2 \leq \cdots \leq f_k$. The largest Forney index $f_k = L$ is the *memory* of the code and the sum $m = f_1 + f_2 + f_3 + \cdots + f_k$ is the *degree* of the code.

3.3 Minimal trellises for convolutional codes

The dynamics of a convolutional code can be depicted by a finite state machine. If we assume the starting point be the zero state and make time expansion of the finite state diagram, we construct a conventional trellis for this code. This trellis in principle is infinite but has a very regular structure consisting of a short initial transient section called the *head* and repeated copies of what we shall call the *trellis module.* In a truncated trellis there is also a section called the *tail.* In each trellis module, there are 2^m initial states and 2^m final states. In general, the conventional trellises for convolutional codes have a regular structure as shown in Figure 3.1. Thus we may concentrate on one trellis module instead of the whole trellis to analyze such trellis complexity measurements as the number of edges, the number of vertices and the spatial distribution of these edges and vertices, all with respect to each information bit.

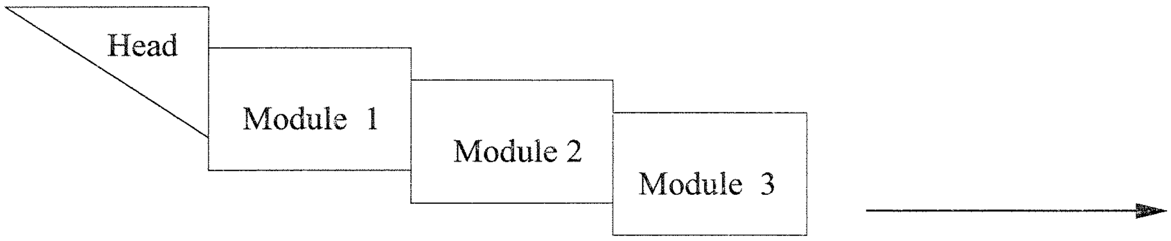

Figure 3.1: General Trellis Structure for Convolutional Codes.

A convolutional code has many trellis modules other than the conventional one. The minimal trellis modules are those which are simpler than others, in terms of some trellis complexity measurements. Based on minimal trellis modules, the minimal trellis structure is built. To demonstrate this idea, consider the discrete semi-infinite generator matrix G^* for a convolutional code.

$$G^* = \begin{pmatrix} G_0 & G_1 & G_2 & \cdots & G_L & & & & \\ & G_0 & G_1 & G_2 & \cdots & G_L & & & \\ & & G_0 & G_1 & G_2 & \cdots & G_L & & \\ & & & G_0 & G_1 & G_2 & \cdots & G_L & \\ & & & & \ddots & \ddots & \ddots & \ddots & \ddots \end{pmatrix} \tag{3.2}$$

A convolutional code is thus like a block code with infinite length. The techniques developed in [58, 42] can be used to construct the minimal trellis. As we shall see, however, the situation is more complex than for block codes.

If we concatenate the $L+1$ matrices $G_0, G_1, G_2, \cdots, G_L$, we obtain a $k \times (L+1)n$ scalar matrix, which we denote by $\tilde{G}$:

$$\tilde{G} = (G_0 \; G_1 \; G_2 \; \cdots \; G_L) \tag{3.3}$$

Then

$$G^* = \begin{pmatrix} \tilde{G} \\ S(\tilde{G}) \\ S^2(\tilde{G}) \\ \vdots \end{pmatrix} \qquad (3.4)$$

where $S(\cdot)$ is the shift-right-one-block operator. We call each shifted version of $\tilde{G}$ a *layer* of G^*. For example, $S^3(\tilde{G})$ represents the third layer in G^*.

Each trellis module corresponds to one complete vertical section of G^*, i.e.,

$$\hat{G} = \begin{pmatrix} G_L \\ G_{L-1} \\ \vdots \\ G_1 \\ G_0 \end{pmatrix}. \qquad (3.5)$$

According to the trellis theory for block codes [58], if there are e_i active elements in the i^{th} column of $\hat{G}$, the number of edges in the corresponding trellis module is

$$E = \sum_{i=1}^{n} 2^{e_i}.$$

If the number of active elements can be reduced in a trellis module, the edge complexity measurement, defined as the number of edges per information bit, can be reduced.

Unlike block codes, it is not enough to put the matrix $\tilde{G}$ into LR-form [58]. For two different rows i, j in $\tilde{G}$, it may happen that

$$R_i = R_j + n. \qquad (3.6)$$

In other words, with respect to G^*, row i in one layer of G^* has identical right end with row j in the next layer. Therefore, the spanlength of row i or j could be reduced further.

Definition: A PGM $G(D)$ is *LR-minimal* if for any two different rows of G^*, r_i and r_j, the following holds:

$$\text{(a)}\quad L_{r_i} \neq L_{r_j}. \tag{3.7}$$

$$\text{(b)}\quad R_{r_i} \neq R_{r_j}. \tag{3.8}$$

The following lemma says that it is not necessary to check Eqn. (3.7) and (3.8) for all rows in G^*.

Lemma 3.1 *A PGM G(D) is LR-minimal if and only if for any two different rows in $\tilde{G}$, i and j,*

$$\text{(a)}\quad L_i \neq L_j \mod n. \tag{3.9}$$

$$\text{(b)}\quad R_i \neq R_j \mod n. \tag{3.10}$$

Proof: Without loss of generality, suppose $G(D)$ to be LR-minimal, and $L_i - L_j = s \cdot n$ for some i, j and $s \geq 0$, then $s \neq 0$. But the left end of row j in the s^{th} layer of G^* is $L_j + s \cdot n$, which is equal to the left end of row i in the first layer of G^*. This contradicts the LR-minimal assumption. So Eqn. (3.7) and (3.8) imply Eqn. (3.9) and (3.10).

Similarly, if Eqn. (3.9) and (3.10) hold, then $G(D)$ must be LR-minimal. ■

The conditions (3.9) and (3.10) or (3.7) and (3.8) are therefore both called the *LR properties* of convolutional codes. In practice, conditions (3.9) and (3.10) are usually more convenient to use.

Definition: A PGM is *trellis-canonical* if it is canonical and, in one trellis module, minimizes

- the total number of edges.
- the total number of states.
- the total number of bifurcations.

Lemma 3.2 *A PGM is trellis-canonical if and only if it is canonical and LR-minimal.*

Proof: It follows that the LR-properties minimize the total number of edges per information bit, the total number of states per information bit, as well as the total number of bifurcations per information bit and some other complexity measurements simultaneously. See [58, 80]. ■

Thus, obtaining the minimal trellis structure is equivalent to obtaining the trellis-canonical PGM.

3.4 The basic algorithm

The question now arises as to how to reduce the trellis complexity for a given convolutional code. Analogous to what we did with block codes, the main objective is to reduce the number of active elements in a trellis module. If the LR properties do not hold on G^*, further reduction of trellis complexities is possible. Since the scalar generator matrices for convolutional codes are infinite, we can not simply apply row operations on G^*.

In general, there are five cases to be considered. For some classes of convolutional codes, the problem can be simplified. We don't consider the effect of column permutations until section 3.6.

In what follows, we assume $i < j$ and let the m^{th} row of the polynomial generator matrix $G(D)$ be denoted by $Row[m]$.

For any two rows r_i and r_j in G^*, let the corresponding rows in $G(D)$ be $Row[I]$, $Row[J]$, respectively.

CASE I:

$$GCD(Row[m]) \neq 1 \tag{3.11}$$

i.e., the greatest common divisor of the m^{th} row of $G(D)$ is a nonzero scalar. We divide $Row[m]$ by its GCD.

CASE II:

$$L_{r_i} = L_{r_j} \quad \text{and} \quad R_{r_i} \geq R_{r_j} \tag{3.12}$$

If r_i and r_j are in the same layer, it is enough to add $Row[J]$ to $Row[I]$. If r_i and r_j are in different layers, there exists some positive integer s such that D^s is a divisor of every entry in $Row[I]$. In other words, the rule of CASE I applies first.

CASE III:

$$R_{r_i} = R_{r_j} \quad \text{and} \quad L_{r_i} < L_{r_j} \tag{3.13}$$

If r_i and r_j are in the same layer, it is enough to add $Row[J]$ to $Row[I]$. If r_i and r_j are in different layers, there exists a positive integer s such that the row operation:

$$Row[I] = Row[I] + D^s \cdot Row[J] \tag{3.14}$$

will shorten the spanlength.

CASE IV:

$$L_{r_i} = L_{r_j} \quad \text{and} \quad R_{r_i} < R_{r_j} \tag{3.15}$$

CASE V:

$$R_{r_i} = R_{r_j} \quad \text{and} \quad L_{r_i} \geq L_{r_j} \tag{3.16}$$

In these two cases, if r_i and r_j are in the same layer, it is enough to add $Row[I]$ to $Row[J]$. If r_i and r_j are in different layers, there exists some positive integer s such that D^s is a divisor of every entry in $Row[I]$. Then return to CASE I.

Example 3.1 *The (3,2) code*

Consider the code with PGM:

$$G(D) = \begin{pmatrix} D^2 & 0 & D^2 \\ D & 1+D & 0 \end{pmatrix} \tag{3.17}$$

This code is LR-minimal. But it is not trellis-canonical. This example shows that we must consider CASE 1.

Now we come to an algorithm to find the trellis-canonical PGM with a general PGM as input. The algorithm calls a subroutine CHECK_GCD(Row_Vector) to check CASE I on the PGM.

The subroutine to check CASE I: CHECK_GCD(Row_Vector).

```
Begin:
IF $(GCD(Row\_Vector) \neq 1)$
    Row_Vector=Row_Vector/GCD(Row_Vector);
RETURN(Row_Vector).
End.
```

However, the algorithm actually examines the LR properties on $\tilde{G}$ instead of G^* and does all operations on $G(D)$. After any row operation on $G(D)$, subroutine CHECK_GCD(Row_Vector) is called again. The output PGM is guaranteed LR-minimal. In the algorithm, i, j are different integers and $0 \leq i,\ j \leq k-1$. s is a non-negative integer.

The algorithm for general PGMs: ALG1.

```
Begin:
FOR ($m = 0$ to $k - 1$) CHECK_GCD($Row[m]$);
WHILE (LR properties do not hold on $\tilde{G}$)
{
    IF (($R_i = R_j + s \cdot n$) and ($L_i < L_j + s \cdot n$) )
        {
        $Row[i] = Row[j]D^s + Row[i]$;
        CHECK_GCD(Row[i]);
        }
    IF ($L_i = L_j$ and $R_i \geq R_j$)
        {
        $Row[i] = Row[j] + Row[i]$;
        CHECK_GCD(Row[i]);
        }
    IF (($L_i = L_j$ and $R_i < R_j$ ) or ($R_i = R_j$ and $L_i \geq L_j$ ))
        {
        $Row[j] = Row[j] + Row[i]$;
        CHECK_GCD(Row[j]);
        }
}
End.
```

Theorem 3.1 *The algorithm ALG1 is convergent.*

Proof: Without loss of generality, assume $L_i = L_j + sn$. Since

$$GCD(Row[i]) = GCD(Row[j]) = 1$$

after calling the GCD checking subroutine, s must be 0. If $R_i \geq R_j$, adding row j to row i will reduce the total of active elements in $\tilde{G}$. This corresponds to adding $Row[n]$ to $Row[m]$. If $R_i < R_j$, adding row i to row j will also reduce the total number of active elements in $\tilde{G}$. This corresponds to add $Row[m]$ to $Row[n]$. Since the total number of active elements in $\tilde{G}$ is finite, it will terminate after a finite number of steps. ■

3.5 The simplified algorithm

Definition: A PGM is *strictly* row-reduced if the spanlength of the corresponding $\tilde{G}$ can not be reduced by an operation of the form

$$Row[m] = Row[n]D^s + Row[m], \tag{3.18}$$

where s is some integer and $m \neq n$.

In fact, this property is equivalent to the LR properties.

Theorem 3.2 *A PGM is strictly row-reduced if and only if it is LR-minimal.*

Proof:

Not LR-minimal$\Longrightarrow$ not strictly row-reduced.

If the LR properties fail, then for two different rows in $\tilde{G}$ and some $s \geq 0$, there are four cases:

1. $L_m = L_n + s \cdot n$ and $R_m \geq R_n + s \cdot n$

   $Row[m] = Row[n]D^s + Row[m]$ will reduce the spanlength of $\tilde{G}$.

2. $L_m = L_n + s \cdot n$ and $R_m < R_n + s \cdot n$

   $Row[n] = Row[m]D^{-s} + Row[n]$ will reduce the spanlength of $\tilde{G}$.

3. $R_m = R_n + s \cdot n$ and $L_m \geq L_n + s \cdot n$

   $Row[n] = Row[m]D^{-s} + Row[n]$ will reduce the spanlength of $\tilde{G}$.

4. $R_m = R_n + s \cdot n$ and $L_m < L_n + s \cdot n$

   $Row[m] = Row[n]D^s + Row[m]$ will reduce the spanlength of $\tilde{G}$.

Thus, the PGM must be not strictly row-reduced.

Not strictly row-reduced$\Longrightarrow$ *not LR-minimal.*

If an operation like Eqn. (3.18) can reduce the spanlength of $\tilde{G}$, either

$$L_m = L_n + s \cdot n$$

or

$$R_m = R_n + s \cdot n$$

will be true, since $L_{Row[n]D^s} = L_n + s \cdot n$ and $R_{Row[n]D^s} = R_n + s \cdot n$. Here s may be negative. Both cases imply that the PGM is not LR-minimal. ■

3.5.1 The basic encoder

Lemma 3.3 *A PGM G(D) is basic if and only if*

$$gcd\left(\Delta_i(D), i = 1, 2, \cdots, \begin{pmatrix} n \\ k \end{pmatrix}\right) = 1 \tag{3.19}$$

where $\Delta_i(D)$ are determinants of $k \times k$ minors of G(D).

Proof: For a proof of this lemma, see [59]. ■

For basic PGMs to be LR-minimal, it is not necessary for them to be strictly row-reduced.

Definition: A PGM is *row-reduced* if the spanlength of the corresponding $\tilde{G}$ can not be reduced by an operation of the form of Eqn. (3.18) where $s \geq 0$.

Theorem 3.3 *A basic PGM G(D) is LR-minimal if and only if it is row-reduced.*

Proof: Let the greatest common divisor of row i be $d_i(D)$, then $\Delta_k(D)$, the determinants of $k \times k$ minors of G(D), are all divisible by $d_i(D)$. Since G(D) is basic, $d_i(D) = 1$ from Lemma 3.3.

Suppose G(D) is not LR-minimal, then CASE II, III, IV, V might be true. In any of these four cases, operations like (3.18) can reduce the spanlength of $\tilde{G}$ and $s \geq 0$, as we saw in the previous section. Hence, G(D) is not row-reduced.

If G(D) is not row-reduced, then an operation like (3.18) that reduces the span of $\tilde{G}$ will necessarily reduce the span of G^* or $G(D)$ is not LR-minimal. ■

Lemma 3.4 *If a PGM is LR-minimal, then it is reduced.*

Proof: To prove this theorem, we need a result in [59].

Lemma 3.5 *Let the indicator matrix G_{id} of PGM G(D) be*

$$G_{id}(i,j) = \textit{coefficient of } D^{f_i} \textit{ in } G_{i,j}(D)$$

where f_i is the degree of row i of G(D). Then G(D) is reduced if and only if G_{id} has rank k.

Now we examine the indicator matrix G_{id} of a basic PGM which is LR-minimal. G_{id} has the same right end set as $\tilde{G}$. Since $G(D)$ is LR-minimal, $\tilde{G}$ has LR property. Therefore, G_{id} has R property, which means G_{id} has rank k. ■

Theorem 3.4 *A basic PGM is trellis-canonical if and only if it is row-reduced.*

Proof: If a basic PGM is trellis-canonical, it must be LR-minimal. From Theorem 3.3, it is also row-reduced.

If a basic PGM is row-reduced, then it is LR-minimal. Lemma 3.4 guarantees that it is reduced, too. Thus, the PGM is trellis-canonical. ■

3.5.2 The canonical encoder

Theorem 3.5 *A canonical PGM is trellis-canonical if and only if it is row-reduced.*

Proof: This follows from Theorem 3.4 and the fact that canonical PGMs are basic. ■

3.5.3 The simplified algorithm: ALG2

For the class of basic PGMs, a simplified algorithm readily follows from the theorems in the preceding section. All symbols have the same meanings as in ALG1.

The algorithm for basic PGMs: ALG2.

```
Begin:
WHILE (LR properties do not hold on $\tilde{G}$)
{
    IF (($R_i = R_j + s \cdot n$) and ($L_i < L_j + s \cdot n$) )
        $Row[i] = Row[j]D^s + Row[i]$;
    IF ($L_i = L_j$ and $R_i \geq R_j$)
        $Row[i] = Row[j] + Row[i]$;
    IF (($L_i = L_j$ and $R_i < R_j$ ) or ($R_i = R_j$ and $L_i \geq L_j$ ))
        $Row[j] = Row[j] + Row[i]$;
}
End.
```

Theorem 3.6 *The algorithm ALG2 terminates correctly.*

Proof: ALG2 is basically the same as ALG1, so it terminates. Since basic PGMs will still be basic after any number of elementary row operations, ALG2 needs not worry about GCD of any row in $G(D)$ at any time. When ALG2 terminates, the result PGM is basic and row-reduced, i.e., trellis-canonical. ■

Hence, in general, we would reduce the code to be basic then apply ALG2.

3.6 The Column Permutations

To illustrate the effect of column permutations on the trellis complexity of convolutional codes, we consider the (8,4,3) partial-unit-memory code with canonical generator matrix [1]:

$$G(D) = \begin{pmatrix} 11111111 \\ 11101000 \\ 10110100 \\ 10011010 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10101100 \\ 10010110 \end{pmatrix} D.$$

Applying ALG2 to this code, we obtain the canonical-minimal PGM for this code:

$$G(D) = \begin{pmatrix} 11111111 \\ 00010111 \\ 01001011 \\ 00101110 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10101100 \\ 00111010 \end{pmatrix} D.$$

The matrix for the corresponding trellis module turns out to be:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

The number of edge symbols in one trellis module is 480. Since it represents four bits, it follows that the trellis complexity is 120 symbol-edges[1] per information bit.

However, if column permutations are allowed, the permutation (01243567) results

[1]If one edge corresponds to 3 symbols, it counts 3 symbol-edges.

in an equivalent code whose PGM is:

$$G(D) = \begin{pmatrix} 11111111 \\ 11110000 \\ 10101100 \\ 10011010 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10110100 \\ 10001110 \end{pmatrix} D.$$

The trellis-canonical form of this code is:

$$G(D) = \begin{pmatrix} 11111111 \\ 00001111 \\ 01010011 \\ 00110110 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11011000 \\ 10110100 \\ 00111010 \end{pmatrix} D.$$

The trellis complexity turns out to be 104 symbol-edges per information bit.

3.7 Conclusion

In this chapter, we have shown that every convolutional code has a unique minimal trellis structure, which is usually much simpler than the conventional trellis structure. We also presented algorithms to construct the minimal trellis for any convolutional code. These results allow us to make a direct performance-complexity comparison for block and convolutional codes, which we do in the next chapter.

Chapter 4 LTC versus ACG

4.1 Introduction

In Chapter 1, we presented the minimal trellis theory for block codes. Algorithms were developed in Chapter 2 to search the minimal trellis structures for various codes. The results in Chapter 3, on the other hand, enable us to obtain the minimal trellis structures for convolutional codes. Since the minimal trellis structure determines the computational complexity of the Viterbi decoding algorithm or its variants, we are now in a good position to make a comparison of convolutional codes and block codes.

What is LTC?

In this chapter, the trellis complexity is restricted to mean the number of trellis edges per information bit. Since this number is typically large, we use *logarithmic trellis complexity* (LTC) which is the base–2 logarithm of trellis complexity. Let $|E|$[1] be the total number of edges in the minimal trellis for block codes or the minimal trellis module for convolutional codes, then

$$LTC = \log_2 \frac{|E|}{k}. \tag{4.1}$$

However, the comparison is not yet complete. For example, if there is no coding at all, the LTC would be minimal, but any decoding is meaningless. Thus, the *coding performance* is an indispensable ingredient in this discussion.

Usually E_b/N_0, the ratio of energy per information bit to noise spectral density to achieve some specified error probability, is the measure of a communication system. The amount of improvement of a coded system over uncoded systems is measured by the saving in E_b/N_0. Assuming a Gaussian channel and BPSK modulation, the

[1]Assume the optimal permutation. If the optimal permutation is unknown, we use the best known results.

asymptotic coding gain reflects the "merit" of a particular coding scheme or the amount of improvement as the decoded bit error probability goes to zero. On the other hand, for non-Gaussian channel and many popular modulation schemes such as Trellis Coded Modulation (TCM) and Multi-Level Coded Modulation (MLCM), the coding gain heavily depends on the signal constellation and the symbol–to–signal mapping besides the code itself. Under such a circumstance, there does not exist an omnibus quantity to exactly measure the coding gain. To preclude this dilemma, we employ ACG as the index of coding performance.

What is ACG?

The *asymptotic coding gain* (ACG) is defined as the product of the code rate and the minimal (block codes) or free (convolutional codes) distance.

$$ACG = \frac{k}{n}d = Rd \tag{4.2}$$

There are several advantages to using the ACG as the coding performance measurement.

- ACG is a simple, neat combination of the most important code parameters.
- ACG is independent of any particular modulation technique or channel conditions.
- ACG does have practical implication. It reflects the coding gain for Gaussian channel and BPSK modulation when the signal-to-noise ratio is large.
- It is broadly agreed that Gaussian channel and BPSK is the benchmark on testing binary codes.

What is the comparison?

In a fair comparison of codes, there are two criteria.

1. The ACG which indicates the coding performance. After all, we use codes to combat errors in communication. Thus the higher the ACG, the better the code.

2. The LTC–ACG ratio which measures the coding efficiency. This quantity helps one select the best code meeting the performance or complexity specifications. While there are many codes achieving the ACG requirement, they do not have the same coding complexity. A smaller LTC–ACG ratio means saving in decoding effort.

In this chapter, we shall examine the relationship between LTC and ACG for both block codes and convolutional codes. Since it is extremely difficult to compute the exact LTC-ACG ratio for most codes, we present a universal lower bound, which is similar to a result in [43]. In section 4.2, block codes are considered. Section 4.3 addresses convolutional codes. The comparison between block codes and convolutional codes is discussed in sections 4.3 and 4.4.

4.2 Block codes

4.2.1 The theorem

In this section we shall derive a lower bound on the LTC–ACG ratio by bounding the total number of edges in the minimal trellis in terms of n, k, and d. This can be done by studying the properties of a MSGM.

Lemma 4.1 *For any $[n, k, d]$ linear code, $m_G \geq kd$.*

Proof: Each row of the MSGM is itself a non–zero codeword. There are at least d non–zero entries in each such codeword. Hence $m_{r_i} \geq d$. Summing over all rows leads to the conclusion. ■

Lemma 4.2 *For any $[n, k, d]$ linear code, $\sum e_i = m_G$.*

Proof: An entry a_{ij} in the generator matrix is called *active* if $L_{r_i} \leq j \leq R_{r_i}$. We shall count the number of active entries in two ways. The number of active elements in row r_i is $|\{j : L_{r_i} \leq j \leq R_{r_i}\}|$, which is m_{r_i}. Thus the total number of active elements is the sum of all row spans, i.e., m_G.

On the other hand, the number of active elements in column j is

$$\begin{aligned} |\{i : L_{r_i} \leq j \leq R_{r_i}\}| &= k - |\{i : j+1 \leq L_{r_i}\}| - |\{i : j > R_{r_i}\}| && (4.3)\\ &= k - f_{j+1} - p_j && (4.4)\\ &= e_j && (4.5)\end{aligned}$$

Thus $\sum_i e_i = m_G$. ■

Theorem 4.1 *For any linear block code*

$$LTC \geq ACG - \log_2 R. \tag{4.6}$$

Proof: By the arithmetic–geometric inequality,

$$\begin{aligned} |E| &= \sum_i 2^{e_i} && (4.7)\\ &\geq n(2^{\sum_i e_i})^{\frac{1}{n}} && (4.8)\\ &\geq n2^{\frac{m_G}{n}} \text{ (By Lemma 2).} && (4.9)\end{aligned}$$

Hence

$$\begin{aligned} LTC &= \log_2 \frac{|E|}{k} && (4.10)\\ &\geq \log_2 \frac{n}{k}2^{\frac{m_G}{n}} && (4.11)\\ &\geq \log_2 \frac{n}{k}2^{\frac{kd}{n}} \text{ (By Lemma 1)} && (4.12)\\ &= \log_2 \frac{n}{k} + \frac{kd}{n} && (4.13)\\ &= ACG - \log_2 R. && (4.14)\end{aligned}$$

■

Corollary 4.1 *$LTC \geq ACG$, with equality if and only if $R = 1$.*

Proof: Since $R \leq 1$, $-\log_2 R \geq 0$. Then $LTC \geq ACG$.

If $LTC = ACG$, R must be 1.

For the $[n, n, 1]$ code, $ACG = 1$. Since $e_i = 1$, $LTC = \log_2 \frac{\sum_{i=1}^{k} 2}{k} = 1 = ACG$. ■

4.2.2 Some examples

Example 4.1 *The* $[n, n-1, 2]$ *parity-check code.*

$$\begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

The MSGM for the $[n, n-1, 2]$ code.

Its minimal trellis has $4(n-1)$ edges. Thus $LTC = 2$.

The ACG is

$$2\frac{n-1}{n}.$$

Hence

$$\frac{LTC}{ACG} = \frac{n}{n-1}.$$

Note that this ratio is always strictly greater than 1 but it approaches 1 as the code length goes to infinity. Hence (4.6) is asymptotically tight in this case.

Example 4.2 *The* $[n, 1, n]$ *repetition code.*

$$\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \end{pmatrix}$$

The MSGM for the $[n, 1, n]$ code.

The ACG is 1 and

$$LTC = 1 + \log_2 n.$$

Therefore, the LTC–ACG ratio is

$$\frac{LTC}{ACG} = 1 + \log_2 n.$$

Since $R = 1/n$, the bound (4.6) is exact in this case.

Example 4.3 *First order Reed–Muller codes.*

In [37, 38], the optimal trellises for Reed-Muller codes were completely determined. For the first order RM codes, $n = 2^m, k = m+1, d = 2^{m-1}$, and $ACG = \frac{m+1}{2}$. The edge dimension distribution is:

dimension :	1	2	3	$\cdots$	$m-1$	m
number of :	2	2	2^2	$\cdots$	2^{m-2}	2^{m-1}

(This means that there are two stages with edge dimension one and so on)
Then

$$\begin{aligned} E &= 2 + \sum_{i=1}^{m} 2^i 2^{i-1} & (4.15) \\ &= 2 + \frac{4^{m+1} - 4}{6}. & (4.16) \end{aligned}$$

Therefore

$$\frac{LTC}{ACG} = \frac{2(\log_2(4^{m+1} + 8) - \log_2 6(m+1))}{m+1} \quad (4.17)$$

and

$$\lim_{n \to \infty} \frac{LTC}{ACG} = 4. \quad (4.18)$$

The left end (L_i) set is $\{0, 2^0, 2^1, \cdots, 2^{m-1}\}$. The right end (R_i) set is $\{2^m - 1, 2^m - 2^0 - 1, \cdots, 2^m - 2^{m-1} - 1\}$. The total span

$$\begin{aligned} m_G &= \sum_{i=1}^{m+1} (R_i - L_i + 1) & (4.19) \\ &= \sum_{i=1}^{m+1} (R_i + 1) - \sum_{i=1}^{m+1} L_i & (4.20) \\ &= \sum_{i=0}^{m-1} (2^m - 2^{i+1}) + 2^m & (4.21) \\ &= (m-1)2^m + 2 & (4.22) \end{aligned}$$

From 4.11

$$\begin{aligned} LTC &\geq \frac{(m-1)2^m+2}{n} - \log_2 R && (4.23)\\ &= m-1+\frac{2}{n}+m-\log_2(m+1). && (4.24)\end{aligned}$$

Thus

$$\lim_{n\to\infty}\frac{LTC}{ACG} \geq 4$$

which coincides with the exact value. But (4.6) gives 3 as a lower bound.

The following theorem is a stronger result than Theorem 3 in [44], with a simpler proof. It shows that we can also obtain lower bounds on other trellis complexities as well.

Theorem 4.2 *Let* $s_{max} = \max\{s_i\}$, *then*

$$s_{max} \geq \frac{k(d-1)}{n-1}. \tag{4.25}$$

Proof:

$$\begin{aligned} \sum_{i=0}^{n} s_i &= nk + k - \sum_{i=0}^{n} f_i - \sum_{i=0}^{n} p_i && (4.26)\\ &= nk+k-\sum_{i=1}^{n} f_i - \sum_{i=0}^{n-1} p_i - k - k && (4.27)\\ &= \sum_{i=1}^{n} e_i - k = m_G - k && (4.28)\end{aligned}$$

Since $s_0 = s_n = 0$, we have

$$(n-1)s_{max} \geq \sum_{i=0}^{n} s_i = m_G - k \geq kd - k, \tag{4.29}$$

i.e.,

$$s_{max} \geq \frac{k(d-1)}{n-1} > \frac{k(d-1)}{n}. \tag{4.30}$$

■

4.2.3 The asymptotic analysis

To study the asymptotic behavior of the LTC–ACG ratio, we first note that in [43], some asymptotic results on s_{max} were presented. These results are closely related to the discussion in this chapter. In the sequel, S_n is the symmetric permutation group of order n. $R_{max}(x)$ is the McEliece–Rodemich–Rumsey–Welch [63] upper bound, given by

$$R_{max}(x) \leq \min_{0 \leq u \leq 1-2x} 1 + g(u^2) - g(u^2 + 2xu + 2x), \tag{4.31}$$

where

$$g(u) = H_2((1 - \sqrt{1-x})/2), \tag{4.32}$$

and $H_2(x)$ is the binary entropy function defined as

$$H_2(x) = -x \log_2 x - (1-x) \log_2 (1-x). \tag{4.33}$$

$R_{max}(x)$ is strictly less than any other known upper bound, thus providing the best bound in the following theorems.

Theorem 4.3 [43] *For any sequence of block codes with parameters (n, k_n, d_n), if $\lim_{n\to\infty} \frac{k_n}{n} = R$, $\lim_{n\to\infty} \frac{d_n}{n} = \delta$ and $R\delta \neq 0$, then*

$$\lim_{n\to\infty} \inf \frac{s_{max}}{n} \geq \frac{R - R_{max}(L\delta)}{(L-1)} \tag{4.34}$$

for any integer $L \geq 2$.

■

In the original statements in [43], s_{max} is replaced by

$$s^* = \min\{s_{max} : S_n \text{ acts on the column vectors}\}.$$

However, there is no real difference.

Theorem 4.4 *Let C be a sequence of block codes with parameters (n, k_n, d_n) such that $\lim_{n\to\infty} \frac{k_n}{n} = R$, $\lim_{n\to\infty} \frac{d_n}{n} = \delta$ and $R\delta \neq 0$, then*

$$\liminf_{n\to\infty} \frac{LTC}{ACG} \geq \max_{L\geq 2} \frac{R - R_{max}(L\delta)}{R\delta(L-1)}. \tag{4.35}$$

Proof: Since $s_i \leq e_i \leq s_i + 1$,

$$2^{s_{max}} \leq \sum_i 2^{e_i} = |E| \leq n2^{s_{max}+1}.$$

We have

$$s_{max} - \log k \leq LTC \leq s_{max} + 1 + \log \frac{n}{k}$$

which implies

$$\lim_{n\to\infty} \frac{LTC}{n} = \lim_{n\to\infty} \frac{s_{max}}{n}. \tag{4.36}$$

Hence

$$\begin{aligned} \liminf_{n\to\infty} \frac{LTC}{ACG} &= \frac{\lim_{n\to\infty} \inf \frac{LTC}{n}}{\lim_{n\to\infty} \frac{ACG}{n}} && (4.37)\\ &= \frac{\lim_{n\to\infty} \inf \frac{s_{max}}{n}}{\lim_{n\to\infty} \frac{kd}{n^2}} && (4.38)\\ &\geq \max_{L\geq 2} \frac{R - R_{max}(L\delta)}{R\delta(L-1)}. && (4.39) \end{aligned}$$

■

Corollary 4.2

$$\lim_{n\to\infty} \inf \frac{LTC}{ACG} \geq 2 \tag{4.40}$$

Proof: By the Plotkin bound, $\delta \leq \frac{1}{2}$. Suppose $\delta = 0.5$, then $L\delta \geq \frac{1}{2}$. Note that $R_{max}(x) = 0$ if $x \geq 0.5$. With $L = 2$, (4.35) becomes:

$$\lim_{n\to\infty} \inf \frac{LTC}{ACG} \geq 2. \tag{4.41}$$

If $\delta < 0.5$, let $L = \lceil \frac{1}{2\delta} \rceil$. It is easy to check that $L\delta \geq \frac{1}{2}$ and $L > 1$. Substituting L in (4.35), we have

$$\begin{aligned} \lim_{n\to\infty} \inf \frac{LTC}{ACG} &\geq \frac{1}{\delta(L-1)} && (4.42) \\ &\geq \frac{1}{\delta(\frac{1}{2\delta}+1-1)} && (4.43) \\ &= 2. && (4.44) \end{aligned}$$

■

Unfortunately, (4.40) does not cover *Example 1* to *Example 3* and many popular codes, including higher order RM and BCH codes. Combining (4.6) and (4.40) gives the following asymptotic lower bound for linear codes.

Theorem 4.5 *For any sequence of linear codes with fixed R, δ,*

$$\lim_{n\to\infty} \inf \frac{LTC}{ACG} \geq \begin{cases} 2 & \textit{if } R\delta > 0 \\ 1 & \textit{if } R\delta = 0 \end{cases}. \tag{4.45}$$

■

4.3 The convolutional codes

4.3.1 The LTC-ACG ratio

A convolutional code is of infinite length. But if we retain the first M rows of the scalar generator matrix and ignore the rest, the resulting subcode is in fact a block code. If M is large enough, the minimal trellis (without column permutation) of the block code mainly consists of the trellis modules. This simple observation allows us to approximate convolutional codes by block codes.

The sth truncation of an (n, k) convolutional code is an $[n(f_k + s), ks]$ block code whose generator matrix G^s is just the first ks rows of the scalar generator matrix G^*. Denote the sth truncation of the code C by C^s. If G^* corresponds to the trellis-canonical generator matrix for the convolutional code, G^s is a MSGM for C^s.

Lemma 4.3 *Let LTC_* be the trellis complexity index of code $*$. (The subscript is omitted when understood.) Then*

$$\lim_{s\to\infty} LTC_{C^s} = LTC_C.$$

Proof: If s is sufficiently large, the minimal trellis for C^s contains $s - f_k$ copies of the trellis module, in addition to a head and a tail whose matrix forms are:

$$G_{head} = \begin{pmatrix} G_0 & G_1 & G_2 & \cdots & G_{L-1} \\ & G_0 & G_1 & \cdots & G_{L_2} \\ & & \cdots & \cdots & \cdots \\ & & & & G_0 \end{pmatrix}$$

$$G_{tail} = \begin{pmatrix} G_L & & & & \\ G_{L-1} & G_L & & & \\ \cdots & \cdots & \cdots & \cdots & \\ G_1 & \cdots & G_{L-2} & G_{L-1} & G_L \end{pmatrix}$$

Then

$$LTC_{C^s} = \log_2 \frac{(s - f_k)|E| + |E_{tail}| + |E_{head}|}{ks}$$

where $|E_{tail}|$, $|E_{head}|$ and $|E|$ are the number of edges in the tail, the head and one trellis module, respectively.

Apparently,

$$0 < |E_{tail}| < f_k|E|, \tag{4.46}$$

$$0 < |E_{head}| < f_k|E|. \tag{4.47}$$

The trellis complexity index for C^s can thus be bounded by

$$\log_2 \frac{(s - f_k)|E|}{ks} < LTC_{C^s} < \log_2 \frac{(s + f_k)|E|}{ks}. \tag{4.48}$$

Note that n, k and f_k are fixed.

Let s go to infinity:

$$\lim_{s\to\infty} \log_2 \frac{s - f_k}{s} + \log_2 \frac{|E|}{k} < \lim_{s\to\infty} LTC_{C^s} < \log_2 \frac{|E|}{k} + \lim_{s\to\infty} \log_2 \frac{s + f_k}{s} \tag{4.49}$$

$$LTC_C = \log_2 \frac{|E|}{k} \leq \lim_{s\to\infty} LTC_{C^s} \leq \log_2 \frac{|E|}{k} = LTC_C \tag{4.50}$$

■

Theorem 4.6 *For any convolutional code C,*

$$LTC \geq ACG - \log_2 R. \tag{4.51}$$

Proof: Since every codeword of C^s is a codeword of C^{s+1} and C,

$$d^s \geq d^{s+1} \geq d_{free}. \tag{4.52}$$

From Theorem 4.1,

$$LTC_{C^s} \geq ACG_{C^s} - \log_2 \frac{ks}{n(f_k + s)}.$$

Thus

$$\begin{aligned} LTC &= \lim_{s \to \infty} LTC_{C^s} && (4.53) \\ &\geq \lim_{s \to \infty} \frac{k}{n} d^s - \log_2 R && (4.54) \\ &\geq ACG - \log_2 R. && (4.55) \end{aligned}$$

■

Corollary 4.3 *For any convolutional code,*

$$\frac{LTC}{ACG} > 1. \tag{4.56}$$

4.3.2 Asymptotic analysis

In this section, we shall investigate the asymptotic behavior of the LTC–ACG ratio for convolutional codes. However, what asymptotic analysis means here is a little different. For block codes, we examine a sequence of codes whose length, dimension and minimum distance all are variable, but whose relative distance and rate converge to some number, respectively. For convolutional codes, we study properties of a sequence of codes of variable memory f_k (usually in increasing order), but with fixed values of n and k. Figure 4.1 shows that for the best-known $(2, 1, f_k)$ convolutional codes, the LTC–ACG ratio approaches, and then crosses, the line of slope 2, as the memory increases. In fact, we can prove that this is true for any nondegenerate sequence of convolutional codes. To do this, we will continue to employ block codes to "approximate" convolutional codes.

Consider the sequence of block codes $C^s_{f_k}$, which are the sth truncations of an (n, k) convolutional code C_{f_k} with free distance d_{f_k}. If f_k and s approach infinity independently, we can hardly expect a meaningful result, so hereafter let $s = \lambda f_k$ for some constant $\lambda > 1$.

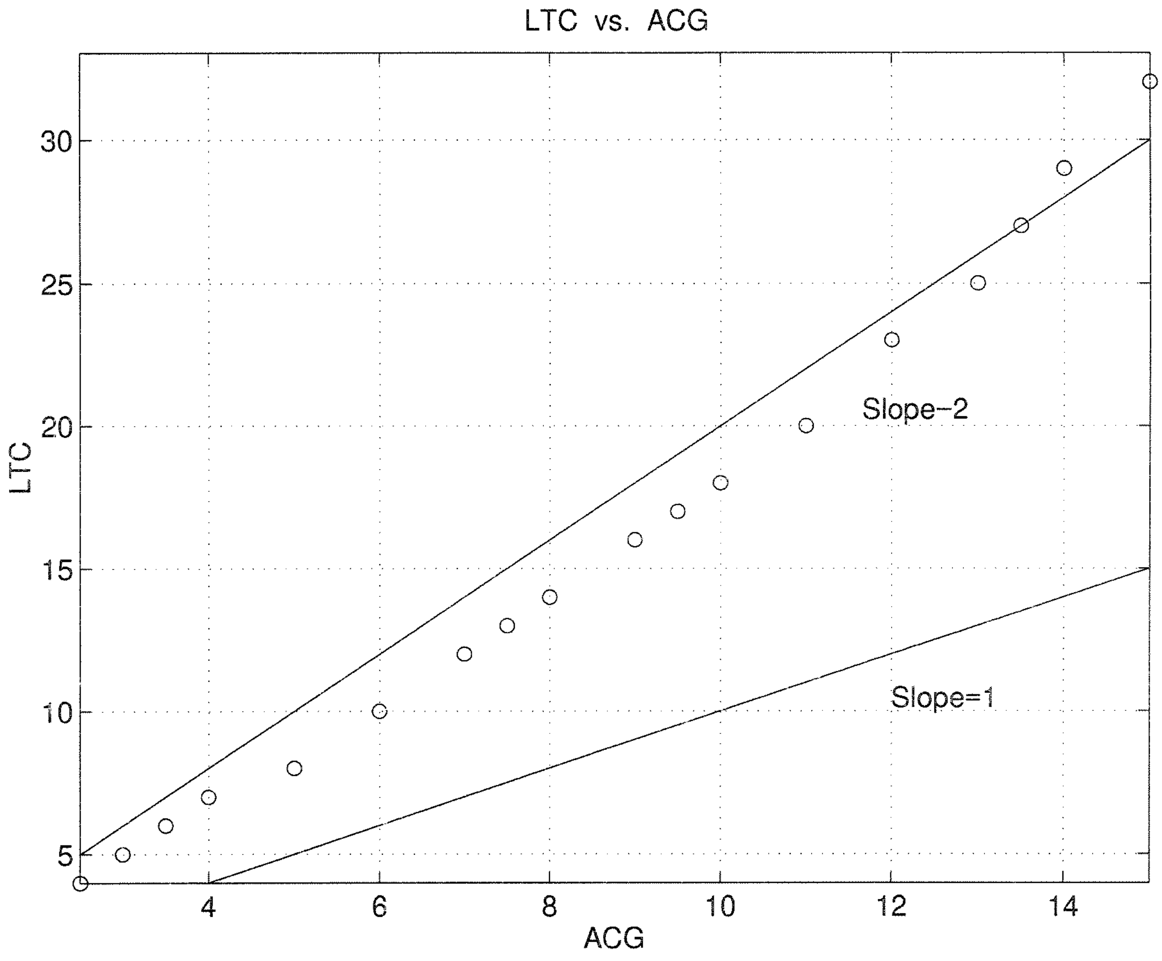

Figure 4.1: The plot of LTC versus ACG for optimal $(2, 1, f_k)$ codes. Small circles represent codes with increasing memory.

For this sequence, the rate is convergent:

$$\begin{aligned}
\lim_{e_k\to\infty} \frac{ks}{n(f_k+s)} &= \frac{\lambda}{1+\lambda}\cdot\frac{k}{n} && (4.57)\\
&= \frac{\lambda}{1+\lambda}R && (4.58)\\
&> 0. && (4.59)
\end{aligned}$$

The asymptotic relative distance would be

$$\begin{aligned}
\delta &= \lim_{f_k\to\infty} \frac{d_{C^s_{f_k}}}{n(f_k+s)} && (4.60)\\
&= \frac{1}{n(1+\lambda)} \lim_{f_k\to\infty} \frac{d_{C^s_{f_k}}}{f_k} && (4.61)\\
&= \frac{1}{n(1+\lambda)} \lim_{f_k\to\infty} \frac{d_{f_k}}{f_k}. && (4.62)
\end{aligned}$$

Definition: A sequence of convolutional codes is *degenerate* if

$$\lim_{f_k\to\infty} \inf \frac{d_{f_k}}{f_k} = 0. \qquad (4.63)$$

For a nondegenerate sequence whose relative distance is convergent, from Theorem 4.5, we have

$$\begin{aligned}
\lim_{f_k\to\infty} \inf \frac{LTC_{C^s_{f_k}}}{n(f_k+s)} &= \frac{1}{n(1+\lambda)} \lim_{f_k\to\infty} \inf \frac{LTC_{C^s_{f_k}}}{f_k}\\
&\geq 2 \lim_{f_k\to\infty} \frac{ACG_{C^s_{f_k}}}{n(f_k+s)}. && (4.64)
\end{aligned}$$

Lemma 4.4

$$\lim_{f_k\to\infty} \frac{LTC_{C^s_{f_k}} - LTC_{f_k}}{f_k} = 0 \qquad (4.65)$$

Proof: From (4.49) we have

$$\lim_{f_k\to\infty} \frac{\log_2 \frac{\lambda-1}{\lambda}}{f_k} < \lim_{f_k\to\infty} \frac{LTC_{C^s_{f_k}} - LTC_{f_k}}{f_k} < \lim_{f_k\to\infty} \frac{\log_2 \frac{\lambda+1}{\lambda}}{f_k},$$

i.e.,

$$|\lim_{f_k\to\infty}\frac{LTC_{C^s_{f_k}}-LTC_{f_k}}{f_k}|<\lim_{f_k\to\infty}\frac{c}{f_k}\to 0,$$

where c is a constant. ■

Theorem 4.7 *For any nondegenerate sequence of convolutional codes with fixed n and k,*

$$\lim_{f_k\to\infty}\inf\frac{LTC_{f_k}}{ACG_{f_k}}\geq 2.$$

Proof: First divide the sequence into subsequences such that the relative distance of each subsequence is convergent. For each subsequence, from Lemma 4.4 and equation (4.64)

$$\begin{aligned}
\lim_{f_k\to\infty}\inf\frac{LTC_{f_k}}{ACG_{f_k}} &= \frac{\lim_{f_k\to\infty}\inf\frac{LTC_{f_k}}{f_k}}{R\lim_{f_k\to\infty}\frac{d_{free}}{f_k}} && (4.66)\\
&= \frac{\lim_{f_k\to\infty}\inf\frac{LTC_{C^s_{f_k}}}{f_k}}{R\lim_{f_k\to\infty}\frac{d_{free}}{f_k}} && (4.67)\\
&\geq \frac{2n(1+\lambda)\lim_{f_k\to\infty}\frac{ACG_{C^s_{f_k}}}{n(f_k+s)}}{R\lim_{f_k\to\infty}\frac{d_{free}}{f_k}} && (4.68)\\
&= \frac{2\lambda}{1+\lambda}\frac{Rn(1+\lambda)\delta}{R\lim_{f_k\to\infty}\frac{d_{free}}{f_k}} && (4.69)\\
&\geq \frac{2\lambda}{1+\lambda}\frac{\lim_{f_k\to\infty}\frac{d_{free}}{f_k}}{\lim_{f_k\to\infty}\frac{d_{free}}{f_k}} && (4.70)\\
&= \frac{2\lambda}{1+\lambda}>2 && (4.71)
\end{aligned}$$

since λ can be arbitrarily large.

The theorem follows by taking the minimum over all subsequences. ■

4.4 More examples

In the appendix, we list the ACG and the LTC for a large number of "good" convolutional codes, and a few block codes. Plots are also presented to offer better visual understanding of the LTC-ACG ratio. In Fig. 4.2 we show a scatter plot of collected (ACG,LTC) pairs to provide a direct comparison. In all these plots, there is one line of slope one and one of slope two. Theorem 4.1 and 4.6 guarantee that all points are above the line of slope one. It is interesting that the plots reveal that for most "good" convolutional codes of relative small memory, the LTC-ACG ratios fall in the 1.5 $\sim$ 2.0 region. For example, for the (8,4,3,8) PUM code, it is 1.68. For the "NASA" standard (2,1,6,10) code, the ratio is 1.60 which is smaller than any other known "good" codes. For block codes, the LTC-ACG ratio is generally above the slope two line, implying worse efficiency. This partially supports the general belief that block codes are "worse" than convolutional codes.

4.5 Conclusion

In this section, we studied the performance/complexity tradeoff for block and convolutional codes. An interesting analogy to real life is

$$\frac{\text{Investment}}{\text{Profit}} \simeq \frac{LTC}{ACG}.$$

We first obtained bounds on the LTC-ACG ratio by examining the span distribution for block codes. The connection of these bounds with the work of [43] has been discussed. The powerful span analysis produced even stronger result.

Intuitively, all theorems regarding block codes can be generalized to convolutional codes, and we did use block codes to approximate convolutional codes. But caution is still needed to avoid plausible arguments. The generalization is thus subtle. Although empirical study suggested that block codes usually have worse efficiency than convolutional codes, it is too early to conclude that block codes are worse.

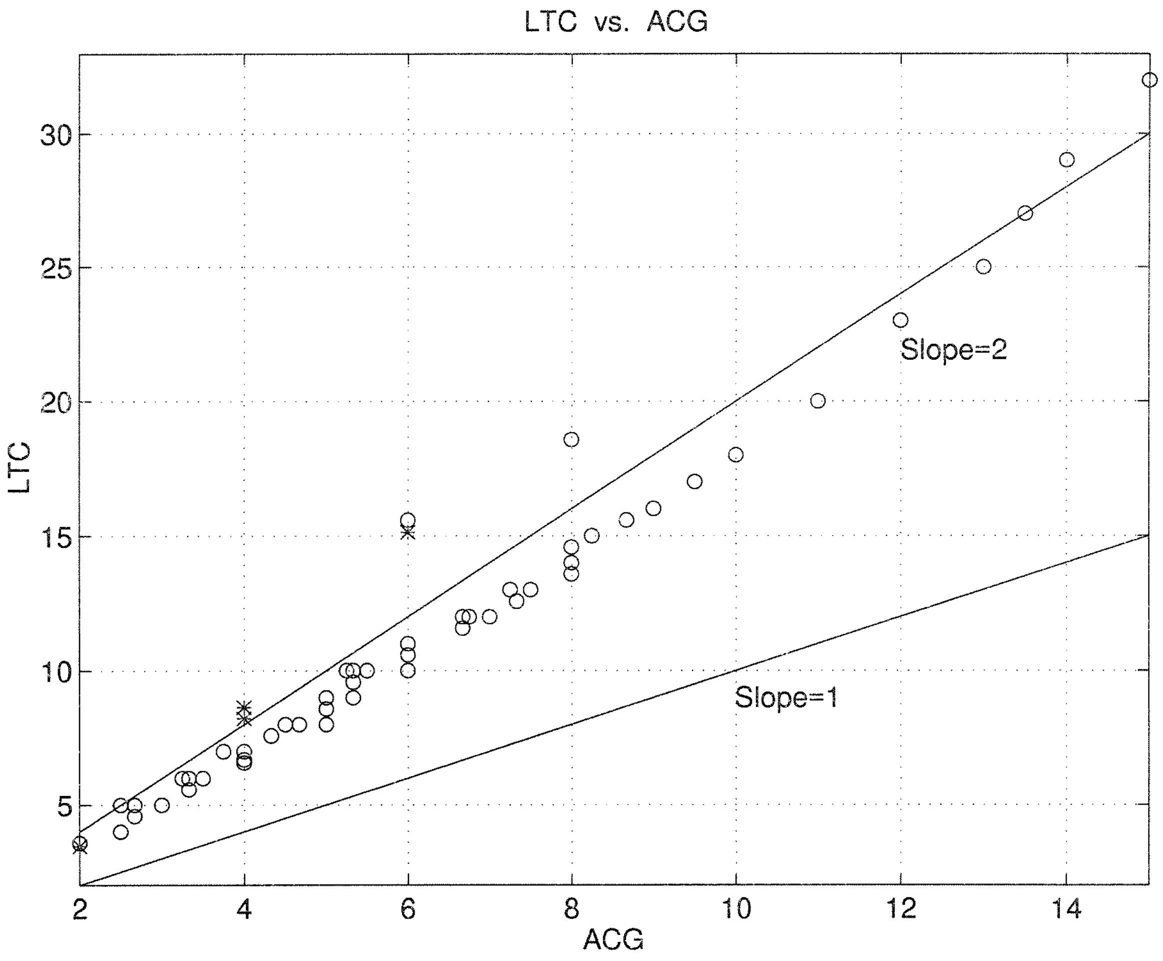

Figure 4.2: A scatter plot of (ACG,LTC) pairs for the codes in the Appendix. Convolutional codes are indicated with small circles and block codes with $*$'s.

Chapter 5 Trellis Decoding Complexity

5.1 Introduction

Since its introduction in 1967, the Viterbi Algorithm (VA) has been recognized as an attractive decoding technique for convolutional codes. Unfortunately, the complexity of the standard VA prohibits its use in many applications. Recently, extensive research has been done on reducing the decoding complexity of Viterbi–like algorithms for block codes [22, 37, 38, 40, 45, 57, 58, 64]. This in turn spurred research on low complexity maximum likelihood decoding for convolutional codes [33, 60, 61, 72].

By decoding complexity, we mean the number of addition equivalent operations required per decoded information bit. In chapter 3 (also see [60]), a complexity lower bound for bit–by–bit Viterbi decoding was given. A general algorithm for achieving this lower bound was proved. However, even lower decoding complexity is achievable by grouping consecutive channel outputs and using group–by–group Viterbi decoding. To achieve this, we propose a Hybrid Trellis Decoding Algorithm (HTDA), which exploits sectionalization techniques for block codes [45]. Hole and Ytrehus recently proposed a Two–Step Trellis Decoding Algorithm (TSTDA) for decoding Partial Unit Memory (PUM) convolutional codes [33]. The TSTDA also takes advantage of grouping channel outputs. But the HTDA is applicable to most codes while the TSTDA is efficient only for codes with parallel branches (e.g., PUM codes). We shall see that the HTDA is more efficient than the TSTDA even for some PUM codes.

In section 5.2, the basic Viterbi decoding algorithm is briefly analyzed. Section 5.3 contains a concise description of the TSTDA. In Section 5.4, the HTDA is detailed. An example is presented to show that the HTDA is more efficient than the TSTDA for PUM codes, which in turn is superior to the standard VA for comparable punctured codes [33]. In section 5.5, we shall discuss the bounds on the decoding complexity of the HTDA.

5.2 The basic algorithm

An optimal decoder (Figure 5.1) produces an estimate $\mathbf{v}$ of the transmitted sequence $\mathbf{u}$ given the channel output $\mathbf{r}$, such that the *a posteriori* probability $P(\mathbf{v}|\mathbf{r})$ is maximized. If all codewords are equally likely to be transmitted, this is tantamount to maximizing $P(\mathbf{r}|\mathbf{v})$, the likelihood of $\mathbf{r}$ given $\mathbf{v}$.

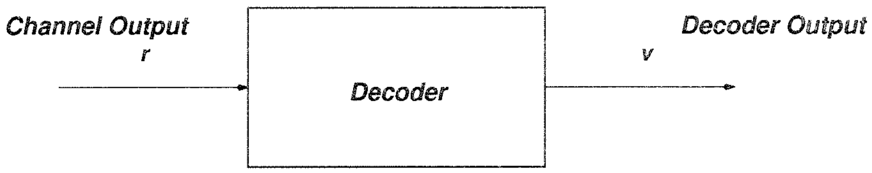

Figure 5.1: A General Decoder.

Assume a memoryless channel which outputs real numbers. Let $f_i(\cdot)$ be the channel transition probability densities, where $i \in GF(2)$. The optimal decoding strategy may as well maximize any monotonic increasing function of $P(\mathbf{r}|\mathbf{v})$. For example,

$$D(\mathbf{r},\mathbf{v}) \overset{\text{def}}{=} \sum_i (-1)^{v_i} m(r_i) = 2P(\mathbf{r}|\mathbf{v}) - \sum_i (\log f_0(r_i) + \log f_1(r_i)) \tag{5.1}$$

is such a function, where $m(r_i) = \log(f_0(r_i)/f_1(r_i))$. Hereafter we assume the optimal decoder input is $m(r_i)$. Then a simplified VA can be employed in decoding.

The Viterbi algorithm is best understood by examining the trellis structures for convolutional codes. In Chapter 3 we defined the minimal trellis for convolutional codes. However, the internal structure of a trellis module may vary. For example, the trellis module considered in Chapter 3 is of depth[1] n where the label length of each edge is one (the primitive trellis). A depth–1 trellis module is one whose label lengths are n for all edges. Figure 5.2 shows two trellis modules for the same code. A general trellis module may have m depths and multiple label lengths, provided that $\sum_{i=1}^{m} ll_i = n$, where ll_i is the label length of depth i. (This is analogous to the trellis

[1]The meaning of depth and stage in this Chapter is different from that in Chapter 1.

"segmentation" for block codes [45].) On the other hand, the trellis complexity of any "segmentation" of the trellis module is completely determined by the minimal trellis structure proposed in chapter 3.

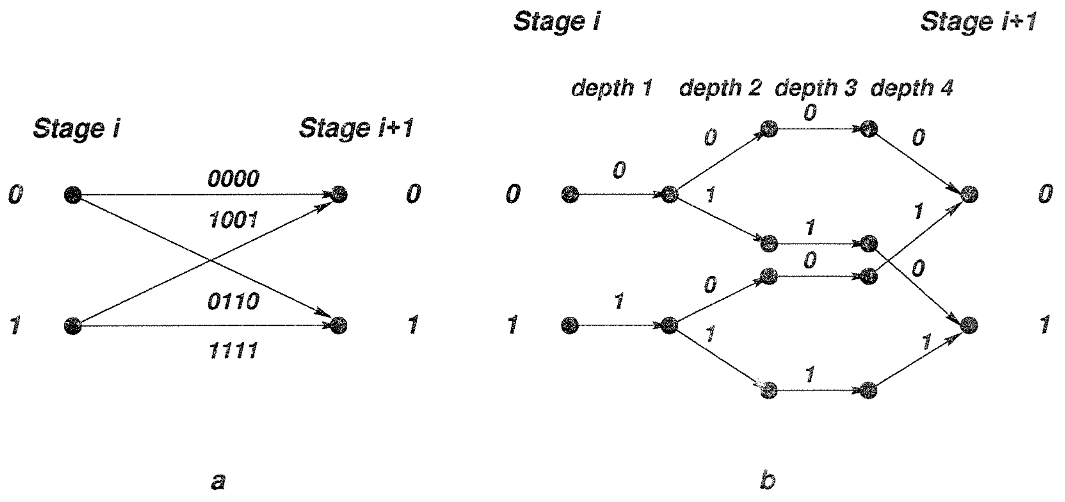

Figure 5.2: a: Depth–1 trellis module. b: Depth–4 trellis module.

Now consider the simple trellis module shown in Figure 5.3. There are four states, 00, 01, 10 and 11. Let $\mathbf{l}_{11}^{01}$ be the label of the edge from state 01 to state 11. d_{00}^{i} denotes the distance accumulated up to stage i at state 00. To update the distance of state 00 at stage $i+1$ with decoder input $\mathbf{r}$, first the edge metrics $\mathcal{D}(\mathbf{l}_{00}^{00}, \mathbf{r})$ and $\mathcal{D}(\mathbf{l}_{00}^{10}, \mathbf{r})$ are computed and added to d_{00}^{i} and d_{10}^{i}, respectively. The VA chooses the smaller number as the updated d_{00}^{i} and the corresponding path as the survivor. After all four states have been updated, the VA either chooses a survivor as its output or repeats the same procedure on the identical module with a different received vector.

There are three kinds of computations (measured by the number of addition equivalent operations) involved in the VA: the computation of edge metrics, P_E; the accumulation of distance, P_A; and the updating of state distance, P_U. If one trellis module represents k information bits, we define the decoding complexity of the VA

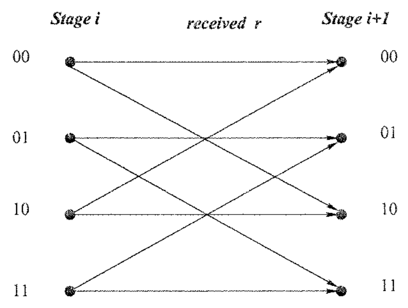

Figure 5.3: Viterbi Algorithm on a simple trellis module.

as

$$\gamma_{VA} = \frac{P_E + P_A + P_U}{k}. \tag{5.2}$$

For a depth-n trellis module, an algorithm to compute γ_{VA} was given in chapter 3 (also see [60, 61]). However, more efficient decoding is possible using general trellis modules. The TSTDA and the HTDA introduced in this paper use depth-1 trellis modules, though HTDA is applicable to general trellis modules.

5.3 Two-step trellis decoding

In [33], a two-step trellis decoding procedure for high rate convolutional codes was presented. A set of edges is called a parallel branch if every edge originates from the same state and ends on the same state as well. The TSTDA first decodes all parallel branches by a local Viterbi decoder, then constructs a simplified trellis module where every parallel branch is replaced by the survivor chosen by the local Viterbi Decoder. The VA is then applied to this reduced trellis module to complete the decoding.

The efficiency of the TSTDA is heavily dependent on the existence of parallel branches. The higher the percentage of parallel branches, the more efficient the algorithm. Thus it is suitable for partial unit memory convolutional codes, which have many parallel branches.

5.4 Hybrid trellis decoding algorithm

For convenience of analysis, we may divide trellis decoding into two distinct processes. The first is to compute edge metrics. The second is to accumulate and update state distances. In the TSTDA, the two processes are interweaved to achieve lower decoding complexity. In fact, keeping these two processes separated leads to more efficient decoding. Since the computation effort required by the second process is essentially determined by the infrastructure of the trellis module chosen, e.g., the number of edges and states used, little saving can be expected. We shall therefore focus on reducing the cost of edge metric computation.

5.4.1 Edge metric computation

Let the edge label set be $\mathcal{C}$, which is a $[n, k^*, d]$ linear block code. For each edge e, its metric is

$$\sum_{i=1}^{n}(-1)^{l_e^i} r_i, \tag{5.3}$$

where $\mathbf{l}_e$ is the edge label. For a single codeword in $\mathcal{C}$, $n-1$ additions are required to compute its metric. Since different codewords may share internal results, more efficient methods are available for computing the metrics [45]. For example, once the metric for (0010101) is known, the metric for (1101010) can be obtained by negation. Let $\theta_{\mathcal{C}}$ be the minimal number of addition equivalent operations required to compute all metrics. $\theta_{\mathcal{C}}$ is more than a function of n, k^* and d, but heavily depends on the infrastructure of the code.

5.4.2 State distance updating

This part of the computation is primarily determined by the number of edges and number of vertices in a trellis module. One exception is that when there exist parallel branches. In this case, the HTDA compares all edge metrics in a parallel branch and selects the smallest one before accumulating state distance rather than accumulating the state distance for every edge before comparing and selecting the smallest one.

Denote the reduced computation for distance accumulating and distance updating by P'_A and P'_U, respectively. The decoding complexity of the HTDA is then

$$\gamma_{HTDA} = \frac{\theta_{\mathcal{C}} + P'_A + P'_U}{k}. \tag{5.4}$$

5.4.3 One example

Consider the (8,4,3,8) PUM code with the trellis-canonical generator matrix

$$G(D) = \begin{pmatrix} 11111111 \\ 01010011 \\ 00110110 \\ 00001111 \end{pmatrix} + \begin{pmatrix} 00000000 \\ 11110000 \\ 01011100 \\ 01100110 \end{pmatrix} D \tag{5.5}$$

whose primitive trellis is demonstrated in Figure 5.4.

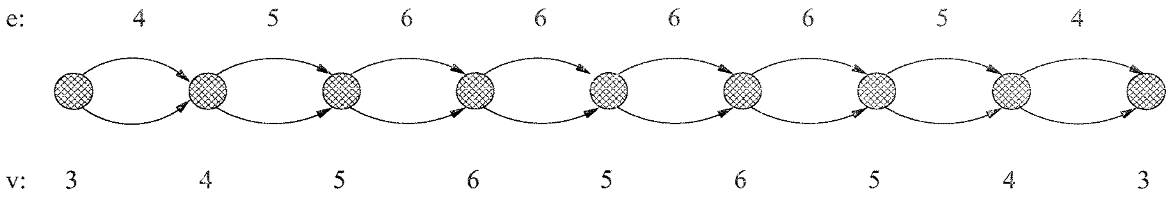

Figure 5.4: The simplified structure of the primitive trellis module for (8,4,3,8) PUM code. Each big vertex represents a set of vertices in the primitive trellis. The sequence above the figure is the edge dimension profile. The sequence below the figure is the state dimension profile.

By grouping the eight sections, a depth-1 trellis is obtained. There are 128 edges in this trellis and 2 parallel edges between any pair of vertices. The edge label set is a [8,5,2] block code whose generator matrix is:

$$\begin{pmatrix} 11111111 \\ 01010011 \\ 00110110 \\ 00001111 \\ 01100110 \end{pmatrix}.$$

Because of the presence of the all-one vector, it is enough to compute the metrics for the [8,4,2] code with generator matrix

$$\begin{pmatrix} 01010011 \\ 00110110 \\ 00001111 \\ 01100110 \end{pmatrix}.$$

We do this by computing the first four bits and the last four bits separately, then combining them using 16 additions. The first four bits form a [4,3,2] code and can be split again into two [2,2,1] codes. Each [2,2,1] code needs 2 additions. To get the metrics for the [4,3,2] code from the two [2,2,1] codes, 8 additions are enough. The last four bits form a [4,2,2] code. Similarly, 8 additions are enough to compute its metrics. Thus, to compute the edge metrics for the [8,5,2] code requires only 9 additions per decoded information bit. This number is smaller than that required by any algorithm in [45]. Because the two parallel edges in each parallel branch are complementary (defined in example 5.1), comparison is not needed to select the survivor. It turns out that the HTDA requires only 25 additions and 14 comparisons to decode each information bit, compared to 42 additions and 22 comparisons for the TSTDA [33] and 88 additions and 22 comparisons for the VA on a depth-n trellis [60]. Clearly the HTDA is significantly superior for this code.

5.5 Complexity bounds

In this section, we will investigate decoding complexity bounds for the HTDA. As we have seen, the HTDA consists of two parts: the computation of edge metrics, which depends on the label set and the state distance accumulating and updating which is determined by the structure of the trellis module. In [45], various techniques to compute edge metrics were presented. Each of them produces an upper bound for θ_C. However, the discussion in [45] is very specific. To improve their results, we will prove some general lower bounds as well as upper bounds for θ_C. These bounds are shown to be exact in some cases. The bounds do not depend on the computation mechanism employed. Since the computation of state distance accumulating and updating can be accurately computed from the trellis structure, we are not interested in bounds for this part of the computation.

5.5.1 Lower bounds on θ_C

To derive the bounds on edge computation, it is necessary to express the problem in rigorous mathematic language.

Problem Formulation: Given a subset (code) $C = \{c_1, c_2, \ldots, c_N\}$ of the vector space V_n over GF(2) and n real numbers $r_1, r_2, \ldots, r_n$, for each codeword $c_j = \{c_j^1, c_j^2, \ldots, c_j^n\} \in C$, let its metric $M_{c_j} = \sum_{i=1}^{n}(-1)^{c_j^i} r_i$. For an algorithm $\Lambda \in \mathcal{A}^*$ as in Figure 5.1, where $\mathcal{A}^*$ is the set of all algorithms using allowed operations (which are to be defined) to compute M_k, $k = 1, \ldots, N$, let the number of addition equivalent operations used be θ_Λ. Then $\theta_C = \min_{\Lambda \in \mathcal{A}^*} \theta_\Lambda$.

It is quite difficult to exactly compute θ_C in general, but not the bounds.

Example 5.1 *Gray Code.*

Suppose the code is the whole vector space V_n. There are 2^n codewords. Two codewords c_a and c_b are said to be complementary if $c_a^i + c_b^i = 1$ for any $i \in \{1, 2, \ldots, n\}$. Since $M_a + M_b = 0$, only one of them actually needs computing. Thus the 2^n vectors

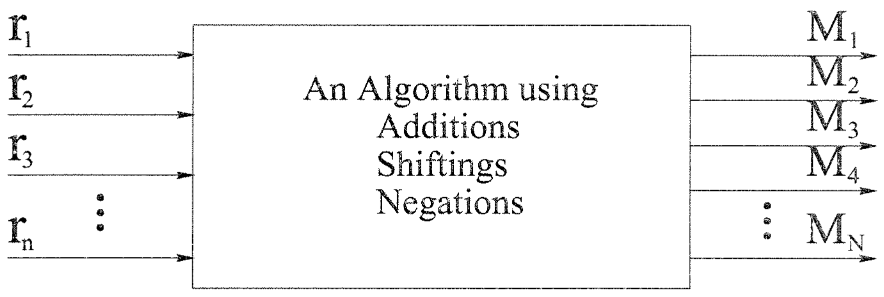

Figure 5.5: An algorithm to compute code metrics.

of V_n are divided into two disjoint sets, as shown for $n = 3$:

$$\begin{array}{cccc} & r_1 & r_2 & r_3 \\ c_1 & 1 & 0 & 0 \\ c_2 & 1 & 0 & 1 \\ c_3 & 1 & 1 & 1 \\ c_4 & 1 & 1 & 0 \\ & & \mathbf{A} & \end{array} \qquad \begin{array}{cccc} & r_1 & r_2 & r_3 \\ c_5 & 0 & 1 & 1 \\ c_6 & 0 & 1 & 0 \\ c_7 & 0 & 0 & 0 \\ c_8 & 0 & 0 & 1 \\ & & \mathbf{B} & \end{array}$$

In group **A** the codewords are in Gray code order. This requires 2 additions to compute M_{c_1}. Since $M_{c_2} = M_{c_1} - 2r_3$, $M_{c_3} = M_{c_2} - 2r_2$, $M_{c_4} = M_{c_3} + 2r_3$, only one addition is needed to compute each of them. It turns out that only 5 additions are necessary to compute all 8 codeword metrics. In general we have

$$\theta_C \leq 2^{n-1} + n - 2. \tag{5.6}$$

We shall prove that this is also the best we can do.

However, it is not clear yet what kinds of operations are legal in this computing. In the above example, only three elementary operations, i.e., addition, negation and shifting, were used. Hereafter, we shall assume that only these three operations are legal. This assumption is reasonable because in all modern computers, the addition of two real numbers can be regarded as a unit of complexity. (In some cases this complexity may be partially transferred to hardware.) Since shifting does not produce any metric, we incorporate it with the other two operations. Figure 5.6 depicts the two major operations, addition and negation.

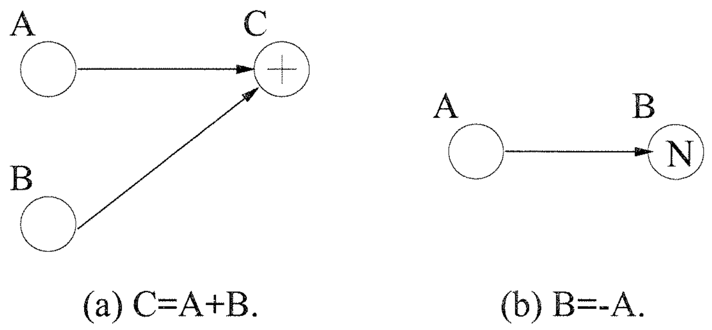

Figure 5.6: a: Addition. b: Negation.

Theorem 5.1 *Let C be a binary code consisting of N vectors of length n, and suppose there are k pairs of codewords which are complementary. Then*

$$\theta_C \geq N + n - k - 2. \tag{5.7}$$

Proof: For any codeword c, we may construct a directed tree T_c showing the process of computing M_c for a given algorithm. For example, if $c = 1001$, Figure 5.3 is such a tree for some algorithms. The vertices representing the n real numbers are initial vertices while all others are called operation vertices. The vertex representing M_c is called the root. Each operation vertex corresponds one operation and the result. Every vertex has at most one out degree and two in degrees. Each tree is the minimal representation of the process of computing M_c in the sense that deleting any edge or vertex won't produce M_c.

We number all trees such that the root value of T_j can't be generated by T_i $1 \leq i < j$ or their subtrees. This ordering is always possible. This implies that any two trees have at least one different operation vertex.

Now we count the number of distinct operation vertices in the forest of all trees. In T_1, there are at least $n-1$ operation vertices. Note that the root of T_2 is not in T_1, the root of T_3 is not in T_1 or T_2, or in general the root of T_j is not in T_i for any $i < j$; all these roots are distinct operation vertices. Thus there are at least $N+n-2$

distinct vertices in the forest. Since in the best case, k roots are negation operations, the theorem follows. ■

Corollary 5.1 *Let C be a $[n, k, d]$ linear code, then*

$$\theta_C \geq \begin{cases} 2^k + n - 2 & \text{if } \mathbf{1} \notin C \\ 2^{k-1} + n - 2 & \text{if } \mathbf{1} \in C. \end{cases} \tag{5.8}$$

Example 5.2 *Hadamard Code.*

Consider the order–2^n Hadamard code. It has 2^n codewords and distance 2^{n-1}. The DFT algorithm requires $n2^n$ additions to compute its metrics. The lower bound of Theorem 5.1 is $2^{n+1} - 2$ additions.

Example 5.3 *The $[8, 5, 2]$ code in section 5.4.3.*

The lower bound is 22 additions, compared to 32 additions in the actual algorithm.

5.5.2 Upper bounds on θ_C

Any algorithm gives an upper bound. In [44], four different upper bounds were given. Since we at most compute the metrics for the whole vector space, (5.6) is a simple upper bound. The "brute force" computing algorithm which computes each metric independently is obviously the worst case. For each codeword, exactly $n-1$ additions are necessary. Assuming there are k pairs of complementary codewords, only $N - k$ metrics need to be computed. Therefore, we have

Theorem 5.2 *Let C be a (n, N, d) code with k pairs of complementary codewords. Then*

$$\theta_C \leq \min\{2^{n-1} + n - 2, (n-1)(N-k)\}. \tag{5.9}$$

Example 5.4 *The whole vector space.*

For the whole vector space, assuming $k = n$ and $\mathbf{1} \in C$ in Corollary 5.1, the upper bound and the lower bound coincide, which proves the optimality of Example 5.1. For a pair of complementary vectors, Theorem 5.1 gives the lower bound $n-1$, which is also identical with the upper bound given by Theorem 5.2.

5.6 Bounds on arbitrary segmented trellises

Consider a minimal trellis module whose $\hat{G}$ is a $m \times n$ matrix of row dimension k. The state dimensions are $s_0, s_1, \cdots, s_n = s_0$ and the edge dimensions are $e_1, e_2, \cdots, e_n$. If the minimal trellis module is partitioned into L sections, and the state dimensions are $s_{t_0} = s_0, s_{t_1}, s_{t_2}, \cdots, s_{t_{L-1}}, s_{t_L} = s_n$, then edge dimension in section l is

$$e_l^* = e_{t_l} + \sum_{i=t_{l-1}+1}^{t_l-1} (e_i - s_i). \tag{5.10}$$

Assume the label set of section l is an $[n_l, k_l]$ code C_l, then the decoding complexity of this sectionalized trellis module is

$$\sum_{i=1}^{L} (2^{e_i^*+1} + \theta_{C_i} - 2^{s_{t_i}}) \tag{5.11}$$

addition equivalent operations.

There are parallel edges in the trellis only if there are constant rows in the polynomial generator matrix. Suppose there are 2^{P_l} parallel edges between some pairs of vertices in section l, the decoding complexity would be

$$\sum_{i=1}^{L} (2^{e_i^*} + 2^{e_i^*-P_i} + \theta_{C_i} - 2^{s_{t_i}}) \tag{5.12}$$

addition equivalent operations. In the case these 2^{P_l} edges in each parallel branch are complementary pairs, the decoding complexity reduce to

$$\sum_{i=1}^{L} (2^{e_i^*-1} + 2^{e_i^*-P_i} + \theta_{C_i} - 2^{s_{t_i}}). \tag{5.13}$$

5.7 Conclusion

In this chapter, we have studied the practical trellis decoding complexity and its relation with the abstract trellis complexity. A Hybrid Trellis Decoding Algorithm (HTDA) for convolutional codes was proposed and compared with Two–Step Trellis Decoding Algorithm (TSTDA) [33]. The HTDA is shown to be superior, in terms of decoding complexity, to the TSTDA and the standard Viterbi Algorithm (VA) for comparable punctured codes. Upper and lower bounds on the decoding complexity of the HTDA are also presented. These bounds are exact in some cases.

Chapter 6 Conclusion

Shannon's original theory showed that channel capacity is attainable via coding. However, it did not provide clues about how to construct such codes and how to decode. It has thus been a major focus of communication research to devise codes and (especially recently) low complexity decoding algorithms that approach the promised (by Shannon) performance limits.

Observing the prolific research on the trellis complexity of error-correcting codes and lattices and the interplay of coding and computational complexity, the IEEE Information society published a special issue on codes and complexity. (November, 1996, IEEE Trans. Inform. Theory.) This special issue includes 32 papers covering five topics: the trellis complexity of block codes and lattices, dynamical systems and convolutional codes, decoding algorithms and performance/complexity tradeoff, coding and computational complexity and symbolic dynamics and constrained codes. All these papers are well-written and mostly self-contained. Any reader interested in doing research in the above areas is strongly recommended to read the special issue.

Finally, we now present some interesting open problems to conclude this thesis.

The permutation trellis complexity of block codes.

- **Open Problem 1:** In Chapter 2, we showed that the permutation trellis complexities are closely related to the permutation total span. We also developed algorithms minimizing the permutation total span of block codes. A number of optimal or good permutations for many codes thus were obtained. Behind the success, however, it has not been proved if other permutation trellis complexities could be minimized by minimizing the permutation total span. We do not even know if the total span and the edge complexity could be minimized by one permutation for an arbitrary code. Any proof or counter–example to the above questions is highly desired.

- **Open Problem 2:** In [36], the authors proved that the problem of finding a column permutation that minimizes the number of vertices at a given depth in the minimal trellis for a binary linear block code is NP–complete. Is the problem of finding one optimal coordinate permutation (see Chapter 2) for a binary linear block code also NP–complete? A proof of the above statement is highly welcome. Also, more effort should be devoted to find more powerful fast algorithms to find the optimal permutations for various codes.

The minimal trellis structure of convolutional codes.

- **Open Problem 3:** In the computer-aided search for the permutation edge complexity of the (8,4,3,8) partial unit memory code, surprisingly, we found that each of the $8! = 40\ 326$ coordinate permutations had minimal edge complexity either 120 or 104. This strongly suggests an equivalence among permutations, which if understood theoretically, could make it simpler to find the permutation trellis complexity and even one permutation achieving it. Block codes also demonstrate this phenomenon.

The complexity/performance tradeoff.

- **Open Problem 4:** The plots in chapter 4 and the tables in appendix A showed that an optimal convolutional code (in terms of free distance) does not necessarily offer the best complexity/performance ratio. Thus, in an engineering point of view, we would like to know what kind of codes offers the best complexity/performance ratio, provided that the performance is above a given threshold.

- **Open Problem 5:** On the other hand, the matrix $\hat{G}$ of the trellis–canonical PGM falls in the category of "low-complexity templates". One interesting question is what is the best convolutional code (in terms of the free distance) that a given template can support.

- **Open Problem 6:** More accurate bounds on the LTC/ACG ratio would also be welcome.

The trellis decoding complexity.

- **Open Problem 7:** The implementation of Viterbi-like algorithms is quite a different problem. It deals with the trellis sectionalization, the metric computing scheme and the VLSI design, *etc.* Thus, calculating the decoding complexity (the total number of addition equivalent operations) is a very important but complex engineering problem.

- **Open Problem 8:** We proved lower bounds on the metric computation complexity in chapter 5. But more work is needed to do to improve the bounds and design optimal computing scheme.

- **Open Problem 9:** The birth of Turbo codes and Turbo decoding algorithms has opened a new frontier for modern telecommunication research. Recent research showed that the Turbo decoding algorithm is related to the backward–forward propagation algorithm (in learning theory) and belief propagation in Bayesian network (in artificial intelligent theory). Like the Viterbi algorithm, Turbo decoding algorithm can be classified as "decoding on graph". Under certain conditions, the Turbo decoding algorithm reduces to the Viterbi algorithm. We believe that the study of all these algorithms will open a new page in the history of communication and coding theory and engineering.

Appendix A Good Permutations for Block Codes

In this appendix we present tables listing some interesting binary block codes, together with our current best estimate of their permutation trellis complexities. In the table, the "Upper Bound" represents the fewest edges in any BCJR trellis for a permutation of the listed code that our algorithm has been able to find in limited time. The "Complexity" is the average trellis complexity over information space dimension, which is the number of edges in the best trellis found so far, divided by the number of information bits k in the code. This is a rough measure of the decoding complexity per decoded bit, if the Viterbi algorithm is used on the trellis. The "Lower Bound" is the bound from eqn. (2.49) (except for the $[7, 4, 3, 4]$ Hamming code). The codes parameters are, from left to right, *length, dimension, distance, and dual distance.*

Remarks:

- The $*$ means that the permutation found by the search algorithm is optimal.
- One optimal permutation of the $[24, 12, 8, 8]$ extended Golay code was discovered by Forney in [27].
- Kasami *et al.* in [37, 38] first showed that the standard permutation of Reed-Muller codes is optimum, which based on the results of [84]. These results include some codes in the table such as the $[8, 4, 4, 4]$ self-dual code, $[16, 5, 8, 4]$ BCH code, $[16, 11, 4, 8]$ BCH code, $[32, 6, 16, 4]$ BCH code, $[32, 26, 4, 16]$ BCH code, and $[32, 16, 8, 8]$ BCH code.
- The optimal permutation of the $[32, 21, 6, 12]$ BCH code corresponds to 14,972 edges in the optimal trellis [83].
- The $[17, 9, 5, 6]$ QR code is from [39].
- The generator matrix for the $[31, 10, 12, 5]$ JMG code is

$$\begin{pmatrix}
1000000000101010111100100101001 \\
0100000000111111000101101111011 \\
0010000000010100110111111011111 \\
0001000000101001101111110111110 \\
0000100000111001100011001010101 \\
0000010000011001111010110000011 \\
0000001000110011110101100000110 \\
0000000100001101010111100100101 \\
0000000010011010101111001001010 \\
0000000001110101011110010010100
\end{pmatrix}$$

Table A.1: Code table of good permutations: Part I.

Code	**Upper Bound**	**Complexity**	**Lower Bound**
[7,4,3,4]* *Hamming code*	36	9	36
[8,4,4,4]* *Self−dual code*	44	11	44
[15,5,7,4] *BCH code*	156	31.2	148
[15,7,5,4] *BCH code*	300	42.8	172
[15,11,3,8] *BCH code*	196	17.8	148
[16,5,8,4]* *BCH code*	172	34.4	172
[16,7,6,4] *BCH code*	420	60	236
[16,11,4,8]* *BCH code*	252	22.9	252
[17,9,5,6] *QR code*	412	45.8	380

Table A.2: Code table of good permutations: Part II.

Code	Upper Bound	Complexity	Lower Bound
[18,9,6,6] *QRcode*	732	81.33	508
[23,12,7,8] *Golay code*	3068	255.7	2684
[24,12,8,8]* *Golay code*	3580	298.3	3580
[31,10,12,5] *BCH code*	7068	706.8	3484
[31,10,12,5] *JMG code*	7500	750	3484
[31,16,7,8] *BCH code*	5884	367.75	4796
[32,6,16,4]* *BCH code*	684	114	684
[32,11,12,6] *BCH code*	11804	1073.09	5948
[32,16,8,8]* *BCH code*	6396	399.75	6396

Table A.3: Code table of good permutations: Part III.

Code	Upper Bound	Complexity	Lower Bound
[32,21,6,12] *BCH code*	17340	825.7	8316
[32,26,4,16]* *BCH code*	1180	45.4	1180
[47,24,11,12] *QR code*	794620	33109.2	645116
[48,24,12,12]* *Self−dual code*	860156	35839.8	860156
[64,39,10,8] *BCH code*	95839228	$2.46\ 10^6$	161020
[64,45,8,16] *BCH code*	1739004	38644.5	143868
[64,51,6,24] *BCH code*	231228	4533.88	82556
[64,57,4,32] *BCH code*	5676	99.6	5084
[80,40,16,16] *Self−dual code*	1937899516	$4.845\ 10^7$	146931708

Appendix B Tables of LTC versus ACG

In this appendix, we list the ACG and the LTC for a large number of good convolutional codes, and a few block codes. The Partial-Unit-Memory codes are from [1]. Other convolutional codes are from various publications but all could be found in [21].

From left to right, The parameters of PUM codes are read as *length, dimension, degree, free distance.* For all other convolutional codes they are read as *length, dimension, memory, free distance.* For block codes, the parameters are *length, dimension, distance.*

Table B.1: Selected best (2,1,m) codes. (From pp. 85–88 in [21])

Code	**LTC**	**ACG**	**LTC-ACG Ratio**
$(2, 1, 2, 5)$	4	2.5	1.60
$(2, 1, 3, 6)$	5	3	1.67
$(2, 1, 4, 7)$	6	3.5	1.71
$(2, 1, 5, 8)$	7	4	1.75
$(2, 1, 6, 10)$	8	5	1.60
$(2, 1, 8, 12)$	10	6	1.67
$(2, 1, 10, 14)$	12	7	1.71
$(2, 1, 11, 15)$	13	7.5	1.73
$(2, 1, 12, 16)$	14	8	1.75

Table B.2: Selected best (2,1,m) codes, continued. (From pp. 85–88 in [21])

Code	**LTC**	**ACG**	**LTC–ACG Ratio**
$(2,1,14,18)$	16	9	1.78
$(2,1,15,19)$	17	9.5	1.79
$(2,1,16,20)$	18	10	1.80
$(2,1,18,22)$	20	11	1.82
$(2,1,21,24)$	23	12	1.92
$(2,1,23,26)$	25	13	1.92
$(2,1,25,27)$	27	13.5	2.00
$(2,1,27,28)$	29	14	2.07
$(2,1,30,30)$	32	15	2.13

Table B.3: Selected best (3,2,m) codes. (From pp. 90 in [21])

Code	**LTC**	**ACG**	**LTC-ACG Ratio**
$(3, 2, 2, 3)$	3.58	2.00	1.79
$(3, 2, 3, 4)$	5.00	2.67	1.87
$(3, 2, 4, 5)$	6.00	3.33	1.80
$(3, 2, 5, 6)$	7.00	4.00	1.75
$(3, 2, 6, 7)$	8.00	4.67	1.71
$(3, 2, 7, 8)$	9.00	5.33	1.69
$(3, 2, 8, 8)$	10.00	5.33	1.88
$(3, 2, 9, 9)$	11.00	6.00	1.83
$(3, 2, 10, 10)$	12.00	6.67	1.80

Table B.4: Selected best (4,3,m) codes. (From pp. 90 in [21])

Code	**LTC**	**ACG**	**LTC-ACG Ratio**
$(4, 3, 3, 4)$	5.00	3.00	1.67
$(4, 3, 5, 5)$	7.00	3.75	1.87
$(4, 3, 6, 6)$	8.00	4.50	1.78
$(4, 3, 8, 7)$	10.00	5.25	1.90
$(4, 3, 9, 8)$	11.00	6.00	1.83

Table B.5: Selected best (3,1,m) codes. (From pp. 89 in [21])

Code	**LTC**	**ACG**	**LTC-ACG Ratio**
$(3, 1, 2, 8)$	4.58	2.67	1.72
$(3, 1, 3, 10)$	5.58	3.33	1.68
$(3, 1, 4, 12)$	6.58	4.00	1.64
$(3, 1, 5, 13)$	7.58	4.33	1.75
$(3, 1, 6, 15)$	8.58	5.00	1.72
$(3, 1, 7, 16)$	9.58	5.33	1.80
$(3, 1, 8, 18)$	10.58	6.00	1.76
$(3, 1, 9, 20)$	11.58	6.67	1.74
$(3, 1, 10, 22)$	12.58	7.33	1.72
$(3, 1, 11, 24)$	13.58	8.00	1.70
$(3, 1, 12, 24)$	14.58	8.00	1.82
$(3, 1, 13, 26)$	15.58	8.67	1.80

Table B.6: Selected best (4,1,m) codes. (From pp. 89 in [21])

Code	**LTC**	**ACG**	**LTC-ACG Ratio**
$(4,1,2,10)$	5.00	2.50	2.00
$(4,1,3,13)$	6.00	3.25	1.85
$(4,1,4,16)$	7.00	4.00	1.75
$(4,1,5,18)$	8.00	4.50	1.78
$(4,1,6,20)$	9.00	5.00	1.80
$(4,1,7,22)$	10.00	5.50	1.82
$(4,1,8,24)$	11.00	6.00	1.83
$(4,1,9,27)$	12.00	6.75	1.78
$(4,1,10,29)$	13.00	7.25	1.79
$(4,1,11,32)$	14.00	8.00	1.75
$(4,1,12,33)$	15.00	8.25	1.82
$(4,1,13,36)$	16.00	9.00	1.78

Table B.7: Selected block codes and PUM codes.

Code	LTC	ACG	LTC-ACG Ratio
[8,4,4] *Self−dual Code*	3.46	2.00	1.73
[24,12,8] *Golay Code*	8.22	4.00	2.06
[32,16,8] *BCH Code*	8.64	4.00	2.16
[48,24,12] *Self−dual Code*	15.13	6.00	2.52
[n,$n-1$,2] *Parity−check Code*	2.00	$\frac{2(n-1)}{n}$	$\frac{n}{n-1}$
[n,1,n] *Repetition Code*	$1 + \log_2 n$	1	$1 + \log_2 n$
(8,4,3,8) *PUM Code*	6.70	4.00	1.68
(24,12,7,12) *PUM Code*	15.58	6.00	2.60
(24,12,10,16) *PUM Code*	18.58	8.00	2.32

Bibliography

[1] K. Abdel–Ghaffar, R. J. McEliece, and G. Solomon, "Some Partial–Unit–Memory Convolutional Codes," *JPL TDA Progress Report,* vol. 42–107, (November 1991), pp. 57-72. Also see *Proc. 1991 International Symposium on Information Theory,* p. 196.

[2] A. Albanese, J. Blömer, J. Edmonds, M. Luby, and M. Sudan, "Priority Encoding Transmission," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1737–1744, November 1996.

[3] N. Alon, and M. Luby, "A Linear Time Erasure–Resilient Code with Nearly Optimal Recovery," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1732–1736, November 1996.

[4] J. J. Ashley, "A Linear Bound for Sliding–Block Decoder Window Size, II," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1913–1924, November 1996.

[5] J. J. Ashley, R. Karabed, and P. H. Siegel, "Complexity and Sliding–Block Decodability," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1925–1947, November 1996.

[6] J. J. Ashley, B. H. Marcus, and R. M. Roth, "On the Decoding Delay of Encoders for Input–Constrained Channels," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1948–1956, November 1996.

[7] L. R. Bahl, J.Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inform. Theory,* vol. IT–20, pp. 284–287, March 1974.

[8] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan, "Linearity Testing in Characteristic Two," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1781–1795, November 1996.

[9] Y. Berger, and Y. Be'ery, "The Twisted Squaring Construction, Trellis Complexity, and Generalized Weights of BCH and QR Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1817–1827, November 1996.

[10] G. Bégin, and D. Haccoun, "High–Rate Punctured Convolutional Codes: Structure Properties and Construction Technique," *IEEE Trans. Comm.,* vol. COM–37, pp. 1381–1385, December 1989.

[11] E. R. Berlekamp, *Algebraic Coding Theory.* New York: McGraw-Hill, 1968.

[12] I. Bocharova, and B. Kudryashov, "Nonsyndrome Maximum Likelihood Decoding of Linear Codes Using a Trellis," *Proc. 4th International Workshop on Algebraic and Combinatorial Coding Theory,* (September 1994, Novogrod, Russia), pp. 35–39.

[13] G. Brassard, C. Crépeau, and M. Sántha, "Oblivious Transfers and Intersecting Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1769–1780, November 1996.

[14] A. E. Brouwer, and T. Verhoeff, "An Updated Table of Minimum–distance Bounds for Binary Linear Codes," *IEEE Trans. Inform. Theory,* vol. IT–39, pp. 662–677, March 1993.

[15] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Likelihood Decoding," *IEEE Trans. Inform. Theory,* vol. IT–25, pp. 97–100, January 1979.

[16] G. C. Clark, Jr., and J. B. Cain, *Error-Correction Coding for Digital Communications.* New York: Plenum Press, 1981.

[17] G. D. Cohen, S. Litsyn, and G. Zémor, "On Greedy Algorithms in Coding Theory," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 2053–2057, November 1996.

[18] D. J. Costello, Jr., "Free Distance Bounds for Convolutional Codes," *IEEE Trans. Inform. Theory,* vol. IT–20, pp. 356–365, May 1974.

[19] D. G. Daut, J. W. Modestino, and L. D. Wismer, "New Short Constraint Length Convolutional Code Constructions for Selected Rational Rates," *IEEE Trans. Inform. Theory,* vol. IT–28, pp. 794–800, September 1982.

[20] U. Dettmar, and U. Sorger, "On Maximum Likelihood Decoding of Unit Memory Codes," *Proc. 6th Swedish–Russian International Workshop on Information Theory,* (August 1993), pp. 184–188.

[21] A. Dholakia, *Introduction to Convolutional Codes with Applications.* Boston: Kluwer Academic Publishers, 1994.

[22] S. Dolinar, L. Ekroot, A. B. Kiely, R. J. McEliece, and W. Lin, "The Permutation Trellis Complexity of Linear Block Codes," *Proc. 32nd Annual Allerton Conf. on Communication, Control, and Computing,* (September 1994), pp. 60–74.

[23] I. Dumer, "Suboptimal Decoding of Linear Codes: Partition Technique," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1971–1986, November 1996.

[24] F. Fagnani, and S. Zampieri, "Dynamical Systems and Convolutional Codes Over Finite Abelian Groups," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1892–1912, November 1996.

[25] G. D. Forney, Jr., "The Viterbi Algorithm," *Proc. IEEE,* Vol 61, pp. 268–276, March, 1973.

[26] G. D. Forney, Jr., "Convolutional Codes I: Algebraic Structure," *IEEE Trans. Inform. Theory,* vol. IT–16, pp. 268–278, November 1970.

[27] G. D. Forney, Jr., "Coset Codes — Part II: Binary Lattices and Related Codes," *IEEE Trans. Inform. Theory,* vol. IT–34, pp. 1152–1187, September 1988.

[28] G. D. Forney, Jr., "Dimension/Length Profiles and Trellis Complexity of Linear Block Codes," *IEEE Trans. Inform. Theory,* vol. IT–40, pp. 1741–1752, November 1994.

[29] G. D. Forney, Jr., and M. D. Trott, "The Dynamics of Group Codes: State Spaces, Trellis Diagrams, and Canonical Encoders," *IEEE Trans. Inform. Theory,* vol. IT–39, pp. 1491–1513, September 1993.

[30] G. D. Forney, Jr., and A. Vardy, "Generalized Minimum–Distance Decoding of Euclidean–Space Codes and Lattices," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1992–2026, November 1996.

[31] G. D. Forney, Jr., R. Johannesson, and Z.-X. Wan, "Minimal and Canonical Rational Generator Matrices for Convolutional Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1865–1880, November 1996.

[32] D. Haccoun, and G. Bégin, "High–Rate Punctured Convolutional Codes for Viterbi and Sequential Decoding," *IEEE Trans. Comm.,* vol. COM–37, pp. 1113–1125, November 1989.

[33] M. F. Hole, and Ø. Ytrehus, "Two-Step Trellis Decoding of Partial Unit Memory Convolutional Codes," *IEEE Trans. Inform. Theory,* vol. IT–43, pp. 324–330, January 1997.

[34] H. D. L. Hollmann, "Bounded–Delay–Encodable, Block–Decodable Codes for Constrained Systems," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1957–1970, November 1996.

[35] B. Honary, G. Markarian, and M. Darnell, "Trellis Decoding for Block Codes," *Proc. 3rd IEE Int. Symp. on Comm. Theory Appl.,* (July 1993, Ambleside, U.K.), pp. 79–93.

[36] G. B. Horn, and F. R. Kschischang, "On the Intractability of Permuting a Block Codes to Minimize Trellis Complexity," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory*, vol. IT–42, pp. 2042–2048, November 1996.

[37] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On the Optimum Bit Orders with Respect to the State Complexity of Trellis Diagrams for Binary Linear Codes," *IEEE Trans. Inform. Theory,* vol. IT–39, pp. 242–245, January 1993.

[38] T. Kasami, T. Takata, T. Fujiwara, and S. Lin, "On Complexity of Trellis Structure of Linear Block codes," *IEEE Trans. Inform. Theory,* vol. IT–39, pp. 1057–1064, May 1993.

[39] A. D. Kot, and C. Leung, "On the Construction and Dimensionality of Linear Block Code Trellises," *Proc. 1993 ISIT,* p. 291.

[40] A. B. Kiely, S. Dolinar, R. J. McEliece, L. Ekroot, and W. Lin, "Trellis Decoding Complexity of Linear Block Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory*, vol. IT–42, pp. 1687–1697, November 1996.

[41] F. R. Kschischang, "The Trellis Structure of Maximal Fixed-Cost Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1828–1838, November 1996.

[42] F. R. Kschischang, and V. Sorokine, "On the Trellis Structure of Block Code," *IEEE Trans. Inform. Theory,* vol. IT–41, pp. 1924–1937, November 1995.

[43] A. Lafourcade, and A. Vardy, "Lower Bounds on Trellis Complexity of Block Codes," *IEEE Trans. Inform. Theory,* vol. IT–41, pp. 1938–1954, November 1995.

[44] A. Lafourcade, and A. Vardy, "Asymptotically Good Codes Have Infinite Trellis Complexity," *IEEE Trans. Inform. Theory,* vol. IT–41, pp. 555–559, March 1995.

[45] A. Lafourcade, and A. Vardy, "Optimal Sectionalization of a Trellis," *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 689–703, May 1996.

[46] K. Larsen, "Short Convolutional Codes with Maximal Free Distance for Rate 1/2, 1/3, and 1/4," *IEEE Trans. Inform. Theory,* vol. IT–19, pp. 371–372, May 1973.

[47] G. S. Lauer, "Some Optimal Partial–Unit–Memory Codes," *IEEE Trans. Inform. Theory,* vol. IT–25, pp. 540–547, March 1979.

[48] W. Lin, and R. J. McEliece, "Asymptotic Coding Gain and Trellis Complexity," *Proc. 33rd Allerton Conference on Communication, Control, and Computing,* (October 1995), pp. 313–322.

[49] W. Lin, and R. J. McEliece, "Hybrid Trellis Decoding for Some Convolutional Codes," *Proc. 1996 Conference on the Information Science and Systems, Princeton,* (March 1996), pp. 572–577.

[50] W. Lin, and R. J. McEliece, "Trellis-Canonical Generator Matrices for Convolutional Codes," accepted, *ISIT 1997.*

[51] W. Lin, A. Kiely, and R. J. McEliece, "The Evolution of the Trellis for Block Code," In preparation.

[52] H.-A. Loeliger, and T. Mittelholzer, "Convolutional Codes Over Groups," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1660–1686, November 1996.

[53] K. Lumbard, and R. J. McEliece, "Counting Minimal Generator Matrices," *Proc. 1994 IEEE Inter. Symp. Inform. Theory,* (June 1994, Trondheim, Norway), p. 18.

[54] F. J. MacWilliams, and N.J.A. Sloane, *The Theory of Error-Correcting Codes.* Amsterdam: North-Holland, 1977.

[55] J. L. Massey, "Foundations and Methods of Channel Coding," *Proc. Inter. Conf. on Inform. Theory and Systems, NTG–Fachberichte,* vol. 65, pp. 148–157, 1978.

[56] R. J. McEliece, *The Theory of Information and Coding.* Reading, Mass.: Addison–Wesley, 1977.

[57] R. J. McEliece, "The Viterbi Decoding Complexity of Linear Block Codes," *Proc. 1994 ISIT,* (June 1994), p. 341.

[58] R. J. McEliece, "On the BCJR Trellis for Linear Block Codes," *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1072–1092, July 1996.

[59] R. J. McEliece, "The Algebraic Theory of Convolutional Codes," chapter in the *Handbook of Coding Theory,* in preparation.

[60] R. J. McEliece, and W. Lin, "The Trellis Complexity of Convolutional Codes," *Proc. 1995 ISIT,* (September, 1995), p. 131.

[61] R. J. McEliece, and W. Lin, "The Trellis Complexity of Convolutional Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1855-1864, November 1996.

[62] R. J. McEliece, and R. P. Stanley, "The General Theory of Convolutional Codes," *JPL TDA Progress Report,* vol. 42–113 (May 1993), pp. 89–98.

[63] R. J. McEliece, E. R. Rodemich, H. Rumsey, Jr., and L. R. Welch, " New Upper Bounds on the Rate of a Code via the Delsarte–MacWilliams Inequalities," *IEEE Trans. Inform. Theory,* vol. IT–23, pp. 157–166, March 1977.

[64] D. J. Muder, "Minimal Trellises for Block Codes," *IEEE Trans. Inform. Theory,* vol. IT–34, pp. 1049–1053, September 1988.

[65] E. Passke, "Short Binary Convolutional Codes with Maximal Free Distance for Rate 2/3 and 3/4," *IEEE Trans. Inform. Theory,* vol. IT–20, pp. 683–688, September 1974.

[66] L. C. Perez, J. Seghers, and D. J. Costello, Jr., "A Distance Spectrum Interpretation of Turbo Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1698–1709, November 1996.

[67] J. Rosenthal, J. M. Schumacher, and E. V. York, "On Behaviors and Convolutional Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1881–1891, November 1996.

[68] L. J. Schulman, "Coding for Interactive Communication," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1745–1756, November 1996.

[69] P. Schuurman, "A Table of State Complexity Bounds for Binary Linear Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 2034–2042, November 1996.

[70] B.-Z. Shen, K. K. Tzeng, and C. Wang, "A Bounded–Distance Decoding Algorithm for Binary Linear Block Codes Achieving the Minimum Effective Error Coefficient," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1987–1991, November 1996.

[71] V. Sidorenko, G. Markarian, and B. Honary, "Code Trellis and the Shannon Product," *Proc. 7th Joint Swedish–Russian International Workshop on Information Theory,* (June 1994), pp. 220–224.

[72] V. Sidorenko, G. Markarian, and B. Honary, "Minimal Trellis Design for Linear Codes Based on the Shannon Product," the Special Issue on Codes and Com-

plexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 2048–2053, November 1996.

[73] V. Sidorenko, and V. Zyablov, "Decoding of Convolutional Codes Using a Syndrome Trellis," *IEEE Trans. Inform. Theory,* vol. IT–40, pp. 1663-1666, September 1994.

[74] M. Sipser, and D. A. Spielman, "Expander Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1710–1722, November 1996.

[75] G. Solomon, and H. C. A. van Tilborg, "A Connection Between Block and Convolutional Codes," *SIAM J. of Appl. Math,* vol. 37, no.2, Oct. 1979.

[76] D. A. Spielman, "Linear–Time Encodable and Decodable Error–Correcting Codes," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1723–1731, November 1996.

[77] J. Stern, "A New Paradigm for Public Key Identification," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1757–1768, November 1996.

[78] V. Tarokh, and I. F. Blake, "Trellis Complexity Versus the Coding Gain of Lattices I," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1796–1807, November 1996.

[79] V. Tarokh, and I. F. Blake, "Trellis Complexity Versus the Coding Gain of Lattices II," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1808–1816, November 1996.

[80] A. Vardy, and F. R. Kschischang, "Proof of a Conjecture of McEliece Regarding the Expansion Index of the Minimal Trellis," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 2027–2034, November 1996.

[81] V. V. Vazirani, H. Saran, and B. Sundar Rajan, "An Efficient Algorithm for Constructing Minimal Trellises for Codes over Finite Abelian Groups," the Special Issue on Codes and Complexity of *IEEE Trans. Inform. Theory,* vol. IT–42, pp. 1839–1854, November 1996.

[82] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inform. Theory,* vol. IT–13, pp. 260-269, April 1967.

[83] Xiao-an Wang, and Stephen B. Wicker, "The Design and Implementation of Trellis Decoders for Some BCH Codes," *submitted to IEEE Trans. Inform. Theory,* 1996.

[84] V. K. Wei, "Generalized Hamming Weights for Linear Codes," *IEEE Trans. Inform. Theory,* vol. IT–37, pp. 1412-1418, September 1991.

[85] J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes," *IEEE Trans. Inform. Theory,* vol. IT–24, pp. 76-80, January 1978.

[86] Øyvind Ytrehus, "Ascetic Convolutional Codes," *Proc. 33nd Annual Allerton Conf. on Communication, Control, and Computing,* pp. 382–390, (October 1995).

[87] V. Zyablov, and V. Sidorenko, "Soft Decision Maximum Likelihood Decoding of Partial Unit Memory Codes," *Problems of Information Transmission,* vol. 28, No. 1, pp. 18-22, January–March 1994.