Dynamic Programming Algorithms for Maximum Likelihood Decoding

by

Kevin Geoffrey Kochanek

A.B. in Physics, Cornell University, 1991
M.S. in Physics, University of Illinois at Urbana-Champaign, 1993
Sc.M. in Applied Mathematics, Brown University, 1995

Thesis

Submitted in partial fulfillment of the requirements for
the Degree of Doctor of Philosophy
in Division of Applied Mathematics at Brown University

May 1998

Abstract of "Dynamic Programming Algorithms for Maximum Likelihood Decoding," by Kevin Geoffrey Kochanek, Ph.D., Brown University, May 1998

The Viterbi algorithm is the traditional prototype dynamic programming algorithm for maximum likelihood decoding. Seen from the perspective of formal language theory, this algorithm recursively parses a trellis code's regular grammar. This thesis discusses generalized Viterbi algorithms for the maximum likelihood decoding of codes generated by context-free grammars and transmitted across either memoryless or Markov communications channels. Among the codes representable by context-free grammars are iterated squaring constructions—including the Reed–Muller codes. Two additional strategies are introduced for handling large Reed–Muller-like codes. First, by systematically discarding information bits, a code's grammatical and decoding complexities can be reduced to manageable levels without seriously reducing its information capacity. Second, a coarse-to-fine dynamic programming algorithm for the maximum likelihood decoding of Reed–Muller-like codes is presented; this algorithm almost uniformly outperforms the Viterbi algorithm.

This dissertation by Kevin Geoffrey Kochanek is accepted in its present form by
Division of Applied Mathematics as satisfying the
dissertation requirement for the degree of
Doctor of Philosophy

Date..... .....
Stuart Geman

Recommended to the Graduate Council

Date..... .....
David Mumford

Date..... .....
Donald McClure

Approved by the Graduate Council

Date..... .....

The Vita of Kevin Geoffrey Kochanek

Kevin Geoffrey Kochanek was born in State College, PA on March 31, 1970. He attended Georgetown Day High School in Washington, DC from 1983 to 1987. In 1991, he graduated from Cornell University *summa cum laude* in physics. After earning a masters in physics from the University of Illinois at Urbana-Champaign in 1993, he transferred into the applied mathematics program at Brown University. He received a masters in applied mathematics in 1995 and defended this Ph.D. thesis on October 16, 1997.

Preface

Coding theorists have long recognized dynamic programming as a powerful tool for performing exact maximum likelihood decoding. The most widely examined dynamic programming application, known as the Viterbi algorithm, decodes a given code by computing the shortest length path through its associated trellis diagram. Seen from the perspective of formal language theory, the Viterbi algorithm recursively parses a code's regular grammar. By further exploring the relationship between codes and formal grammars, this thesis aims to extend the applicability of dynamic programming techniques within coding theory.

Chapter 1 provides a brief introduction to the fundamental concepts from coding theory and formal language theory that underpin the remainder of the thesis. After discussing the structure of error correcting codes and the optimality of maximum likelihood decoding, we introduce the Viterbi algorithm and its grammatical interpretation. We also introduce the family of Reed–Muller codes, our canonical example of codes derivable from context-free grammars.

Chapter 2 presents generalized Viterbi algorithms for the class of codes derived from context-free grammars. Here we discuss maximum likelihood decoding algorithms for both memoryless and Markov communications channels, simultaneously introducing the posterior probability as a useful reliability statistic.

In chapter 3, we construct codes generated by context-free grammars to which these algorithms may be applied. Reinterpreting Forney's iterated squaring construction in grammatical terms, we develop a large class of such codes, including the widely known Reed–Muller codes. Moreover, we relate the computational complexity of our decoding algorithms to the corresponding grammatical complexity of the given codes. Since many of the larger Reed–Muller codes are effectively undecodable (even using dynamic programming methods), we construct a family of thinned Reed–Muller codes whose grammatical and decoding complexities are strictly controlled.

Chapter 4 presents a coarse-to-fine dynamic programming algorithm for the maximum likelihood decoding of thinned Reed–Muller codes. This coarse-to-fine procedure computes the maximum likelihood codeword by applying the standard dynamic programming approach to a sequence of codes that in some sense approximate the original code. Its implementation is highly dependent on the particular grammatical structure of these thinned

Reed–Muller codes.

Finally, Chapter 5 is a conclusion that combines an analysis of simulated decoding trials with a discussion of the important unifying themes of this thesis.

Acknowledgments

Although the process of researching and writing a Ph.D. thesis is largely an individual effort, the substance and character of the final product in fact reveal the contributions of the many people who made it possible. First and foremost among them are my parents who have always fostered my intellectual and creative development. I dedicate this thesis to you, though you will find it exceedingly difficult to read.

I have also benefited enormously from the informal yet rigorous atmosphere of the Division of Applied Mathematics. In particular, I would like to thank my adviser Stuart Geman who provided a near perfect combination of guidance and autonomy. My committee members Donald McClure, David Mumford, and Basilis Gidas were also an invaluable source of constructive criticism and advice. Finally, I am grateful to Wendell Fleming for introducing me to probability theory and tutoring me in stochastic processes.

In addition, there are a number of people from outside the Division who have influenced the course of my graduate studies. Chief among them is my undergraduate adviser Neil Ashcroft who has for many years been a role model and mentor. Paul Goldbart and Eduardo Fradkin from Illinois kindly aided my transfer from physics to applied mathematics. Sidney Winter generously supported a memorable summer of studying management theory at the Wharton School. Finally, I would also like to thank Ronel Elul and Robert Ashcroft for introducing me to financial economics.

Of course, without the generous financial support of the Division and Brown University, this thesis could not have been written. Finally, I am grateful to Jean Radican, Laura Leddy, Roselyn Winterbottom, and Trudee Trudell for cheerfully helping me manage a host of administrative details.

Contents

Chapter 1

Preliminaries

1.1 Error Correcting Codes

1.1.1 Introduction

A code is simply a mathematical structure designed to store information. More formally, a binary (N, M) *code* $\mathcal{C}$ is a set of M binary strings of length N. A given stream of source data (perhaps a satellite image) represented by a finite string of information bits is encoded as a sequence of codewords with each successive substring of length $\log_2 M$ selecting an associated codeword from $\mathcal{C}$ according to a known rule. As we shall see in Section 1.2, noisy communications channels will typically corrupt transmitted bit streams; thus, any receiver must use a decoding scheme to infer the transmitted codeword from a channel's potentially garbled output. To facilitate both efficient and reliable decoding, commonly used codes exhibit ample algebraic and geometric structure.

In order to understand the role of geometry in error correcting codes, we introduce the following norm and metric on the space $\mathbf{Z}_2^N$ of binary N-tuples. For strings $\mathbf{x}$ and $\mathbf{y}$, the (Hamming) *weight* $w(\mathbf{x})$ of $\mathbf{x}$ is the number of non-zero bits in $\mathbf{x}$ and the (Hamming) *distance* $d(\mathbf{x}, \mathbf{y})$ between $\mathbf{x}$ and $\mathbf{y}$ is the weight of the string $\mathbf{x} - \mathbf{y}$. If S and T are sets of strings, the *subset distance* $d(S, T)$ is the minimum distance between two elements of S and T. Finally, the *minimum distance* $d(S)$ of a set S is the minimum distance between two distinct elements of S (which can in general differ from the *minimum weight* $w(S)$ of S, the smallest non-zero weight of any string in S). An (N, M, d) code is an (N, M) code with minimum distance d.

Often the minimum distance of a code alone determines its capacity to detect and correct transmission errors [10]. Consider a communications channel (such as the binary symmetric channel introduced in Section 1.2) that can corrupt a transmitted codeword by flipping its component bits from 0 to 1 or vice-versa. Imagine sending a codeword $\mathbf{c}$ from an (N, M, d) code $\mathcal{C}$ across this channel. If the channel flips between 1 and $d-1$ bits of $\mathbf{c}$, the received word will not be a codeword—the codeword closest to $\mathbf{c}$ differs in at least d bits. In other words, $\mathcal{C}$ *detects* up to $d-1$ errors. Now suppose we adopt the following intuitive decoding scheme (seen in Section 1.2 as optimal in some circumstances): upon receiving the string $\mathbf{d}$ (not necessarily a codeword), infer that the transmitted codeword is that element of $\mathcal{C}$ which is closest to $\mathbf{d}$ (in Hamming distance). If the channel flips between 1 and $\lfloor (d-1)/2 \rfloor$ bits, this decoding scheme correctly yields the sent codeword $\mathbf{c}$. Thus, $\mathcal{C}$ also *corrects* up to

$\lfloor (d-1)/2 \rfloor$ errors.

Among the most widespread algebraic structures occurring in coding theory are groups [10]. A *linear* binary $[N, K, d]$ code $\mathcal{C}$ is a set of 2^K binary N-tuples with minimum distance d (i.e. an $(N, 2^K, d)$ code) that forms a group under mod-2 vector addition. Being a K-dimensional subspace of the linear space $\mathbf{Z}_2^N$, $\mathcal{C}$ is uniquely characterized by a K element basis of *generators* $G_C = \{\mathbf{g}_k | 1 \leq k \leq K\}$ in the sense that any element of $\mathcal{C}$ can be expressed as a linear combination of these generators. The *generator matrix* G_C may be equivalently viewed as either a set of K generators or as a $K \times N$ matrix with the generators ordered by row. Symbolically,

$$\mathcal{C} \ = L(G_C) \stackrel{\Delta}{=} \{\sum_{k=1}^{K} a_k \mathbf{g}_k | \mathbf{a} \ \epsilon \ \mathbf{Z}_2^K\} = \{\mathbf{a} G_C | \mathbf{a} \ \epsilon \ \mathbf{Z}_2^K\}.$$

Generator matrices inherit an algebra from their associated linear codes [4]. The direct sum $C_1 + C_2$ of two codes C_1 and C_2, generated respectively by G_1 and G_2, has the generator matrix $G_1 + G_2 \stackrel{\Delta}{=} G_1 \bigcup G_2$. In this algebra, subtraction is not defined and the zero generator matrix is the empty set.

Codes that also happen to be groups exhibit a rich structure. If $\mathcal{C}'$ is a subgroup of $\mathcal{C}$, the distinct cosets of $\mathcal{C}'$ partition the group $\mathcal{C}$ itself:

$$\mathcal{C} \ = \bigcup_{\mathbf{c} \ \epsilon \ [\mathcal{C}/\mathcal{C}']} \mathcal{C}' + \mathbf{c} \ = \mathcal{C}' + [\mathcal{C}/\mathcal{C}'],$$

where $[\mathcal{C}/\mathcal{C}']$ is a complete set of coset representatives and $+$ denotes a direct sum. If $\mathcal{C}'$ is generated by $G_{C'} \subset G_C$ and $[\mathcal{C}/\mathcal{C}']$ is chosen to be generated by $G_{C/C'} = G_C \backslash G_{C'}$, then the corresponding partition of $\mathcal{C}$'s generator matrix is

$$G_C = G_{C'} + G_{C/C'}.$$

Furthermore, since $\mathcal{C}'$ is itself linear, the minimum distance of each coset $\mathcal{C}' + \mathbf{c}$ in this partition equals $d(\mathcal{C}')$. In general, if $\mathcal{C}_1 > \mathcal{C}_2 > \cdots > \mathcal{C}_m$ is a nested sequence of subgroups of $\mathcal{C}$ having minimum distances $d(\mathcal{C}) \leq d(\mathcal{C}_1) \leq \cdots \leq d(\mathcal{C}_m)$, then $\mathcal{C}$ can be successively partitioned into cosets—all sharing the same minimum distance at any given level of refinement.

We now introduce two examples of error correcting codes that feature prominently in our discussion of maximum likelihood decoding.

1.1.2 Reed–Muller Codes

The family of Reed–Muller codes will emerge in Chapter 3 as the canonical example of codes derivable from context-free grammars. The following construction was introduced by Forney [4].

Consider the following alternative basis for $\mathbf{Z}_2^2$: $G_{(2,2)} \triangleq \{\mathbf{g}_0, \mathbf{g}_1\}$ where $\mathbf{g}_0 \triangleq [1,0]$ and $\mathbf{g}_1 \triangleq [1,1]$. Recall that $G_{(2,2)}$ can be considered to be a 2×2 generator matrix for the code $\mathbf{Z}_2^2$ with rows $\mathbf{g}_0$ and $\mathbf{g}_1$. Now define the $N \times N$ ($N = 2^n$) matrix $G_{(N,N)} = G_{(2,2)}^n \triangleq \otimes_{i=1}^n G_{(2,2)}$, the n-fold Kronecker product of $G_{(2,2)}$ with itself. (If A and B are binary matrices of dimensions $m \times n$ and $p \times q$ respectively, then $A \otimes B$ is the $mp \times nq$ matrix obtained by replacing every element a_{ij} in A with the matrix $a_{ij}B$.) As in the case $N = 2$, $G_{(N,N)}$ is both a basis and generator matrix for $\mathbf{Z}_2^N$; its rows are simply all 2^n n-fold Kronecker products of $\mathbf{g}_0$ and $\mathbf{g}_1$. Since $w(\mathbf{g}_1 \otimes \mathbf{x}) = 2w(\mathbf{x})$ and $w(\mathbf{g}_0 \otimes \mathbf{x}) = w(\mathbf{x})$ for any vector $\mathbf{x}$, the weight of such a row is 2^{n-r} where r $(n-r)$ is the number of $\mathbf{g}_0$'s ($\mathbf{g}_1$'s) in the corresponding Kronecker product.

The generator matrices defining the Reed–Muller codes are constructed by selecting specific subsets of rows from $G_{(N,N)}$. Let $G_{\partial RM}(r,n)$ be the set of $\binom{n}{r}$ rows of weight 2^{n-r} (i.e. all n-fold Kronecker products of r $\mathbf{g}_0$'s and $(n-r)$ $\mathbf{g}_1$'s) drawn from $G_{(N,N)}$. Moreover, let $G_{RM}(r,n) = \sum_{s=0}^{r} G_{\partial RM}(s,n)$ be the set of all rows from $G_{(N,N)}$ of weight 2^{n-r} or greater. The *Reed–Muller code* RM(r,n) is defined to be the binary block code (of length $N = 2^n$, dimension $K = \sum_{s=0}^{r} \binom{n}{s}$, and (as shown in section 3.2) minimum distance $d = 2^{n-r}$) generated by the matrix $G_{RM}(r,n)$:

$$RM(r,n) = L(G_{RM}(r,n)).$$

Furthermore, $G_{\partial RM}(r,n)$ can be chosen as the generator matrix for the group of coset representatives $[RM(r,n)/RM(r-1,n)]$:

$$[RM(r,n)/RM(r-1,n)] = L(G_{\partial RM}(r,n)).$$

For completeness, define $RM(-1,n) \triangleq \{\mathbf{0}\}$.

1.1.3 Convolutional Codes

Convolutional codes provide the second example of a grammatically derived family of codes that is amenable to decoding by dynamic programming.

A convolutional code is most easily defined as the output of a deterministic finite automaton [3]. At discrete time intervals, a rate k/n binary convolutional encoder accepts a k-bit input sequence and generates an n-bit ($n \geq k$) output sequence depending only on the current state of the machine and the input string. Upon accepting an input string, the automaton makes a transition from it's current state to one of $M = 2^k$ input-determined successor states. Typically, a convolutional code is terminated at time $T - \nu$ by requiring the machine to accept a fixed input sequence for ν time units, thus forcing its state to a given end-state at time T. The resulting set of output strings, an $(nT, M^{T-\nu})$ code, is a *rate k/n convolutional code*.

The output of such a convolutional encoder—a convolutional codeword— can be represented as a path through a branching graph [3]. For a code of length nT, the graph has n sections of branches connecting $n + 1$ time-indexed sets of nodes ordered from left to right (0 to n). A node at time t is labeled according to the corresponding state of the machine, whereas a branch between two nodes at successive times is labeled by the n-bit output sequence associated with the allowed transition. At the extreme left lies the sole start-node (representing the encoder in its start-state at time 0) from which M branches emerge, each corresponding to a possible input string of k-bits. These M branches terminate in M time 1 nodes which in turn branch M times, and so on. Thus, at time $t \leq T - \nu$ there are M^t nodes in one-to-one correspondence with the set of all possible inputs to date; for $T - \nu \leq t \leq T$, the number of nodes remains constant at $M^{T-\nu}$ and the graph branches singly—reflecting the uniform terminal input stream.

The standard graphical representation of a convolutional code, called a *trellis diagram*, eliminates the inherent redundancy of the above branching graph [3]. Since the number of nodes grows exponentially with time, at some point the number of nodes exceeds the cardinality S of the automaton's state space; thus, many nodes, while representing different sequences of inputs, correspond to identical states of the encoder. Since the action of the machine depends only on its current state (and a sequence of future inputs), the subtrees emerging from different nodes in identical states may be merged. The graph resulting from

this consolidation, a trellis diagram, has at most $2 + (n-1)S$ interconnected nodes and terminates at a single end-node at time n.

1.2 Maximum Likelihood Decoding

Having discussed various static features of error correcting codes, consider now the problem of transmission. Physically, a *communications channel* is a device that accepts codewords from a code $\mathcal{C}$ and emits potentially garbled words belonging to an output set $\mathcal{D}$. If $\mathcal{C}$ is an (N, M) code, $\mathcal{D}$ will typically be a subset of binary or real-valued N-tuples. One can therefore model the channel as a conditional probability distribution on the output set with the *channel probability* $p(\mathbf{d}|\mathbf{c})$ denoting the conditional probability that $\mathbf{d} \epsilon \mathcal{D}$ is received, given that $\mathbf{c} \epsilon \mathcal{C}$ is sent. If $\mathcal{D}$ is not a discrete space, assume that the channel probability distribution admits a conditional density of the form $p(\mathbf{d}|\mathbf{c})$. The conclusions of this section do not depend on this distinction.

A channel is termed *memoryless* if it independently corrupts the bits of a transmitted codeword:

$$p(\mathbf{d}|\mathbf{c}) = \prod_{i=1}^{N} p(d_i|c_i)$$

The coding theory literature features a number of memoryless channels of which the most common are:

1. The binary symmetric channel. $\mathcal{D} = \mathbf{Z}_2^N$: $p(1|0) = p(0|1) = p$.
2. The binary asymmetric channel. $\mathcal{D} = \mathbf{Z}_2^N$: $p(1|0) = p, p(0|1) = q$.
3. The additive gaussian noise channel. $\mathcal{D} = \mathbf{R}^N$: $d_i = c_i + n_i$, $n_i \sim N(0, \sigma^2)$.
4. The bipolar channel with additive gaussian noise. $\mathcal{D} = \mathbf{R}^N$: $d_i = 2c_i - 1 + n_i$, $n_i \sim N(0, \sigma^2)$.

Given any output $\mathbf{d} \epsilon \mathcal{D}$ of a noisy communications channel, a *decoding scheme* $f : \mathcal{D} \to \mathcal{C}$ is a function that infers the corresponding transmitted codeword $f(\mathbf{d})$. If $f(\mathbf{d})$ is indeed the transmitted codeword, the scheme has *corrected* any transmission errors; otherwise, it has made a *decoding error*. An *optimal decoding scheme* minimizes the probability of making decoding errors, thereby maximizing the probability of correcting errors.

Following the standard Bayesian approach [10], introduce a prior probability distribution $p(\mathbf{c})$ on the channel's input—the codebook $\mathcal{C}$. By Bayes rule, the posterior probability that $\mathbf{c} \in \mathcal{C}$ was in fact sent is

$$p(\mathbf{c}|\mathbf{d}) = \frac{p(\mathbf{d},\mathbf{c})}{p(\mathbf{d})} = \frac{p(\mathbf{d}|\mathbf{c})p(\mathbf{c})}{\sum_{\mathbf{c} \in \mathcal{C}} p(\mathbf{d}|\mathbf{c})p(\mathbf{c})}.$$

$P(\mathbf{c}|\mathbf{d})$ is also called the *backwards channel probability*. The optimal decoding scheme that maximizes the probability of correcting errors is clearly

$$\hat{\mathbf{c}}(\mathbf{d}) = \arg\max_{\mathbf{c} \in \mathcal{C}} p(\mathbf{c}|\mathbf{d}).$$

Having no particular knowledge concerning the channel inputs, impose the uniform prior $p(\mathbf{c}) = 1/M$ on the codebook $\mathcal{C}$. The posterior probability becomes

$$p(\mathbf{c}|\mathbf{d}) = \frac{p(\mathbf{d}|\mathbf{c})}{\sum_{\mathbf{c} \in \mathcal{C}} p(\mathbf{d}|\mathbf{c})}, \tag{1.2.1}$$

simplifying the decoding scheme accordingly:

$$\hat{\mathbf{c}}(\mathbf{d}) = \arg\max_{\mathbf{c} \in \mathcal{C}} p(\mathbf{d}|\mathbf{c}).$$

This procedure is called *maximum likelihood decoding* (or soft decoding), for the codeword $\hat{\mathbf{c}}(\mathbf{d})$ is the channel input that makes the output $\mathbf{d}$ most likely.

To illustrate this decoding approach, consider a binary symmetric channel with bit-error probability $p \leq 1/2$. If the codeword $\mathbf{c}$ is transmitted across this channel, producing the output word $\mathbf{d}$, the number of bit-errors incurred during transmission equals the distance $d(\mathbf{c},\mathbf{d})$. The resulting likelihood function,

$$p(\mathbf{d}|\mathbf{c}) = p^{d(\mathbf{c},\mathbf{d})}(1-p)^{N-d(\mathbf{c},\mathbf{d})},$$

is maximized when $d(\mathbf{c},\mathbf{d})$ is a minimum [10]; to decode the received word $\mathbf{d}$, one simply selects the nearest codeword $\hat{\mathbf{c}}(\mathbf{d})$. This common scheme is termed *minimum distance decoding* (or hard decoding). However, for most channels maximum likelihood and minimum distance decoding do not coincide.

For all but the smallest codes, sequentially searching a codebook for the optimal element

under a decoding scheme is impossible. However, for a variety of important codes and channels, maximum likelihood decoding can be formulated as a dynamic programming problem.

1.3 The Viterbi Algorithm

The Viterbi algorithm [11] was introduced in 1967 to expedite the decoding of convolutional codes. In fact, it can be applied to all trellis-based codes (including linear block codes [12]).

Consider the maximum likelihood decoding of a length $N = nT$ convolutional codeword transmitted through a memoryless channel. Taking the negative logarithm of the likelihood converts the decoding problem into the following minimization of a bit-wise additive cost function:

$$\hat{\mathbf{c}}(\mathbf{d}) = \arg \min_{\mathbf{c} \ \epsilon \ \mathcal{C}} \sum_{i=1}^{N} - \ln p(d_i|c_i).$$

Now introduce a metric on the branches of the code's trellis diagram [3]. If $\{d_i | n(t-1) < i \leq nt\}$ is the length n output of the channel in the time interval $(t-1, t]$ and $\{c_i | n(t-1) < i \leq nt\}$ is a possible channel input associated with a particular branch in the t-th section of the trellis diagram, assign that branch a "length" equal to

$$\sum_{i=n(t-1)+1}^{nt} - \ln p(d_i|c_i).$$

Maximum likelihood decoding is thereby reduced to finding the shortest "length" path through the code's trellis [2].

Viterbi suggested the following dynamic programming approach for finding this shortest path [3].

1. Set $t = 0$ and initialize the length of the start node to zero.

2. For each node (state) s at depth (time) $t+1$, consider the set of predecessors at depth t—nodes for which there is a connecting branch to s. For each predecessor, compute the sum of the predecessor's length and the length of the branch connecting it to s. Label the node s with a length equal to the smallest of these sums and by the bit sequence corresponding to the shortest length path to s—the shortest path label of the minimizing predecessor concatenated with the label of its connecting branch.

3. If $t < T$, increment t and goto 2; otherwise stop computing and return the shortest length path to the final node, the maximum likelihood estimate for the transmitted codeword.

The validity of this method—called the *Viterbi algorithm*—is evident once one realizes that the shortest path to any given node must be the extension of the shortest length path to one of its predecessor nodes.

The computational complexity of Viterbi's algorithm is easily established. For a trellis diagram with n sections and at most S nodes at each time step, $2nS^2$ is a loose upper bound on the number of real number operations involved. For each node at a given depth, computing the optimal extension requires p additions and $p-1$ comparisons, where p—the number of predecessor nodes—is necessarily less than or equal to S. Provided M^t exceeds S for most values of t, the Viterbi algorithm without question outperforms a sequential search of the size $M^{T-\nu}$ convolutional codebook.

1.4 Context-Free Grammars

In this section, we introduce some concepts from the theory of formal languages [6] and reformulate the Viterbi algorithm accordingly, replacing unwieldy trellis diagrams with sets of simple grammatical rules.

A *context-free grammar* (CFG) is a four-tuple $G = (V, T, P, S)$ consisting of two finite sets of symbols—T of *terminals* and V of *variables* (including S, the *start symbol*)—and a finite set P of *productions*, grammatical rules of the form $A \rightarrow \alpha$, where A is a variable and α is a string of symbols from $V \bigcup T$. G generates a *context-free language* L(G), a set of strings over the alphabet T of terminals each element of which can be derived from S (i.e. constructed by applying a finite sequence of productions to the start symbol). Context-free languages lacking the empty string (i.e. prohibiting any null productions) can be generated by grammars in *Chomsky normal form* with all productions of the type $A \rightarrow BC$ or $A \rightarrow \alpha$ (A,B,C being variable and α being terminal). Thus, the derivation of any string in such a language can be viewed as a binary tree with each leaf extended by a single additional branch and vertex; each interior vertex represents a variable (the tree root being S itself), each exterior vertex a terminal, and each branching a production.

A special subclass of context-free languages within Chomsky's linguistic hierarchy is

the class of regular languages. A *regular language* is generated by a *regular grammar*, a context-free grammar that is *right-linear* (having productions of the form $A \rightarrow wB$ or $A \rightarrow w$ where A and B are variables and w is a string of terminals). The derivation of a string belonging to a regular language can be viewed as a linear graph, a finite sequence of connected node-branch pairs in which each node represents a variable, each branch a string of terminals, and each pair a production.

Without rigorously demonstrating the equivalence of finite automata and regular languages, we clearly see that trellis-codes (e.g. convolutional codes) are generated by regular grammars over the binary alphabet. Each path through a trellis diagram is a right linear derivation; state labels of trellis nodes correspond to grammatical variables, whereas node-branch pairs correspond to productions. The codebook, the aggregate of all possible paths through the trellis diagram—a collection of right-linearly derived strings—is thus a regular language. Similarly, from the grammatical point of view, the Viterbi algorithm performs maximum likelihood decoding by recursively parsing the code's regular grammar. To find the minimal "length" string ending in the variable A at time $t+1$, it computes the minimum "length" of all strings ending in the variable A that are derivable (in a single production) from the set of minimum "length" strings terminating in an arbitrary variable at time t.

Having recognized the equivalence of maximum likelihood decoding by the Viterbi algorithm and parsing regular grammars by dynamic programming, in the following chapters we generalize Viterbi's approach to the class of codes derivable from context-free grammars.

Chapter 2

Maximum Likelihood Decoding of CFG Representable Codes

In this chapter, we develop generalized Viterbi algorithms for the maximum likelihood decoding of codes derived from context-free grammars and transmitted across either memoryless or Markov communications channels. Moreover, we introduce similar algorithms for computing an important reliability statistic—the posterior probability that a decoded word indeed matches the transmitted codeword.

2.1 A Grammatical Template

A length $N = 2^p$ binary block code $\mathcal{C}$ generated by a context-free grammar in Chomsky normal form can be viewed as a collection of binary derivation trees each having nodes occupied by symbols and branches representing productions. Our general template [5] for such codes is therefore a binary tree with *levels* labeled $0 \leq l \leq p$ each containing 2^{p-l} *nodes*. Each node at level l may exist in any one of the *states* $\{0, \ldots, N^{(l)} - 1\}$ representing allowed grammatical symbols. At level 0, the state of the i-th node represents the value of a codeword's i-th bit—$N^{(0)} = 2$. Moreover, the uniqueness of a grammar's start symbol requires $N^{(p)} = 1$. Within this framework, the set of allowed *productions* from a level l node in state j is $I_j^{(l)} \subset \{0, \ldots, N^{(l-1)} - 1\}^2$, a subset of state-pairs permitted its level $(l-1)$ daughter nodes. Obviously, $I_j^{(0)} = \emptyset$ for $j = 0, 1$. The collections $\{N^{(l)} | 0 \leq l \leq p\}$ and $\{I_j^{(l)} | 0 \leq l \leq p, 0 \leq j \leq N^{(l)} - 1\}$ uniquely determine both the code $\mathcal{C}$ and its associated context-free grammar.

To encode a stream of source data, one systematically selects productions (to be applied to the start symbol) according to successive subsequences of information bits. For the family of codes constructed in Chapter 3, the number of productions available at at node depends only on its level, taking the form

$$|I^{(l)}| \triangleq |I_j^{(l)}| = 2^{Q^{(l)}}$$

for $0 \leq j \leq N^{(l)} - 1$. The first $Q^{(p)}$ information bits select a single production from the root node at level p in state 0, the state-pair $(c_1^{(p-1)}, c_2^{(p-1)})$ at level $p-1$. In general, $2^{p-l}Q^{(l)}$ bits select 2^{p-l} independent productions from a 2^{p-l}-tuple $\mathbf{c}^{(l)}$ of states at level l, yielding a 2^{p-l+1}-tuple $\mathbf{c}^{(l-1)}$ of states at level $l-1$. At each level of the tree-template, there is a grammatically permitted subset $\mathcal{C}^{(l)}$ containing 2^{p-l}-tuples of level l states; $\mathcal{C}^{(0)}$ is itself the

codebook $\mathcal{C}$. Thus, the number of information bits encoded within a single codeword is

$$I(p, \{Q^{(l)}\}) = \sum_{l=1}^{p} 2^{p-l} Q^{(l)}.$$

2.2 Memoryless Channels

Suppose a codeword from $\mathcal{C}$ is transmitted across a memoryless communications channel (in which bits are independently corrupted). The maximum likelihood decoding of the received word $\mathbf{d}$ is the codeword

$$\hat{\mathbf{c}} = \arg \min_{\mathbf{c} \,\epsilon\, \mathcal{C}} \sum_{i=1}^{2^p} -\ln p(d_i|c_i)$$

that minimizes a bit-wise additive cost function. As in the regular grammar (or Viterbi) case, this optimization problem admits a dynamic programming solution [5].

Each leaf—or level 0 node—of the tree-template for $\mathcal{C}$ contributes a state-dependent cost to the total cost of a codeword:

$$C_i^0(j) = -\ln p(d_i|c_i = j),$$

for $1 \leq i \leq 2^p, 0 \leq j \leq 1$. Now consider a permissible assignment of states $\mathbf{c}^{(1)} \,\epsilon\, \mathcal{C}^{(1)}$ at level 1 and the corresponding subset $\mathcal{C}^{(0)}(\mathbf{c}^{(1)})$ of codewords derivable from $\mathbf{c}^{(1)}$ in 2^{p-1} independent productions. Evidently, only the minimum cost codeword $\mathbf{c}^{(0)}(\mathbf{c}^{(1)})$ within this restricted subset could possibly be a candidate for the overall cost minimum. Moreover, $\mathbf{c}^{(0)}(\mathbf{c}^{(1)})$ can be computed in 2^{p-1} independent minimizations; for each level 1 node, there is a production that minimizes the sum of the costs of its daughter nodes. Assigning this cost sum to the level 1 node in its given state defines an additive cost function on $\mathcal{C}^{(1)}$ which can in turn be minimized for any sequence of level 2 states. Iterating this procedure ultimately yields $\hat{\mathbf{c}}$.

With the nodal level 0 cost functions as initial data, the dynamic programming algorithm for maximum likelihood decoding proceeds as follows. For each level $1 \leq l \leq p$, recursively define the *optimal productions*

$$P_i^l(j) = \arg \min_{(k,k') \,\epsilon\, I_j^{(l)}} [C_{2i-1}^{l-1}(k) + C_{2i}^{l-1}(k')]$$

and the *optimal costs*

$$\begin{aligned} C_i^l(j) &= \min_{(k,k')\ \epsilon\ I_j^{(l)}} [C_{2i-1}^{l-1}(k) + C_{2i}^{l-1}(k')] \\ &= C_{2i-1}^{l-1}(P_i^l(j)_1) + C_{2i}^{l-1}(P_i^l(j)_2) \end{aligned}$$

for $1 \leq i \leq 2^{p-l}$ and $0 \leq j \leq N^{(l)} - 1$. Thus, $P_i^l(j)$ is the leading production in the derivation of the minimum cost ($C_i^l(j)$) substring derived from the ith level l node in state j. The maximum likelihood codeword $\hat{\mathbf{c}}$ has cost (negative log-likelihood) $C_1^p(0)$ and is derived by applying the set $\{P_i^l(j) | 1 \leq i \leq 2^{p-l}, 0 \leq j \leq N^{(l)} - 1, 1 \leq l \leq p\}$ of optimal productions to the level p start state 0:

$$\hat{\mathbf{c}}^{(p)} = 0 \rightarrow \cdots \rightarrow \hat{\mathbf{c}}^{(l)} \rightarrow \cdots \rightarrow \hat{\mathbf{c}}^{(0)} = \hat{\mathbf{c}},$$

where the lth optimal state sequence $\hat{\mathbf{c}}^{(l)}\ \epsilon\ \mathcal{C}^{(l)}$ is given by the relation

$$(\hat{c}_{2i-1}^{(l)}, \hat{c}_{2i}^{(l)}) = P_i^{l+1}(\hat{c}_i^{(l+1)})$$

for $1 \leq i \leq 2^{p-l-1}$.

Like Viterbi's algorithm, this dynamic programming procedure is significantly faster than a sequential search of the codebook. For every possible state at each positive level node in the tree, the algorithm performs an optimization over $|I^{(l)}|$ productions (requiring $|I^{(l)}|$ real number additions and $|I^{(l)}| - 1$ real number comparisons—hereafter referred to as *real number operations*). Thus, the number of decoding operations for a CFG representable code with parameters $\{N^{(l)}, |I^{(l)}| | 0 \leq l \leq p\}$ is

$$O(p, \{N^{(l)}\}, \{|I^{(l)}|\}) = \sum_{l=1}^{p} 2^{p-l} N^{(l)} (2|I^{(l)}| - 1). \tag{2.2.1}$$

For example, the thinned Reed–Muller code $RM^{(8)}(6, 10)$, a linear $[1024, 440, 16]$ code introduced in Chapter 3, is decodable in 2^{21} real number operations.

The dynamic programming approach can also be used to determine the reliability of a decoding scheme [5]. Assuming a uniform prior on the codebook $\mathcal{C}$, the posterior probability $p(\hat{\mathbf{c}}|\mathbf{d})$ that the decoded word $\hat{\mathbf{c}}$ was in fact transmitted, given that $\mathbf{d}$ was received is

expressed in equation 1.2.1. For a memoryless channel, it is more conveniently written:

$$p(\hat{\mathbf{c}}|\mathbf{d})^{-1} = \sum_{\mathbf{c}\ \epsilon\ \mathcal{C}} \prod_{i=1}^{2^p} \frac{p(d_i|c_i)}{p(d_i|\hat{c}_i)}.$$

Now observe that the right-hand sum can be computed by a dynamic programming algorithm virtually identical to the decoding procedure previously discussed. The nodal state-dependent level 0 contributions to the inverse posterior probability are

$$S_i^0(j) = \frac{p(d_i|c_i = j)}{p(d_i|\hat{c}_i)}$$

for $1 \leq i \leq 2^p, 0 \leq j \leq 1$. For each level $1 \leq l \leq p$, recursively define

$$S_i^l(j) = \sum_{(k,k')\ \epsilon\ I_j^{(l)}} S_{2i-1}^{l-1}(k) S_{2i}^{l-1}(k').$$

The posterior probability

$$p(\hat{\mathbf{c}}|\mathbf{d}) = S_1^p(0)^{-1}$$

is therefore computable in as many real number operations (2.2.1) as the maximum likelihood codeword itself.

2.3 Markov Channels

Although it is ubiquitous in the coding theory literature, the memoryless channel, which corrupts individual bits independently, is an unrealistic model for many actual communications channels. Often, real channel errors appear in bursts, corrupting entire sequences of bits [10]. The simplest mathematical model of a transmission line subject to burst noise is a *Markov channel* in which bit-errors are more likely than non-errors to succeed a given error. Having accepted an input $\mathbf{c}\ \epsilon\ \mathcal{C}$, such a channel produces the Markov process $\mathbf{d}$ as output, according to the probability

$$p(\mathbf{d}|\mathbf{c}) = \prod_{i=1}^{2^p} p_i(d_i|c_i, c_{i-1}, d_{i-1}).$$

Note that in the $i = 1$ term the dummy parameters c_0 and d_0 should be ignored. For clarity, we propose a general class of Markov channel models for which the channel proba-

bility necessarily factors as above: given a channel input $\mathbf{c}$, invertible observation functions $O_i(c_i, \cdot)$, and a hidden Markov process $\{e_i | 1 \leq i \leq 2^p\}$, we define the channel output $\mathbf{d}$ by the relation $d_i = O_i(c_i, e_i)$. The maximum likelihood decoding of $\mathbf{d}$ is the codeword

$$\hat{\mathbf{c}} = \arg \min_{\mathbf{c} \, \epsilon \, \mathcal{C}} \sum_{i=1}^{2^p} - \ln p_i(d_i | c_i, c_{i-1}, d_{i-1}).$$

In contrast to the memoryless channel problem addressed in section 2.2, each level 0 adjacent *pair* of leaves in the tree-template contributes a state-dependent cost to the total cost of a codeword. However, by again minimizing over the state-spaces of judiciously chosen subsets of nodes, one can systematically reduce the problem level by level.

Consider a peculiar subset $\mathcal{C}'$ of the code $\mathcal{C}$, those codewords derivable from a permissible assignment of level 2 states $\mathbf{c}^{(2)} \, \epsilon \, \mathcal{C}^{(2)}$, yet sharing a given fixed set of bits $\{c_i | i \bmod 4 = 0, 1\}$. Codewords within $\mathcal{C}'$ are composed of 2^{p-2} concatenated 4-bit substrings in which the first and fourth *flanking* bits are fixed while the second and third *internal* bits remain variable. The total cost of such a codeword is analogously partitioned into 2^{p-2} variable contributions—each the sum of the costs associated with the three adjacent state-pairs contained in a substring—and a fixed contribution—the sum of the *interaction* costs of adjacent pairs of flanking bits. Having fixed the cost of interactions between substrings by fixing their flanking bits, the minimum cost codeword in $\mathcal{C}'$—and candidate for the overall minimizer—can be computed in 2^{p-2} independent minimizations; for each *trio*, consisting of a level 2 node and the two bits flanking its associated substring, there is a derivation of interior bits that minimizes the variable substring cost. Assigning this minimum cost to the trio and defining the interaction cost of adjacent trios to be the interaction cost of their corresponding flanking bits effectively reduces the original problem over $\mathcal{C}$ to a formally identical problem over the significantly smaller set of flanking bits. Iterating this procedure ultimately yields $\hat{\mathbf{c}}$.

The dynamic programming algorithm for the maximum likelihood decoding of the output $\mathbf{d}$ of a Markov communications channel is formally presented as follows. First, for each state $0 \leq j \leq N^{(l)} - 1$ at level $2 \leq l \leq p$, establish a set of *auxiliary productions*,

$$\begin{aligned} \tilde{I}_j^{(l)} \quad \triangleq \quad & \{(k, k') \epsilon \, \mathbf{Z}_2^2 | (k, \alpha, \beta, k') \epsilon \bigcup_{\mathbf{p} \epsilon \, I_j^{(l)}} \tilde{I}_{p_1}^{(l-1)} \times \tilde{I}_{p_2}^{(l-1)}, \\ & \text{for some } \alpha, \beta \, \epsilon \, \mathbf{Z}_2 \} \end{aligned}$$

with $\tilde{I}_j^{(1)} \triangleq I_j^{(1)}$ for $0 \le j \le N^{(1)} - 1$. $\tilde{I}_j^{(l)}$ represents the set of *flanking bits* that can be grammatically derived from the level l state j. Second, assemble the algorithm's initial data—level 0 and 1 state-dependent costs—respectively defined by:

$$Q_i^0(j, j') = -\ln p_i(d_i | c_i = j', c_{i-1} = j, d_{i-1})$$

for $1 \le i \le 2^p, 0 \le j, j' \le 1$, and

$$Q_i^1(j, \mathbf{a}) = Q_{2i}^0(a_1, a_2)$$

for $\mathbf{a} \epsilon\ \tilde{I}_j^{(1)}, 0 \le j \le N^{(1)} - 1$, and $1 \le i \le 2^{p-1}$.

Third, for each level $2 \le l \le p$, each node $1 \le i \le 2^{p-l}$, each state $0 \le j \le N^{(l)} - 1$, each auxiliary production $\mathbf{a} \epsilon\ \tilde{I}_j^{(l)}$, each ordinary production $\mathbf{q} \epsilon\ I_j^{(l)}$, and each interior sequence $\mathbf{b} \epsilon\ \mathbf{Z}_2^2$, recursively define the *objective function*

$$\begin{aligned} K_i^l(j, \mathbf{a}, \mathbf{q}, \mathbf{b}) &= Q_{2i-1}^{l-1}(q_1, (a_1, b_1)) + Q_{2i}^{l-1}(q_2, (b_2, a_2)) \\ &\quad + Q_{(i-1)2^l + 2^{l-1}+1}^0(b_1, b_2), \end{aligned}$$

the *optimal production*

$$P_i^l(j, \mathbf{a}) = \arg \min_{\{(\mathbf{q}, \mathbf{b}) \epsilon\ I_j^{(l)} \times \mathbf{Z}_2^2 | (a_1, b_1, b_2, a_2) \epsilon\ \tilde{I}_{q_1}^{(l-1)} \times \tilde{I}_{q_2}^{(l-1)}\}} K_i^l(j, \mathbf{a}, \mathbf{q}, \mathbf{b}),$$

and the *optimal cost*

$$\begin{aligned} Q_i^l(j, \mathbf{a}) &= \min_{\{(\mathbf{q}, \mathbf{b}) \epsilon\ I_j^{(l)} \times \mathbf{Z}_2^2 | (a_1, b_1, b_2, a_2) \epsilon\ \tilde{I}_{q_1}^{(l-1)} \times \tilde{I}_{q_2}^{(l-1)}\}} K_i^l(j, \mathbf{a}, \mathbf{q}, \mathbf{b}) \\ &= K_i^l(j, \mathbf{a}, \mathbf{q}, \mathbf{b})|_{P_i^l(j, \mathbf{a})}. \end{aligned}$$

Fourth, compute the optimal flanking bits for the level p start state

$$\mathbf{a}^{(p-1)} = \arg \min_{\mathbf{a} \epsilon\ \tilde{I}_0^{(p)}} Q_1^0(0, a_1) + Q_1^p(0, \mathbf{a})$$

and the overall minimum cost

$$\begin{aligned} Q &= \min_{\mathbf{a}\epsilon\ \tilde{I}_0^{(p)}} Q_1^0(0,a_1) + Q_1^p(0,\mathbf{a}) \\ &= Q_1^0(0,a_1^{(p-1)}) + Q_1^p(0,\mathbf{a}^{(p-1)}). \end{aligned}$$

Finally, apply the set $\{P_i^l(j,\mathbf{a})|2 \leq l \leq p, 1 \leq i \leq 2^{p-l},\ 0 \leq j \leq N^{(l)}-1,\ \mathbf{a}\epsilon\ \tilde{I}_j^{(l)}\}$ of optimal productions to the optimal start trio $(0,\mathbf{a}^{(p-1)})$ to generate the maximum likelihood codeword $\hat{\mathbf{c}}$ of cost (negative log-likelihood) Q. Given the optimal state sequence $\{c_i^{(l)}|1 \leq i \leq 2^{p-l}\}$ at level l and the associated optimal sequence $\{a_i^{(l-1)}|1 \leq i \leq 2^{p-l+1}\}$ of flanking bits—with the pair $(a_{2i-1}^{(l-1)}, a_{2i}^{(l-1)})$ flanking $c_i^{(l)}$, the optimal state and flanking sequences $\{c_i^{(l-1)}|1 \leq i \leq 2^{p-l+1}\}$ and $\{a_i^{(l-2)}|1 \leq i \leq 2^{p-l+2}\}$ are computed for $p \geq l \geq 2$ as follows. If for $1 \leq i \leq 2^{p-l}$ $P_i^l(c_i^{(l)}, (a_{2i-1}^{(l-1)}, a_{2i}^{(l-1)}))$ equals $(q_1,q_2,b_1,b_2)\epsilon\ I_{c_i^{(l)}}^{(l)} \times \mathbf{Z}_2^2$, assign:

$$\begin{aligned} c_{2i-1}^{(l-1)} &= q_1, & c_{2i}^{(l-1)} &= q_2, \\ a_{4i-2}^{(l-2)} &= b_1, & a_{4i-1}^{(l-2)} &= b_2, \\ a_{4i-3}^{(l-2)} &= a_{2i-1}^{(l-1)}, & a_{4i}^{(l-2)} &= a_{2i}^{(l-1)}. \end{aligned}$$

At level 0, the optimal flanking sequence is $\hat{\mathbf{c}}$ itself: $\hat{c}_i = a_i^{(0)},\ 1 \leq i \leq 2^p$.

As before, the computational complexity of this dynamic programming algorithm for decoding the output of a Markov communications channel can be expressed in terms of the grammar's parameters $\{N^{(l)}, |I^{(l)}||0 \leq l \leq p\}$. Comparing the Markov algorithm to the memoryless one, we see that the effective sizes of the state and production spaces in the former case are four times that of the latter. Moreover, each step requires two (rather than one) real number additions. Thus, the number of decoding operations for a Markov channel is

$$O^{Markov}(p,\{N^{(l)}\},\{|I^{(l)}|\}) = \sum_{l=2}^{p} 2^{p-l} 4 N^{(l)} (12|I^{(l)}| - 1), \tag{2.3.2}$$

or approximately 24 times the number required for a memoryless channel.

We close this chapter by presenting an analogous dynamic programming algorithm for computing the reliability of the above Markov channel decoding scheme. The posterior probability $p(\hat{\mathbf{c}}|\mathbf{d})$ that the maximum likelihood codeword $\hat{\mathbf{c}}$ was in fact transmitted, given

that $\mathbf{d}$ was received, can be expressed for the Markov channel as

$$p(\hat{\mathbf{c}}|\mathbf{d})^{-1} = \sum_{\mathbf{c}\epsilon\ \mathcal{C}} \prod_{i=1}^{2^p} \frac{p_i(d_i|c_i, c_{i-1}, d_{i-1})}{p_i(d_i|\hat{c}_i, \hat{c}_{i-1}, d_{i-1})}.$$

Assemble the algorithm's initial data at the respective levels 0 and 1:

$$S_i^0(j, j') = \frac{p_i(d_i|c_i = j', c_{i-1} = j, d_{i-1})}{p_i(d_i|\hat{c}_i, \hat{c}_{i-1}, d_{i-1})}$$

for $1 \leq i \leq 2^p$ and $0 \leq j, j' \leq 1$; and

$$S_i^1(j, \mathbf{a}) = S_{2i}^0(a_1, a_2)$$

for $1 \leq i \leq 2^{p-1}$, $0 \leq j \leq N^{(1)} - 1$, and $\mathbf{a}\epsilon\ \tilde{I}_j^{(1)}$. At the successive levels $2 \leq l \leq p$, compute

$$S_i^l(j, \mathbf{a}) = \sum_{\mathbf{q}\epsilon\ I_j^{(l)}} \sum_{\{\mathbf{b}\epsilon\ \mathbf{Z}_2^2 | (a_1, b_1, b_2, a_2)\epsilon\ \tilde{I}_{q_1}^{(l-1)} \times \tilde{I}_{q_2}^{(l-1)}\}} T_i^l(j, \mathbf{a}, \mathbf{q}, \mathbf{b}),$$

where

$$T_i^l(j, \mathbf{a}, \mathbf{q}, \mathbf{b}) = S_{2i-1}^{l-1}(q_1, (a_1, b_1)) S_{2i}^{l-1}(q_2, (b_2, a_2)) S_{(i-1)2^l + 2^{l-1}+1}^0(b_1, b_2)$$

for $1 \leq i \leq 2^{p-l}$, $0 \leq j \leq N^{(l)} - 1$, $\mathbf{a}\epsilon\ \tilde{I}_j^{(l)}$, $\mathbf{q}\epsilon\ I_j^{(l)}$, and $\mathbf{b}\epsilon\ \mathbf{Z}_2^2$. The posterior probability is then given by

$$p(\hat{\mathbf{c}}|\mathbf{d})^{-1} = \sum_{\mathbf{a}\epsilon\ \tilde{I}_0^{(p)}} S_1^0(0, a_1) S_1^p(0, \mathbf{a}),$$

and is computed in as many real number operations (2.3.2) as the maximum likelihood codeword itself.

Chapter 3

Construction of CFG Representable Codes

In 1988, Forney [4] introduced a general technique known as the squaring construction to provide a unified framework for analyzing coset codes. In this chapter, we reinterpret the squaring construction as a grammatical device, deriving in the process context-free grammars for a large family of useful codes—including the Reed–Muller codes introduced in Section 1.1.2.

3.1 The Squaring Construction

If a code S is partitioned by M subsets $T_i, 1 \leq i \leq M$, with minimum distances $d(S) \leq d(T) = \min_i d(T_i)$, the (*true*) *squaring construction* is defined to be the code $U = |S/T|^2 \stackrel{\Delta}{=} \bigcup_{i=1}^{M} T_i \times T_i$ of minimum distance $d(U) = \min[d(T), 2d(S)]$. Alternatively, given any non-identity permutation π on the set $\{1 \leq i \leq M\}$, a (*twisted*) *squaring construction* can be defined as the code $U^\pi \stackrel{\Delta}{=} \bigcup_{i=1}^{M} T_{\pi(i)} \times T_i$ of minimum distance $d(T) \geq d(U^\pi) \geq \min[d(T), 2d(S)]$. True or twisted, the squaring construction is simply a technique for creating larger, greater distance codes from a given code.

When the relevant codes are linear, the squaring construction is particularly elegant [4]. If S is a group partitioned by the cosets of a subgroup T having index $|S/T| = M$, the true squaring construction, $U = |S/T|^2 \stackrel{\Delta}{=} \bigcup_{\mathbf{c}\ \epsilon\ [S/T]} T^2 + (\mathbf{c}, \mathbf{c})$ is itself a group nested between the groups S^2 and T^2 ($S^2 > U > T^2$). If S is an $(N, |S|, d(S))$ code and T is an $(N, |T|, d(T))$ code, then U is a $(2N, |S||T|, \min[d(T), 2d(S)])$ code. Invoking the notation of Section 1.1.2, we choose the following convenient sets of cosets representatives:

$$[U/T^2] = \mathbf{g}_1 \otimes [S/T] \stackrel{\Delta}{=} \{(\mathbf{c}, \mathbf{c}) | \mathbf{c}\ \epsilon\ [S/T]\}$$

$$[S^2/U] = \mathbf{g}_0 \otimes [S/T] \stackrel{\Delta}{=} \{(\mathbf{c}, \mathbf{0}) | \mathbf{c}\ \epsilon\ [S/T]\}$$

$$[S^2/T^2] = [S^2/U] + [U/T^2] = (\mathbf{g}_0 + \mathbf{g}_1) \otimes [S/T] = [S/T]^2.$$

(Note that since $G_{(2,2)} = \{\mathbf{g}_0, \mathbf{g}_1\}$ is a basis for $\mathbf{Z}_2^2$, $(\mathbf{g}_0 + \mathbf{g}_1) \otimes A = A^2$ is an identity for any set A.) The group squaring construction can thus be succinctly expressed as either of the direct sums

$$|S/T|^2 = T^2 + \mathbf{g}_1 \otimes [S/T]$$

or

$$|S/T|^2 = \mathbf{g}_1 \otimes S + \mathbf{g}_0 \otimes T.$$

The constructions $|S/S|^2 = S^2$ and $|S/\{\mathbf{0}\}|^2 = \mathbf{g}_1 \otimes S$ correspond to trivial partitions of the set S.

Now suppose each subset T_i of S is generated by a context-free grammar. In other words, there exist M grammatical symbols t_i from which the elements of T_i are derived. It immediately follows that the squaring construction U^π itself is generated by a context-free grammar having start symbol u^π and associated productions $u^\pi \rightarrow t_{\pi(i)}t_i, 1 \leq i \leq M$. By induction, any iterated squaring construction (in which the T_i are themselves squaring constructions on squaring constructions ...) on a set of CFG representable codes is also generated by a context-free grammar [5]. This notion will be clarified in the following sections, wherein we will construct a large family of CFG representable codes by iterating the squaring construction.

3.2 Iterated Squaring Constructions

Iterating the group squaring construction is relatively straightforward. Suppose $S_0 > S_1 > \cdots > S_m$ is a nested sequence of subgroups with minimum distances $d(S_0) \leq d(S_1) \leq \cdots \leq d(S_m)$. Since $S_j^2 > |S_j/S_{j+1}|^2 > S_{j+1}^2$, the m group squaring constructions $|S_j/S_{j+1}|^2, 0 \leq j \leq m-1$, derived from this chain also form a nested sequence of subgroups, $|S_0/S_1|^2 > |S_1/S_2|^2 > \cdots > |S_{m-1}/S_m|^2$. Repeating this procedure m times, successively squaring adjacent pairs of subgroups in the current chain to create the next chain, yields a single group of 2^m-tuples over S_0—the *m-level iterated group squaring construction* denoted by $|S_0/S_1/\cdots/S_m|^M$ $(M = 2^m)$ [4].

In general, it is more convenient [4] to analyze the iterated squaring construction on an infinite subgroup chain composed of the original chain $S_0 > S_1 > \cdots > S_m$ extended to the left and right by the respective dummy sequences $\cdots > S_0 > S_0 >$ and $> S_m > S_m > \cdots$ $(S_{-j} = S_0$ and $S_{m+j} = S_m$ for $j \geq 1)$. Successively squaring adjacent pairs of subgroups yields a tableaux $\{S_{nj} | n \geq 0, j \in \mathbf{Z}\}$ of codes defined by the recursion:

$$S_{0,j} = S_j,$$

$$S_{n+1,j} = |S_{n,j}/S_{n,j+1}|^2 = \mathbf{g}_1 \otimes S_{n,j} + \mathbf{g}_0 \otimes S_{n,j+1}$$

for $n \geq 0, j \in \mathbf{Z}$. The group $S_{n,j} = |S_j/S_{j+1}/\cdots/S_{j+n}|^N$ $(N = 2^n)$ of 2^n-tuples over S_0 has minimum distance

$$d(S_{n,j}) = \min[d(S_{n-1,j+1}), 2d(S_{n-1,j})]$$

and generator matrix

$$G_{n+1,j} = \mathbf{g}_1 \otimes G_{n,j} + \mathbf{g}_0 \otimes G_{n,j+1}$$

for $n \geq 0, j \in \mathbf{Z}$. Iterating these relations n times, we find that

$$d(S_{n,j}) = \min[d(S_{j+n}), 2d(S_{j+n-1}), \cdots, 2^n d(S_j)] \tag{3.2.1}$$

and

$$S_{n,j} = L(G_{n,j})$$

where

$$G_{n,j} = \sum_{r=0}^{n} G_{\partial RM}(r,n) \otimes G_{j+r}$$

(G_j being the generator matrix of S_j) for $n \geq 0, j \in \mathbf{Z}$.

Reed–Muller codes are the principal examples of iterated group squaring constructions [4]. Applying the above construction to the subgroup chain $S_0 = \mathbf{Z}_2 = RM(0,0) > S_1 = \{0\} = RM(-1,0)$ with generator matrices $G_{-j} = G_0 = \{1\}$ and $G_j = G_j = \{0\}$, $j \geq 1$, yields the family $\{S_{n,-j} | n \geq 0, 0 \leq j \leq n\}$ of codes generated by

$$G_{n,-j} = \sum_{r=0}^{n} G_{\partial RM}(r,n) \otimes G_{r-j} = \sum_{r=0}^{j} G_{\partial RM}(r,n) = G_{RM}(j,n);$$

hence, $S_{n,-j} = RM(j,n)$ for $n \geq 0, 0 \leq j \leq n$. Moreover, by evaluating equation 3.2.1 with $d(S_0) = 1$ and $d(S_1) = \infty$, we find that $d(RM(j,n)) = d(S_{n,-j}) = 2^{n-j}$.

Iterated squaring constructions on sets that are not groups are far more cumbersome to describe. Suppose the set S is partitioned by the M sets T_i which are each in turn partitioned by the N sets V_{ij}. The true squaring construction $U = |S/T|^2$ is partitioned by sets of the form $T_i \times T_i$ each of which can itself be partitioned by N twisted squaring constructions over the sets V_{ij}—for N permutations π_k^i on the set $\{1 \leq j \leq N\}$ satisfying $\pi_k^i(j) \neq \pi_{k'}^i(j)\ \forall\ k \neq k'$, $T_i \times T_i = \bigcup_{k=1}^N \bigcup_{j=1}^N V_{i\pi_k^i(j)} \times V_{ij}$. If these MN component squaring

constructions are labeled $|T/V|^2_{ik}$, the 2-level iterated set squaring construction $|S/T/V|^4$ is defined to be $||S/T|^2/|T/V|^2|^2 = \bigcup_{i=1}^{M} \bigcup_{k=1}^{N} |T/V|^2_{ik} \times |T/V|^2_{ik}$. As with the iterated group squaring construction, extending the initial partition chain analogously increases the number of possible iterations [4]. Whereas all the partitions of iterated group squaring constructions are induced by given subgroups, the permutations required to partition iterated set squaring constructions must be carefully chosen at each stage of the iteration. We therefore postpone further discussion of iterated set squaring constructions until the labeling system of the next section is introduced.

3.3 Reed–Muller Grammars

As iterated group squaring constructions, the Reed–Muller codes of length 2^p form a nested sequence of subgroups,

$$RM(p,p) = \mathbf{Z}_2^{2^p} > RM(p-1,p) > \cdots > RM(0,p) > RM(-1,p) = \{\mathbf{0}\}.$$

The space of binary 2^p-tuples therefore admits the coset decomposition:

$$\begin{aligned} RM(p,p) &= [RM(p,p)/RM(p-1,p)] + [RM(p-1,p)/RM(p-2,p)] + \cdots \\ &\quad + [RM(0,p)/RM(-1,p)] + RM(-1,p). \end{aligned}$$

In other words, each binary string in $\mathbf{Z}_2^{2^p}$ can be uniquely expressed as a sum of coset representatives as follows:

$$B^p_{i_0,i_1,\ldots,i_p} = \sum_{k=0}^{p} \mathbf{c}^{(k,p)}_{i_k}$$

where $0 \le i_k \le m^p_k - 1, 0 \le k \le p$,

$$m^p_k \triangleq 2^{\binom{p}{k}} = |RM(p-k,p)/RM(p-k-1,p)|,$$

and

$$\mathbf{c}^{(k,p)}_{i_k} \epsilon\ [RM(p-k,p)/RM(p-k-1,p)] = L(G_{\partial RM}(p-k,p)).$$

However, there remains some ambiguity in the labeling of the coset representatives $\mathbf{c}^{(k,p)}_{i_k}$.

In order to resolve this ambiguity, we define

$$\mathbf{c}_{i_k}^{(k,p)} = \mathbf{i}_k \overline{G}_{\partial RM}(p-k,p)$$

where $\mathbf{i}_k$ is the $\binom{p}{k}$-bit binary representation of the integer $0 \le i_k \le m_k^p - 1$ and $\overline{G}_{\partial RM}(p-k,p)$ is the $\binom{p}{k} \times 2^p$ matrix of rows $\mathbf{g}(j_1, j_2, \ldots, j_p) \stackrel{\Delta}{=} \mathbf{g}_{j_1} \otimes \mathbf{g}_{j_2} \otimes \cdots \otimes \mathbf{g}_{j_p}$ from the generator matrix $G_{\partial RM}(p-k,p)$ ordered lexicographically by label $j_1 j_2 \cdots j_p$—largest first. Alternatively, the matrices $\{\overline{G}_{\partial RM}(p-k,p) | 0 \le k \le p, p \ge 0\}$ can be constructed by the recursions:

$$\overline{G}_{\partial RM}(p-k,p) = \begin{cases} 1 & k = p = 0 \\ \mathbf{g}_0 \otimes \overline{G}_{\partial RM}(p-1,p-1) & k = 0, p \ge 1 \\ \begin{bmatrix} \mathbf{g}_1 \otimes \overline{G}_{\partial RM}(p-k,p-1) \\ \mathbf{g}_0 \otimes \overline{G}_{\partial RM}(p-k-1,p-1) \end{bmatrix} & 1 \le k \le p-1, p \ge 1 \\ \mathbf{g}_1 \otimes \overline{G}_{\partial RM}(0,p-1) & k = p, p \ge 1. \end{cases}$$

Evidently, the coset representatives inherit a similar recursive structure:

$$\begin{aligned} \mathbf{c}_{i_0}^{(0,0)} &= i_0, && 0 \le i_0 \le 1; \\ \mathbf{c}_{i_0}^{(0,p)} &= \mathbf{g}_0 \otimes \mathbf{c}_{i_0}^{(0,p-1)}, && 0 \le i_0 \le 1, p \ge 1; \\ \mathbf{c}_{i_k}^{(k,p)} &= \mathbf{g}_1 \otimes \mathbf{c}_{\lfloor i_k / m_k^{p-1} \rfloor}^{(k-1,p-1)} + \mathbf{g}_0 \otimes \mathbf{c}_{i_k \bmod m_k^{p-1}}^{(k,p-1)}, && 1 \le k \le p, 0 \le i_k \le m_k^p - 1, p \ge 1; \\ \mathbf{c}_{i_p}^{(p,p)} &= \mathbf{g}_1 \otimes \mathbf{c}_{i_p}^{(p-1,p-1)}, && 0 \le i_p \le 1, p \ge 1. \end{aligned}$$

In addition, with this choice of coset representatives the B-symbols can themselves be constructed iteratively:

$$\begin{aligned} B_{i_0}^0 &= i_0 \\ B_{i_0,i_1,\ldots,i_p}^p &= \mathbf{g}_1 \otimes B_{\lfloor i_1/m_1^{p-1} \rfloor, \ldots, \lfloor i_{p-1}/m_{p-1}^{p-1} \rfloor, i_p}^{p-1} \\ &\quad + \mathbf{g}_0 \otimes B_{i_0 \bmod m_0^{p-1}, \ldots, i_{p-2} \bmod m_{p-2}^{p-1}, i_{p-1} \bmod m_{p-1}^{p-1}}^{p-1} \end{aligned}$$

for $0 \le i_k \le m_k^p - 1, 0 \le k \le p, p \ge 0$.

Since coset representatives are linearly related to their labels, the mod-2 vector sum of

B-symbols is simply

$$B^p_{i_0,i_1,\ldots,i_p} + B^p_{j_0,j_1,\ldots,j_p} = B^p_{f(i_0,j_0),f(i_1,j_1),\ldots,f(i_p,j_p)}$$

where the function f performs mod-2 vector addition on the binary expansions of its integer arguments

$$f(i,j) \triangleq \sum_{l=0}^{\infty} \left[(\lfloor i/2^l \rfloor + \lfloor j/2^l \rfloor) \bmod 2 \right] 2^l.$$

In other words, f is the bit-wise exclusive OR operator. For fixed $0 \leq i, j \leq m^p_k - 1$, f exhibits the following general properties:

1. $f(i,\cdot)$ is a permutation on the set $\{0 \leq j \leq m^p_k - 1\}$;
2. $f(0,j) = j$;
3. f is symmetric: $f(i,j) = f(j,i)$;
4. $f(j,j) = 0$.

Having precisely defined the addition of B-symbols, their iterative construction can be reexpressed in the explicitly grammatical form

$$\begin{aligned} B^0_{i_0} &= i_0 \\ B^p_{i_0,i_1,\ldots,i_p} &= B^{p-1}_{f(\lfloor i_1/m_1^{p-1} \rfloor, i_0 \bmod m_0^{p-1}),\ldots,f(\lfloor i_{p-1}/m_{p-1}^{p-1} \rfloor, i_{p-2} \bmod m_{p-2}^{p-1}),f(i_p, i_{p-1} \bmod m_{p-1}^{p-1})} \\ &\quad \times B^{p-1}_{\lfloor i_1/m_1^{p-1} \rfloor,\ldots,\lfloor i_{p-1}/m_{p-1}^{p-1} \rfloor, i_p}. \end{aligned} \tag{3.3.2}$$

If the B-symbols are interpreted as variables in a context-free grammar, these expressions are equivalent to the following grammatical production rules:

$$\begin{aligned} B^0_{i_0} &\rightarrow i_0; \\ B^p_{i_0,i_1,\ldots,i_p} &\rightarrow B^{p-1}_{f(\lfloor i_1/m_1^{p-1} \rfloor, i_0 \bmod m_0^{p-1}),\ldots,f(\lfloor i_{p-1}/m_{p-1}^{p-1} \rfloor, i_{p-2} \bmod m_{p-2}^{p-1}),f(i_p, i_{p-1} \bmod m_{p-1}^{p-1})} \\ &\quad B^{p-1}_{\lfloor i_1/m_1^{p-1} \rfloor,\ldots,\lfloor i_{p-1}/m_{p-1}^{p-1} \rfloor, i_p}. \end{aligned}$$

Individual cosets of Reed–Muller codes, aggregates of binary strings, can also be expressed symbolically. For $1 \leq \alpha \leq p$, the cosets of $RM(p-\alpha, p)$ are

$$B^p_{i_0,i_1,\ldots,i_{\alpha-1}} \triangleq \{B^p_{i_0,i_1,\ldots,i_p} | 0 \leq i_k \leq m^p_k - 1, \alpha \leq k \leq p\}$$

$$= \quad RM(p-\alpha,p) + \sum_{k=0}^{\alpha-1} \mathbf{c}_{i_k}^{(k,p)}$$

$(0 \le i_k \le m_k^p - 1, 0 \le k \le \alpha - 1)$; similarly for $\alpha = 0$,

$$\begin{aligned} B^p &\triangleq \{B^p_{i_0,i_1,\ldots,i_p} | 0 \le i_k \le m_k^p - 1, 0 \le k \le p\} \\ &= RM(p,p) = \mathbf{Z}_2^{2^p}. \end{aligned}$$

The Reed–Muller codes $RM(p-\alpha,p)$, $0 \le \alpha \le p+1$, are indexed by exactly α zeros:

$$RM(p-\alpha,p) = B^p_{\underbrace{0,0,\ldots,0}_{\alpha}\cdot}$$

Reed–Muller cosets inherit an iterative structure from that of their component binary strings. Introducing the auxiliary parameters $x_k = \lfloor i_{k+1}/m_{k+1}^{p-1} \rfloor$ and $z_k = i_k \bmod m_k^{p-1}$ (with $m_p^{p-1} \triangleq 1$) for $0 \le k \le p-1$, the grammatical production 3.3.2 can be rewritten as:

$$B^p_{i_0,i_1,\ldots,i_p} \quad = \quad B^{p-1}_{f(x_0,z_0),f(x_1,z_1),\ldots,f(x_{p-1},z_{p-1})} \times B^{p-1}_{x_0,x_1,\ldots,x_{p-1}}.$$

Since the set $\{0 \le i_k \le m_k^p - 1\}$ can be put in one-to-one correspondence with the product set $\{0 \le x_{k-1} \le m_{k-1}^{p-1} - 1\} \times \{0 \le z_k \le m_k^{p-1} - 1\}$ by the relation $i_k = m_k^{p-1} x_{k-1} + z_k$, any coset of $RM(p-\alpha,p)$ can be expressed as

$$\begin{aligned} B^p_{i_0,i_1,\ldots,i_{\alpha-1}} &= \bigcup_{i_\alpha,i_{\alpha+1},\ldots,i_p} B^p_{i_0,i_1,\ldots,i_p} \\ &= \bigcup_{x_{\alpha-1},x_\alpha,\ldots,x_{p-1}} \bigcup_{z_\alpha,z_{\alpha+1},\ldots,z_{p-1}} B^{p-1}_{f(x_0,z_0),f(x_1,z_1),\ldots,f(x_{p-1},z_{p-1})} \times B^{p-1}_{x_0,x_1,\ldots,x_{p-1}} \\ &= \bigcup_{x_{\alpha-1}=0}^{m_{\alpha-1}^{p-1}-1} B^{p-1}_{f(x_0,z_0),f(x_1,z_1),\ldots,f(x_{\alpha-1},z_{\alpha-1})} \times B^{p-1}_{x_0,x_1,\ldots,x_{\alpha-1}} \end{aligned}$$

This last equality follows from performing the z-unions before the x-unions, using the fact that $f(x,\cdot)$ is a permutation for any fixed x and recognizing the resulting Reed–Muller cosets. When $\alpha = 0$, we recover the obvious result:

$$B^p \quad = \quad B^{p-1} \times B^{p-1}.$$

As with the string B-symbols, these coset B-symbols can be equivalently viewed as variables in a context-free grammar with production rules:

$$\begin{aligned}
B^0 &\rightarrow 0|1; \\
B^p &\rightarrow B^{p-1}B^{p-1}; \\
B^p_{i_0,i_1,\ldots,i_{\alpha-1}} &\stackrel{i}{\rightarrow} B^{p-1}_{f(\lfloor i_1/m_1^{p-1}\rfloor, i_0 \bmod m_0^{p-1}),\ldots,f(\lfloor i_{\alpha-1}/m_{\alpha-1}^{p-1}\rfloor, i_{\alpha-2} \bmod m_{\alpha-2}^{p-1}),f(i, i_{\alpha-1} \bmod m_{\alpha-1}^{p-1})} \\
&\qquad B^{p-1}_{\lfloor i_1/m_1^{p-1}\rfloor,\ldots,\lfloor i_{\alpha-1}/m_{\alpha-1}^{p-1}\rfloor,i}
\end{aligned}$$

for $0 \le i \le m_{\alpha-1}^{p-1} - 1$.

We summarize the results of this section in the following propositions.

Proposition 3.3.1 The linear $[2^p, 2^p, 1]$ code $\mathbf{Z}_2^{2^p} = RM(p,p)$ is generated by a context-free grammar with start symbol B^p and productions:

$$\begin{aligned}
B^l &= B^{l-1} \times B^{l-1} \\
B^0 &= \{0,1\}
\end{aligned}$$

for $1 \le l \le p$.

Proposition 3.3.2 The linear $[2^p, \sum_{s=0}^{p-\alpha}\binom{p}{s}, 2^\alpha]$ code $RM(p-\alpha,p)$ and its nonlinear $(2^p, \prod_{s=0}^{p-\alpha} m_s^p, 2^\alpha)$ cosets $(p+1 \ge \alpha \ge 1)$ are generated by context-free grammars with start symbols $\{B^p_{i_0,i_1,\ldots,i_{\alpha-1}} | 0 \le i_k \le m_k^p - 1,\ 0 \le k \le \alpha-1\}$ and productions:

$$\begin{aligned}
B^l_{j_0,j_1,\ldots,j_{K(l)}} &= \begin{cases} \bigcup_{x=0}^{m_{\alpha-1}^{l-1}-1} B^{l-1}_{f(x_0,z_0),\ldots,f(x_{\alpha-2},z_{\alpha-2}),f(x,z_{\alpha-1})} \times B^{l-1}_{x_0,\ldots,x_{\alpha-2},x} & \alpha \le l \le p \\ B^{l-1}_{f(x_0,z_0),\ldots,f(x_{l-2},z_{l-2}),f(x_{l-1},z_{l-1})} \times B^{l-1}_{x_0,\ldots,x_{l-2},x_{l-1}} & 1 \le l \le \alpha-1 \end{cases} \\
B^0_{j_0} &= j_0
\end{aligned}$$

where

$$\begin{aligned}
K(l) &\stackrel{\Delta}{=} \min(l, \alpha-1), \\
x_k &= \lfloor j_{k+1}/m_{k+1}^{l-1}\rfloor, \\
z_k &= j_k \bmod m_k^{l-1},
\end{aligned}$$

and $m_l^{l-1} \stackrel{\Delta}{=} 1$ for $0 \le j_k \le m_k^l - 1$, $0 \le k \le K(l)$, and $1 \le l \le p$.

3.4 Bent Reed–Muller Grammars

The analytic techniques developed in Section 3 provide an essential framework for examining more general partitions of the space $\mathbf{Z}_2^{2^p}$. In particular, the labeling system induced by the coset decomposition of RM(p,p) can be generalized to describe iterated set squaring constructions and their associated context-free grammars [5].

Consider a grammatical system of binary strings constructed according to the production rules:

$$\begin{aligned} A^0_{i_0} &= i_0 \\ A^p_{i_0,i_1,\ldots,i_p} &= A^{p-1}_{C^{p-1}_0(x_0,z_0),\ldots,C^{p-1}_{p-2}(x_{p-2},z_{p-2}),C^{p-1}_{p-1}(x_{p-1},z_{p-1})} \times A^{p-1}_{x_0,\ldots,x_{p-2},x_{p-1}} \end{aligned}$$

for $0 \le i_k \le m^l_k - 1$, $0 \le k \le p$, and $p \ge 1$. As in proposition 3.3.2, we define the following auxiliary parameters: $x_k = \lfloor i_{k+1}/m^{p-1}_{k+1} \rfloor$ and $z_k = i_k \bmod m^{p-1}_k$. Such a system is considered to be *complete* if for each $0 \le k \le p$ and $p \ge 0$, the function C^p_k exhibits the properties

1. $C^p_k(\cdot, z)$ is a permutation on the set $\{0 \le x \le m^p_k - 1\}$,
2. $C^p_k(x, 0) = x$ (i.e. $C^p_k(\cdot, 0)$ is the identity permutation), and
3. $C^p_k(x, z) \ne C^p_k(x, z')$ whenever $z \ne z'$ (i.e. $C^p_k(x, \cdot)$ is a permutation on the set $\{0 \le z \le m^p_k - 1\}$)

for $0 \le x, z, z' \le m^p_k - 1$. Property (1) will ensure that the codes constructed by aggregating A-symbols are indeed iterated set squaring constructions (proposition 3.4.2). As a result of property (2), these codes will be similar to Reed–Muller cosets (propositions 3.4.1,2), having particularly compact grammatical representations (lemma 3.5.2). And perhaps most important of all, property (3) guarantees that the set of all A^p-symbols completely partitions $\mathbf{Z}_2^{2^p}$. Two particularly simple complete grammatical systems are the Reed–Muller system ($C^p_k = f$) of Section 3 and the cyclic system [5] in which symbol indices are twisted by the cyclic permutations

$$C^p_k(i, j) = (i + j) \bmod m^p_k.$$

As demonstrated in the following lemma, complete grammatical systems partition $\mathbf{Z}_2^{2^p}$ into strings with distinctive distance properties.

Lemma 3.4.1 If $\{A^p_{i_0,i_1,\ldots,i_p} | 0 \le i_k \le m^p_k - 1, 0 \le k \le p, p \ge 0\}$ is a complete grammatical system, then for each $p \ge 0$:

1. $\mathbf{Z}_2^{2^p} = \{A^p_{i_0,i_1,\ldots,i_p} | 0 \le i_k \le m^p_k - 1, 0 \le k \le p\}$.

2. $d(A^p_{i_0,i_1,\ldots,i_p}, A^p_{i_0,i_1,\ldots,i_{l-1},j_l,\ldots,j_p}) \ge 2^l$ whenever $j_l \ne i_l$, $0 \le l \le p$.

Proof . We use induction. Since the case $p = 0$ is trivial, assume the validity of (1) and (2) at level $p - 1$.

(1) At level p, there are exactly $\prod_{k=0}^p m^p_k = 2^{2^p}$ symbols. Moreover, symbols are distinct if and only if the lemma is true. So suppose there are two symbols with distinct labels $(i_0, i_1, \ldots, i_p) \ne (j_0, j_1, \ldots, j_p)$ that both represent the same string $A^p_{i_0,i_1,\ldots,i_p} = A^p_{j_0,j_1,\ldots,j_p}$. Applying productions to this equality, we find that

$$A^{p-1}_{\lfloor i_1/m_1^{p-1}\rfloor,\ldots,\lfloor i_{p-1}/m_{p-1}^{p-1}\rfloor,i_p} = A^{p-1}_{\lfloor j_1/m_1^{p-1}\rfloor,\ldots,\lfloor j_{p-1}/m_{p-1}^{p-1}\rfloor,j_p}$$

and

$$A^{p-1}_{C_0^{p-1}(\lfloor i_1/m_1^{p-1}\rfloor,i_0 \bmod m_0^{p-1}),\ldots,C_{p-2}^{p-1}(\lfloor i_{p-1}/m_{p-1}^{p-1}\rfloor,i_{p-2} \bmod m_{p-2}^{p-1}),C_{p-1}^{p-1}(i_p,i_{p-1} \bmod m_{p-1}^{p-1})} = A^{p-1}_{C_0^{p-1}(\lfloor j_1/m_1^{p-1}\rfloor,j_0 \bmod m_0^{p-1}),\ldots,C_{p-2}^{p-1}(\lfloor j_{p-1}/m_{p-1}^{p-1}\rfloor,j_{p-2} \bmod m_{p-2}^{p-1}),C_{p-1}^{p-1}(j_p,j_{p-1} \bmod m_{p-1}^{p-1})}.$$

Since the level $p - 1$ symbols are necessarily distinct, the corresponding labels must satisfy

$$\lfloor i_k/m_k^{p-1}\rfloor = \lfloor j_k/m_k^{p-1}\rfloor$$

and

$$C_{k-1}^{p-1}(\lfloor i_k/m_k^{p-1}\rfloor, i_{k-1} \bmod m_{k-1}^{p-1}) = C_{k-1}^{p-1}(\lfloor j_k/m_k^{p-1}\rfloor, j_{k-1} \bmod m_{k-1}^{p-1})$$

for $1 \le k \le p$ (with the caveat $m_p^{p-1} = 1$). Whereupon,

$$i_{k-1} \bmod m_{k-1}^{p-1} = j_{k-1} \bmod m_{k-1}^{p-1}$$

by the completeness of the grammatical system: by property (3), $C_{k-1}^{p-1}(\lfloor i_k/m_k^{p-1}\rfloor, \cdot) = C_{k-1}^{p-1}(\lfloor j_k/m_k^{p-1}\rfloor, \cdot)$ is a one-to-one function. Therefore, for $0 \le k \le p$,

$$i_k = m_k^{p-1}\lfloor i_k/m_k^{p-1}\rfloor + i_k \bmod m_k^{p-1} = m_k^{p-1}\lfloor j_k/m_k^{p-1}\rfloor + j_k \bmod m_k^{p-1} = j_k,$$

which contradicts the distinct labels assumption.

(2) Suppose the labels $(i_0, i_1, \ldots, i_p)$ and $(j_0, j_1, \ldots, j_p)$ first differ in the lth index. Applying productions and the induction hypothesis, we deduce that:

$$\begin{aligned} d(A^p_{i_0,i_1,\ldots,i_p}, A^p_{j_0,j_1,\ldots,j_p}) &= d(A^{p-1}_{x_0,x_1,\ldots,x_{p-1}}, A^{p-1}_{y_0,y_1,\ldots,y_{p-1}}) + d(A^{p-1}_{z_0,z_1,\ldots,z_{p-1}}, A^{p-1}_{w_0,w_1,\ldots,w_{p-1}}) \\ &\geq 2^{l-1} + 2^{l-1} = 2^l. \end{aligned}$$

Clearly, the respective pairs of indices $(x_k = \lfloor i_{k+1}/m^{p-1}_{k+1} \rfloor, y_k = \lfloor j_{k+1}/m^{p-1}_{k+1} \rfloor)$ and $(z_k = C^{p-1}_k(x_k, i_k \bmod m^{p-1}_k), w_k = C^{p-1}_k(y_k, j_k \bmod m^{p-1}_k))$ can differ only if $k \geq l-1$. □

Given a complete grammatical system, we construct the corresponding family of iterated set squaring constructions by aggregating symbols in the manner of Section 3. For each $1 \leq \alpha \leq p$, the sets

$$A^p_{i_0,i_1,\ldots,i_{\alpha-1}} \triangleq \{A^p_{i_0,i_1,\ldots,i_p} | 0 \leq i_k \leq m^p_k - 1, \alpha \leq k \leq p\}$$

$(0 \leq i_k \leq m^p_k - 1, 0 \leq k \leq \alpha - 1)$ partition the space of all binary 2^p-tuples,

$$A^p \triangleq \{A^p_{i_0,i_1,\ldots,i_p} | 0 \leq i_k \leq m^p_k - 1, 0 \leq k \leq p\} = \mathbf{Z}_2^{2^p}.$$

Like Reed–Muller cosets, these codes inherit the iterative structure of the underlying grammatical system. Unioning over the indices $i_\alpha, i_{\alpha+1}, \ldots, i_p$ (using completeness property (3)), we find that

$$A^p_{i_0,i_1,\ldots,i_{\alpha-1}} = \bigcup_{x=0}^{m^{p-1}_{\alpha-1}-1} A^{p-1}_{C^{p-1}_0(x_0,z_0),\ldots,C^{p-1}_{\alpha-2}(x_{\alpha-2},z_{\alpha-2}),C^{p-1}_{\alpha-1}(x,z_{\alpha-1})} \times A^{p-1}_{x_0,\ldots,x_{\alpha-2},x}$$

for $1 \leq \alpha \leq p$, where $x_k = \lfloor i_{k+1}/m^{p-1}_{k+1} \rfloor$ and $z_k = i_k \bmod m^{p-1}_k$. Similarly,

$$A^p = A^{p-1} \times A^{p-1}.$$

Thus, the codes $\{A^p_{i_0,i_1,\ldots,i_{\alpha-1}} | 0 \leq i_k \leq m^p_k - 1, 0 \leq k \leq \alpha - 1\}$ are clearly iterated set squaring constructions generated by context-free grammars. Since these codes formally differ from Reed–Muller cosets only in the nature of their associated twisting permutations, they will be referred to as *bent* (i.e. unnaturally twisted) *Reed–Muller codes*. Their properties are

summarized in the following two propositions.

Proposition 3.4.1 For $1 \leq \alpha \leq p$:

1. The bent Reed–Muller code $A^p_{i_0,i_1,\ldots,i_{\alpha-1}}$ is a $(2^p, \prod_{s=0}^{p-\alpha} m_s^p)$ code of minimum distance $d(A^p_{i_0,i_1,\ldots,i_{\alpha-1}}) \geq 2^\alpha$.

2. The codes $\{A^p_{i_0,i_1,\ldots,i_{\alpha-1}} | 0 \leq i_k \leq m_k^p - 1, 0 \leq k \leq \alpha - 1\}$ partition $A^p = \mathbf{Z}_2^{2^p}$.

3. $d(A^p_{\underbrace{0,0,\ldots,0}_{\alpha}}) = 2^\alpha$ (including $\alpha = 0$).

Proof . (1) and (2) These statements follow immediately from lemma 3.4.1.

(3) We use induction. The claim is clearly true when $p = 0$ or $\alpha = 0$. Suppose it is valid at level $p-1$. Since the underlying grammatical system is complete, $A^p_{\underbrace{0,0,\ldots,0}_{\alpha}}$ is a true squaring construction:

$$A^p_{\underbrace{0,0,\ldots,0}_{\alpha}} = \bigcup_{i=0}^{m_{\alpha-1}^{p-1}-1} A^{p-1}_{\underbrace{0,\ldots,0,i}_{\alpha}} \times A^{p-1}_{\underbrace{0,\ldots,0,i}_{\alpha}}.$$

Therefore,

$$\begin{aligned} d(A^p_{\underbrace{0,0,\ldots,0}_{\alpha}}) &= \min[\min_i d(A^{p-1}_{\underbrace{0,\ldots,0,i}_{\alpha}}), 2d(A^{p-1}_{\underbrace{0,0,\ldots,0}_{\alpha-1}})] \\ &\leq 2d(A^{p-1}_{\underbrace{0,0,\ldots,0}_{\alpha-1}}) = 2^\alpha. \ \square \end{aligned}$$

The grammatical features of bent Reed–Muller codes are summarized in proposition 3.4.2.

Proposition 3.4.2 The bent Reed–Muller code $A^p_{i_0,i_1,\ldots,i_{\alpha-1}}$ is generated by a context-free

grammar with productions:

$$A^l_{j_0,j_1,\ldots,j_{K(l)}} \quad =$$

$$\begin{cases} \bigcup_{x=0}^{m_{\alpha-1}^{l-1}-1} A^{l-1}_{C_0^{l-1}(x_0,z_0),\ldots,C_{\alpha-2}^{l-1}(x_{\alpha-2},z_{\alpha-2}),C_{\alpha-1}^{l-1}(x,z_{\alpha-1})} \times A^{l-1}_{x_0,\ldots,x_{\alpha-2},x} & \alpha \le l \le p \\ A^{l-1}_{C_0^{l-1}(x_0,z_0),\ldots,C_{l-2}^{l-1}(x_{l-2},z_{l-2}),C_{l-1}^{l-1}(x_{l-1},z_{l-1})} \times A^{l-1}_{x_0,\ldots,x_{l-2},x_{l-1}} & 1 \le l \le \alpha - 1 \end{cases}$$

$$A^0_{j_0} \quad = \quad j_0$$

where

$$\begin{aligned} K(l) &\triangleq \min(l, \alpha - 1), \\ x_k &= \lfloor j_{k+1}/m_{k+1}^{l-1} \rfloor, \\ z_k &= j_k \bmod m_k^{l-1}, \end{aligned}$$

and $m_l^{l-1} \triangleq 1$ for $0 \le j_k \le m_k^l - 1$, $0 \le k \le K(l)$, and $1 \le l \le p$.

3.5 Counting States

In order to implement the maximum likelihood decoding algorithms of Chapter 2, we must reformulate Reed–Muller codes according to the grammatical template introduced in Section 2.1. In the process, we shall derive the important parameters $\{|I^{(l)}|, N^{(l)}|0 \le l \le p\}$ that determine (for a given code) the computational complexity of our decoding schemes. Emphasizing the primacy of the grammatical rather than the algebraic character of iterated squaring constructions, we analyze the more general case of bent Reed–Muller codes.

By proposition 3.4.2, the context-free grammar that generates the $(2^p, \prod_{s=0}^{p-\alpha} m_s^p, \ge 2^\alpha)$ bent Reed–Muller code $A^p_{i_0,i_1,\ldots,i_{\alpha-1}}$ contains symbols of the form $A^l_{j_0,j_1,\ldots,j_{K(l)}}$ $(K(l) = \min(l, \alpha - 1))$ at the levels $0 \le l \le p$. We put these grammatical symbols in one-to-one correspondence with the set of states $\{0 \le j \le M^{(l)} - 1\}$, $M^{(l)} = \prod_{k=0}^{K} m_k^l$, according to the rule $A^l_{j_0,j_1,\ldots,j_K} \leftrightarrow j$, where

$$j = \sum_{k=0}^{K} j_k \prod_{i=k+1}^{K} m_i^l$$

and

$$j_k = \lfloor j / \prod_{i=k+1}^{K} m_i^l \rfloor \bmod m_k^l$$

(with the understanding that $\prod_{i=K+1}^{K} m_i^l = 1$). Within the general framework of this state-symbol correspondence, we can count the productions and states generated from a variety of different start symbols [5].

Lemma 3.5.1 The CFG associated with the start symbol $A^p_{i_0,i_1,\ldots,i_{\alpha-1}}$ $(1 \le \alpha \le p)$ has $|I^{(l)}|$ productions from any level l state, satisfying:

1.

$$|I^{(l)}| = \begin{cases} m_{\alpha-1}^{l-1} & \alpha \le l \le p \\ 1 & 1 \le l \le \alpha - 1 \\ 0 & l = 0. \end{cases}$$

2. For $0 \le l \le p$, $|I^{(l)}| \le m_{\alpha-1}^{p-1}$.

Proof . (1) This statement follows immediately from proposition 3.4.2. (2) $\{m_k^l | k \le l \le \infty\}$ is monotone in l.

Lemma 3.5.2 Consider the CFG associated with the start symbol $A^p_{\underbrace{0,\ldots,0,i}_{\alpha}}$ $(0 \le i \le m_{\alpha-1}^p - 1,\ 1 \le \alpha \le p)$.

1. The set of grammatically allowed symbols appearing at level $0 \le l \le p$ consists of symbols of the form $A^l_{j_0,j_1,\ldots,j_{K(l)}}$, labeled by the rightmost $K(l)+1$ indices from each of the sequences in the set

   $$\{\ldots,0,0,\ldots,0,i^{(l)},j_{l+\alpha-p},j_{l+\alpha-p+1},\ldots,j_{K(l)} | 0 \le j_k \le m_k^l - 1, l+\alpha-p \le k \le K(l)\},$$

   where

   $$\begin{aligned} K(l) &= \min(l, \alpha - 1), \\ i^{(l)} &= \lfloor i / \prod_{k=l}^{p-1} m_{p-\alpha}^k \rfloor, \end{aligned}$$

and

$$m_k^l \;=\; m_{l-k}^l = \begin{cases} 2^{\binom{l}{k}} & 0 \le k \le l \\ 1 & k < 0, k > l. \end{cases}$$

2. At level $0 \le l \le p$, the state space can be expressed as $\{0, \ldots, N^{(l)} - 1\}$, where the number of states is

$$N^{(l)} \;=\; \begin{cases} 1 & l = p \\ \prod_{k=\max(0,l+\alpha-p)}^{K(l)} m_k^l & 0 \le l \le p-1. \end{cases}$$

3. Moreover, $N^{(l)} \le m_{\alpha-1}^{p-1}$, for $0 \le l \le p$.

Proof . (1) By proposition 3.4.2, allowed symbols at level l are of the general form $A^l_{j_0,j_1,\ldots,j_{K(l)}}$, labeled by $K(l)+1$ indices. However, not all $\prod_{k=0}^{K(l)} m_k^l$ such symbols are derivable from the given start symbol. Indeed, a symbol is grammatically allowed at level l if and only if it is the left or right element of a production from an allowed level $l+1$ symbol.

We begin by slightly modifying our system of grammatical labels. For convenience, the level l symbol $A^l_{j_0,j_1,\ldots,j_{K(l)}}$ will be represented by the semiinfinite sequence $\ldots, j_{-1}, j_0, \ldots, j_{K(l)}$ with $0 \le j_k \le m_k^l - 1$ for $k \le K(l)$. Productions generalize accordingly: when k is negative, define the spurious twisting permutations $C_k^l(i,j) = 0$, $0 \le i, j \le m_k^l - 1 = 0$.

We proceed by induction. Clearly, the case $l = p$ is a degenerate case of the lemma with $i^{(l)} = i$ and no subsequent variable indices j_k—the resulting set of sequences is a singleton. Now suppose the lemma holds at level l. Consider first the case of single productions, $1 \le l \le \alpha - 1$. The labels of all allowed right and left produced symbols are respectively

$$\{\ldots, 0, 0, \ldots, 0, \lfloor i^{(l)}/m_{l+\alpha-p-1}^{l-1} \rfloor, \lfloor j_{l+\alpha-p}/m_{l+\alpha-p}^{l-1} \rfloor, \ldots, \lfloor j_{K(l)}/m_{K(l)}^{l-1} \rfloor |$$
$$0 \le j_k \le m_k^l - 1, l+\alpha-p \le k \le K(l)\}$$

and (suppressing C's subscripts and superscripts)

$$\{\ldots, 0, 0, \ldots, 0, \lfloor i^{(l)}/m_{l+\alpha-p-1}^{l-1} \rfloor, C(\lfloor j_{l+\alpha-p}/m_{l+\alpha-p}^{l-1} \rfloor, i^{(l)} \bmod m_{l+\alpha-p-1}^{l-1}), \ldots,$$
$$C(\lfloor j_{K(l)}/m_{K(l)}^{l-1} \rfloor, j_{K(l)-1} \bmod m_{K(l)-1}^{l-1}) | 0 \le j_k \le m_k^l - 1, l+\alpha-p \le k \le K(l)\}.$$

In contrast, for the case of multiple productions $\alpha \leq l \leq p$, these right and left produced sequences are augmented on the right by the respective single indices j and $C(j, j_{K(l)} \bmod m_{K(l)}^{l-1})$ with the variable j assuming the values $0 \leq j \leq m_{\alpha-1}^{l-1} - 1$. In either case, the set of right or left produced labels simplifies, becoming (for some n)

$$\{\ldots, 0, 0, \ldots, 0, \lfloor i^{(l)}/m_{l+\alpha-p-1}^{l-1} \rfloor, i_n, \ldots, i_{K(l-1)} | 0 \leq i_k \leq m_k^{l-1} - 1, n \leq k \leq K(l-1)\}.$$

This occurs because $m_k^l = m_k^{l-1} m_{k-1}^{l-1}$ and each function C is a permutation. Two final observations complete the proof. First, $n = l + \alpha - p - 1$. In the single production case, the number of variable indices is unchanged and they are left-shifted by one position, whereas in the multiple production case, the number of variable indices increases by one and they are not left-shifted. Second, if the expression $i^{(l-1)} = \lfloor i^{(l)}/m_{l+\alpha-p-1}^{l-1} \rfloor$ is iterated (recalling that $m_k^l = m_{l-k}^l$), the sequences' leading nontrivial index is seen to have the correct form.

(2) $N^{(l)}$ is the cardinality of the set of grammatically allowed symbols at level l. Examining the set of allowed labels in (1), we see that all indices preceding $j_{l+\alpha-p}$ are fixed; moreover, if $l + \alpha - p \leq 0$, the variable indices $\{j_k | l + \alpha - p \leq k < 0\}$ are identically zero. Therefore, the only nontrivial variable indices are $\{j_k | \max(0, l + \alpha - p) \leq k \leq K(l)\}$ each of which ranges over exactly m_k^l values. In addition, the grammar's level l state space can be expressed as $\{0, \ldots, N^{(l)} - 1\}$. Simply use the above state-symbol correspondence (while simultaneously suppressing or restoring the leading index $i^{(l)}$ if it is nonzero).

(3) Clearly, $N^{(p)} = 1$ and $N^{(p-1)} = m_{\alpha-1}^{p-1}$ satisfy the inequality. Since $m_k^l = 1$ for negative k, we can write,

$$N^{(l)} = \prod_{k=l+\alpha-p}^{K(l)} m_k^l$$

for $0 \leq l \leq p - 1$. If $\alpha - 1 \leq l \leq p - 2$,

$$\begin{aligned} N^{(l)} = \prod_{k=l+\alpha-p}^{\alpha-1} m_k^l &\leq (\prod_{k=l+\alpha-p}^{\alpha-3} m_{k+1}^{l+1}) m_{\alpha-2}^l m_{\alpha-1}^l \\ &= (\prod_{k=l+1+\alpha-p}^{\alpha-2} m_k^{l+1}) m_{\alpha-1}^{l+1} = N^{(l+1)}. \end{aligned}$$

This inequality follows from the relations $m_{k+1}^{l+1} = m_{k+1}^l m_k^l$ and $m_k^l \leq m_{k+1}^{l+1}$. Similarly, if

$0 \leq l \leq \alpha - 2$,

$$N^{(l)} = \prod_{k=l+\alpha-p}^{l} m_k^l \quad \leq \quad \prod_{k=l+\alpha-p}^{l} m_{k+1}^{l+1} = N^{(l+1)}.$$

Thus, $N^{(l)}$ increases monotonically from 2 at $(l = 0)$ to $m_{\alpha-1}^{p-1}$ (at $l = p - 1$). □

The parameters $\{|I^{(l)}|, N^{(l)} | 0 \leq l \leq p\}$ derived in the previous two lemmas solely determine the computational complexity of our maximum likelihood decoding algorithms for the code $RM(p - \alpha, p)$. For the memoryless communications channel case, table 3.1 tabulates the number of operations (2.2.1) required to decode the nontrivial Reed–Muller codes of intermediate length. Although the loose upper bound $O(RM(p-\alpha, p)) \ll 2^p (m_{\alpha-1}^{p-1})^2$ is overly pessimistic, it does suggest that the logarithm of the decoding complexity is a polynomial function of the code's log-length p. This hypothesis is effectively substantiated by the data presented in table 3.1. Since each million real number operations requires roughly half a second of real-time computation (on an average workstation), we observe that RM(3,7) and almost all the Reed–Muller codes of length 256 or greater are practically undecodable—even by dynamic programming methods.

		α						
		1	2	3	4	5	6	7
p	4	87	255	127				
	5	183	1,215	3,007	319			
	6	375	5,375	327,039	79,231	767		
	7	759	22,783	151,116,543	4,425,388,799	4,606,719	1,791	
	8	1,527	94,207	292,116,477,439	$> 2^{57}$	$> 2^{57}$	562,599,423	4,095

Table 3.1: Operations to decode $RM(p - \alpha, p)$.

Although many Reed–Muller grammars may be too complex for practical decoding, perhaps there exist sub-grammars with more strictly limited state spaces. The following lemma examines a class of context-free grammars generated by certain restricted subsets of Reed–Muller symbols [5].

Lemma 3.5.3 Given the nonnegative integers l^* and n satisfying $m_{\alpha-1}^{l^*} \leq M = 2^n \leq m_{\alpha-1}^{l^*+1}$, consider a CFG that uses the symbols $\{A^{l^*+1}_{\underbrace{0,\ldots,0}_{\alpha},i} | 0 \leq i \leq M-1\}$ at level $l^* + 1$.

1. At level $0 \leq l \leq l^* + 1$, the above symbol-state correspondence yields the state space

$\{0, \ldots, N^{(l)} - 1\}$ of size

$$N^{(l)} = \begin{cases} M & l = l^* + 1 \\ \max(1, \lfloor M / \prod_{k=l}^{l^*} m_{l^*+1-\alpha}^k \rfloor) \prod_{k=\max(0,l+\alpha-l^*-1)}^{K(l)} m_k^l & 0 \le l \le l^*. \end{cases}$$

2. For $0 \le l \le l^* + 1$, $N^{(l)} \le M$ and $|I^{(l)}| \le M$.

Proof . (1) With a single exception, this follows directly from statements 1 and 2 of lemma 3.5.2. Instead of being fixed for each level l, the index $i^{(l)}$ assumes

$$\lfloor (M-1) / \prod_{k=l}^{l^*} m_{l^*+1-\alpha}^k \rfloor + 1 = \max(1, \lfloor M / \prod_{k=l}^{l^*} m_{l^*+1-\alpha}^k \rfloor)$$

values. (Suppose a and b are powers of 2. If b divides a, then $\lfloor (a-1)/b \rfloor = \lfloor a/b \rfloor - 1 \ge 0$; otherwise $b > a$ and $\lfloor (a-1)/b \rfloor = 0$.)

(2) That $|I^{(l)}| \le M$ follows from lemma 3.5.1. By lemma 3.5.2, $N^{(l)}$ is monotonically increasing in the range $0 \le l \le l^*$. Thus,

$$N^{(l)} \le N^{(l^*)} = \lfloor M / m_{\alpha-1}^{l^*} \rfloor m_{\alpha-1}^{l^*} \le M$$

for $0 \le l \le l^*$. □

3.6 Thinned Reed–Muller Grammars

As demonstrated in Section 5, the state spaces of most Reed–Muller codes of length 256 or greater are too large for practical decoding—even by dynamic programming. Fortunately, however, there exists an enormous family of subgroups of Reed–Muller codes that are readily decodable by the exact maximum likelihood decoding algorithms of Chapter 2. Such codes are constructed by systematically *thinning* the Reed–Muller grammars of Section 3, discarding extraneous productions (or information bits) and thereby strictly limiting the cardinality of each level's state space [5].

Definition 3.6.1 Given $1 \le \alpha \le p$ and $n \ge 0$, the *thinned Reed–Muller code* $RM^{(n)}(p - \alpha, p)$ is the length 2^p code generated by the following CFG with start symbol $B^{p,n}_{\underbrace{0,0,\ldots,0}_{\alpha}}$

and productions:

$$B^{l,n}_{j_0,j_1,\ldots,j_{K(l)}} = \begin{cases} \bigcup_{x=0}^{\min(2^n,m^{l-1}_{\alpha-1})-1} B^{l-1,n}_{f(x_0,z_0),\ldots,f(x_{\alpha-2},z_{\alpha-2}),f(x,z_{\alpha-1})} \times B^{l-1,n}_{x_0,\ldots,x_{\alpha-2},x} & \alpha \le l \le p \\ B^{l-1,n}_{f(x_0,z_0),\ldots,f(x_{l-2},z_{l-2}),f(x_{l-1},z_{l-1})} \times B^{l-1,n}_{x_0,\ldots,x_{l-2},x_{l-1}} & 1 \le l \le \alpha-1 \end{cases}$$

$$B^{0,n}_{j_0} = j_0$$

where

$$\begin{aligned} K(l) &= \min(l, \alpha-1), \\ x_k &= \lfloor j_{k+1}/m^{l-1}_{k+1} \rfloor, \\ z_k &= j_k \bmod m^{l-1}_k, \end{aligned}$$

and

$$m^l_k = m^l_{l-k} \begin{cases} 2^{\binom{l}{k}} & 0 \le k \le l \\ 1 & \text{otherwise} \end{cases}$$

for $0 \le j_k \le m^l_k - 1$, $0 \le k \le K(l)$, and $1 \le l \le p$.

The key properties of thinned Reed–Muller codes are presented in the following proposition. Note that *thinned bent Reed–Muller codes* (defined as above with the substitutions $f \to C^l_k$) share all of these features except linearity.

Proposition 3.6.1 Consider the thinned Reed–Muller code $RM^{(n)}(p-\alpha, p)$ for $1 \le \alpha \le p$ and $n \ge 0$. Define the particular level:

$$l^* = \max\{0 \le l \le p-1 | m^l_{\alpha-1} \le 2^n\}.$$

1. If $n \ge \binom{p-1}{\alpha-1}$, then $l^* = p-1$ and $RM^{(n)}(p-\alpha,p) = RM(p-\alpha,p)$; otherwise, $0 \le l^* \le p-2$ and $RM^{(n)}(p-\alpha,p) \subset RM(p-\alpha,p)$.

2. $RM^{(n)}(p-\alpha,p)$ is a linear $[2^p, \sum_{l=1}^p 2^{p-l}Q^{(l)}, 2^\alpha]$ code with grammatical parameters:

$$|I^{(l)}| = 2^{Q^{(l)}} = \begin{cases} \min(2^n, m^{l-1}_{\alpha-1}) & \alpha \le l \le p \\ 1 & 1 \le l \le \alpha-1 \\ 0 & l = 0 \end{cases}$$

and

$$N^{(l)} = \begin{cases} 1 & l = p \\ 2^n & l^* + 1 \le l \le p-1 \\ \max(1, \lfloor N^{l^*+1} / \prod_{k=l}^{l^*} m_{l^*+1-\alpha}^k \rfloor) \prod_{k=\max(0,l+\alpha-l^*-1)}^{K(l)} m_k^l & 0 \le l \le l^*. \end{cases}$$

The state-symbol correspondence of Section 5 produces a state space of the form $\{0, \ldots, N^{(l)} - 1\}$.

3. For $0 \le l \le p$, the number of states and productions is bounded: $N^{(l)} \le 2^n$ and $|I^{(l)}| \le 2^n$.

Proof . (1) If $n \ge \binom{p-1}{\alpha-1}$, then $2^n \ge m_{\alpha-1}^l$ and $\min(2^n, m_{\alpha-1}^l) = m_{\alpha-1}^l$ for each $0 \le l \le p-1$; the thinned Reed–Muller grammar coincides with the original Reed–Muller grammar of proposition 3.3.2. Otherwise, $2^n < m_{\alpha-1}^{p-1}$ and

$$RM^{(n)}(p-\alpha, p) = \bigcup_{i=0}^{2^n-1} B^{p-1,n}_{\underbrace{0,\ldots,0,i}_{\alpha}} \times B^{p-1,n}_{\underbrace{0,\ldots,0,i}_{\alpha}} \subset \bigcup_{i=0}^{2^n-1} B^{p-1}_{\underbrace{0,\ldots,0,i}_{\alpha}} \times B^{p-1}_{\underbrace{0,\ldots,0,i}_{\alpha}} \subset RM(p-\alpha, p).$$

By induction, the sets associated with thinned symbols are subsets of those sets associated with the original symbols of the same label.

(2) The case $l^* = p - 1$ is considered in lemmas 3.5.1 and 3.5.2. Now suppose $0 \le l^* \le p-2$. By the definition of l^*, $m_{\alpha-1}^l > 2^n$ for $l > l^*$. Thus, if $l - 1 \ge l^* + 1$, the productions of the thinned Reed–Muller grammar assume the form

$$B^{l,n}_{\underbrace{0,\ldots,0,j}_{\alpha}} = \bigcup_{i=0}^{2^n-1} B^{l-1,n}_{\underbrace{0,\ldots,0,f(i,j)}_{\alpha}} \times B^{l-1,n}_{\underbrace{0,\ldots,0,i}_{\alpha}}$$

for $0 \le j \le N^{(l)} - 1$. If $N^{(l)} \le 2^n$, then $\lfloor j / m_{\alpha-1}^{l-1} \rfloor = 0 \ \forall j$ implies productions of this form and $N^{(l-1)} = 2^n$; or $N^{(l)} = 2^n$ for $l^* + 1 \le l \le p - 2$. In contrast, at and below level $l^* + 1$, the thinned Reed–Muller grammar coincides with the original Reed–Muller grammar. The remainder of the formula for $N^{(l)}$ follows directly from lemma 3.5.3.

Both the linearity and minimum distance of $RM^{(n)}(p-\alpha, p)$ are deduced by induction. By proposition 3.3.2, the code $B^{l^*+1,n}_{\underbrace{0,\ldots,0,0}_{\alpha}} = B^{l^*+1}_{\underbrace{0,\ldots,0,0}_{\alpha}}$ is linear with minimum distance 2^α. Moreover, since the operation f is mod-2 vector addition of binary strings,

$\bigcup_{i=0}^{2^n-1} B^{l^*+1,n}_{\underbrace{0,\ldots,0,i}_{\alpha}}$ is linear. Therefore, because $B^{l^*+2,n}_{\underbrace{0,\ldots,0,0}_{\alpha}}$ is the group squaring construction $|\bigcup_{i=0}^{2^n-1} B^{l^*+1,n}_{\underbrace{0,\ldots,0,i}_{\alpha}}/B^{l^*+1,n}_{\underbrace{0,\ldots,0,0}_{\alpha}}|^2$, it too is linear with minimum distance 2^α. By the distance properties of the squaring construction, $d(B^{l^*+2,n}_{\underbrace{0,\ldots,0,0}_{\alpha}}) \leq d(B^{l^*+1,n}_{\underbrace{0,\ldots,0,0}_{\alpha}}) = 2^\alpha$; however, since $B^{l^*+2,n}_{\underbrace{0,\ldots,0,0}_{\alpha}} \subset B^{l^*+2}_{\underbrace{0,\ldots,0,0}_{\alpha}}$, $d(B^{l^*+2,n}_{\underbrace{0,\ldots,0,0}_{\alpha}}) \geq d(B^{l^*+2}_{\underbrace{0,\ldots,0,0}_{\alpha}}) = 2^\alpha$. In addition, $\bigcup_{i=0}^{2^n-1} B^{l^*+2,n}_{\underbrace{0,\ldots,0,i}_{\alpha}} = (\bigcup_{i=0}^{2^n-1} B^{l^*+1,n}_{\underbrace{0,\ldots,0,i}_{\alpha}})^2$ is linear. Clearly, this argument can be iterated for the successive levels $l^* + 3 \leq l \leq p$.

(3) This statement follows from (2) and lemma 3.5.3. □

As a consequence of lemma 3.6.1, the decoding complexity of thinned Reed–Muller codes is strictly controlled by the limiting parameter n. The corresponding loose upper bound on the number of decoding operations is $O(RM^{(n)}(p-\alpha, p)) \ll 2^{p+2n}$, which for fixed n is simply a multiple of the code length 2^p. For example, $RM^{(8)}(6, 10)$, a linear $[1024, 440, 16]$ code roughly half the size of the undecodable linear $[1024, 848, 16]$ code RM(6,10), is decodable in $2^{21} \ll 2^{26}$ real number operations. Appendix A assembles the results of simulated decoding trials for six representative thinned Reed–Muller codes. We discuss these results in Chapter 5 after developing an alternative coarse-to-fine decoding algorithm in Chapter 4.

Chapter 4

A CTFDP Algorithm for Maximum Likelihood Decoding

4.1 Coarse-to-Fine Dynamic Programming

In Section 2.2, we described the standard dynamic programming (DP) approach for minimizing a bit-wise additive cost function over a CFG representable code. One iteratively computes a hierarchy of induced costs, assigning to each state at each node of the code's tree-template the minimum sum of the costs associated with the given state's productions. However, for many codes with large states spaces, the computational cost of standard dynamic programming exceeds practical bounds. Therefore, in this section we consider a variant of the standard approach called *coarse-to-fine dynamic programming* (CTFDP) in which the original DP problem is replaced by an approximating sequence of simpler DP problems ([8], [9]).

Consider a code $\mathcal{C}$ formulated according to the grammatical template of Section 2.1. To implement CTFDP, we begin by constructing a family of codes that in some sense approximate $\mathcal{C}$. First, at each level of the tree-template, we select a sequence of successively finer partitions of the state space $\{0, 1, \ldots, N^{(l)} - 1\}$ beginning with the state space itself and ending with its singlet decomposition. A subset in one of these partition chains, being an aggregate of states, is called a *super-state*. The *coarseness* of a super-state specifies the particular partition to which it belongs. Moreover, *refinement* is the act of splitting a super-state into (typically two) super-states at the adjacent level of coarseness. Second, for any given set of state space partitions, we define generalized super-state productions. If A,B, and C are allowed super-states, respectively occupying a node and its left and right daughter nodes in the tree-template, then $A \rightarrow BC$ is an allowed *super-state production* if there exist states $x \,\epsilon\, A$, $y \,\epsilon\, B$, and $z \,\epsilon\, C$ such that $x \rightarrow yz$ is an allowed state production.

With this choice of super-state productions, any given set of state space partitions—possibly consisting of super-states of varying degrees of coarseness and possibly differing from node to node at any level—uniquely determines a *super-state grammar* corresponding to a *super-code* containing the original code $\mathcal{C}$. To ensure that a super-state grammar generates *bit*-strings (of length equal to that of $\mathcal{C}$), we define the following terminal productions from level 0 super-states: $\{0\} \rightarrow 0$, $\{1\} \rightarrow 1$, and $\{0, 1\} \rightarrow 0|1$. Furthermore, any codeword in $\mathcal{C}$ can be derived from a super-state grammar; given the codeword $\mathbf{c} \,\epsilon\, \mathcal{C}$, there exists a super-state derivation tree that corresponds (by the definition of super-state productions) to the codeword's own state derivation tree and has the bits of $\mathbf{c}$ as its terminal assignments.

Thus, each super-state grammar generates a super-code containing $\mathcal{C}$.

The heart of the CTFDP minimization algorithm involves a series of standard DP computations over successively finer super-state grammars. We begin by applying the standard DP algorithm to the coarsest possible super-state grammar, the grammar consisting of a single super-state (the state space itself) at each node of the tree-template. Of course, the nodal costs of the level-0 super-states are initialized by 2^p independent minimizations over the nodal costs of the states 0 and 1. The output of this first DP step is merely the minimum cost string in the super-code $\mathbf{Z}_2^{2^p}$.

The CTFDP algorithm proceeds by progressively refining the super-state grammar. Given the solution of the the previous DP problem—an *optimal derivation tree* corresponding to a minimum cost super-codeword, we determine whether the optimal derivation tree contains any non-singlet super-states. If so, we refine these super-states, recompute the super-state productions, solve the new DP problem, and again examine the optimal derivation tree. If not, we stop: the current optimal derivation tree represents the minimum cost codeword. Since the terminal optimal derivation tree contains only states from $\mathcal{C}$'s underlying grammar, it certainly generates a codeword (in $\mathcal{C}$). Moreover, this codeword is by definition the minimum cost super-codeword in the terminal super-code—a code that contains $\mathcal{C}$ itself.

Although this CTFDP algorithm must eventually produce the solution to a given minimization problem, it need not necessarily outperform standard DP. For the procedure to converge rapidly, the number of refinements and subsequent DP computations must be minimal. This suggests that super-states should consist of aggregations of "similar" states so that their costs more closely reflect those of their constituents [9]. In addition, the determination of super-state productions must not be too computationally demanding. These considerations thus prompt the following question: is there an efficient CTFDP implementation of maximum likelihood decoding for the grammatical codes of Chapter 3?

4.2 Super-States for Thinned Reed–Muller Codes

To implement a CTFDP version of a given DP problem, one must first partition the problem's state spaces into clusters of super-states. In this section, we construct coarsened grammatical symbols and associated super-states for thinned Reed–Muller codes. The fact that

these coarsened symbols exhibit considerable grammatical structure suggests that thinned Reed–Muller codes are particularly amenable to coarse-to-fine decoding.

We begin by reformulating thinned Reed–Muller grammars according to the grammatical template introduced Section 2.1. This approach deliberately blurs the distinction between grammatical symbols and their associated numerical states. By proposition 3.6.1, the set of grammatically allowed symbols at level l for a given code $RM^{(n)}(p-\alpha,p)$ can be reexpressed as $\{B_i^{l,n}|0 \leq i \leq N^{(l)}-1\}$, where the integer label i denotes the state corresponding to an allowed symbol $B^{l,n}_{i_0,i_1,\ldots,i_{K(l)}}$. The associated grammatical productions are presented in the following lemma.

Lemma 4.2.1 Consider the thinned Reed–Muller code $RM^{(n)}(p-\alpha,p)$ for $1 \leq \alpha \leq p$ and $n \geq 0$.

1. The set of allowed productions from the state $0 \leq i \leq N^{(l)}-1$ at level $1 \leq l \leq p$ is

$$I_i^{(l)} = \{(f(x(i) \wedge j, z(i)), x(i) \wedge j)|0 \leq j \leq |I^{(l)}|-1\}$$

where

$$|I^{(l)}| = \min(2^n, m_{K(l)}^{l-1})$$

and the binary operator $\wedge$ (introduced solely for notational convenience) equals f (i.e. $a \wedge b = f(a,b)$). The auxiliary integers $x(i)$ and $z(i)$ are defined by the relations

$$x(i) \leftrightarrow \mathbf{x}(i) = \mathbf{x}_0\mathbf{x}_1\cdots\mathbf{x}_{K(l)}$$

and

$$z(i) \leftrightarrow \mathbf{z}(i) = \mathbf{z}_0\mathbf{z}_1\cdots\mathbf{z}_{K(l)}.$$

In these expressions, $\mathbf{x}(i)$ ($\mathbf{z}(i)$) is the binary expansion of the integer $x(i)$ ($z(i)$); $\mathbf{z}_k$ is the $\binom{l-1}{k}$-bit binary expansion of the integer $z_k = i_k \bmod m_k^{l-1}$; and $\mathbf{x}_k$ is the $\binom{l-1}{k}$-bit binary expansion of the integer

$$x_k = \begin{cases} \lfloor i_{k+1}/m_{k+1}^{l-1} \rfloor & 1 \leq k \leq K(l)-1 \\ 0 & k = K(l). \end{cases}$$

(Note that for single productions (i.e. $K(l) = l$), the strings $\mathbf{x}_{K(l)}$ and $\mathbf{z}_{K(l)}$ have

length $\binom{l-1}{l} \triangleq 0$ and can therefore be ignored; however, for multiple productions, they are *not* negligible.)

2. The corresponding symbolic productions are:

$$B_i^{l,n} = \bigcup_{j=0}^{|I^{(l)}|-1} B_{f(x(i)\wedge j, z(i))}^{l-1,n} \times B_{x(i)\wedge j}^{l-1,n}.$$

Proof . (1) and (2). The state symbol correspondence, originally defined in Section 3.5, is more intuitively described by the relation $i \leftrightarrow B_{i_0,i_1,\ldots,i_{K(l)}}^{l,n}$ if and only if the binary expansion $\mathbf{i}$ of the integer i equals the concatenation $\mathbf{i}_0\mathbf{i}_1\cdots\mathbf{i}_{K(l)}$ (where $\mathbf{i}_k$ is the $\binom{l}{k}$-bit binary expansion of the integer i_k). Now observe that for single productions (i.e. $K(l) = l$ and $|I^{(l)}| = m_l^{l-1} = 1$), $x(i) \wedge j \leftrightarrow \mathbf{x}_0\mathbf{x}_1\cdots\mathbf{x}_{l-1}$ and $z(i) \leftrightarrow \mathbf{z}_0\mathbf{z}_1\cdots\mathbf{z}_{l-1}$, whereas for multiple productions (i.e. $K(l) = \alpha - 1$ and $1 < |I^{(l)}| \leq m_{\alpha-1}^{l-1}$), $x(i) \wedge j \leftrightarrow \mathbf{x}_0\mathbf{x}_1\cdots\mathbf{x}_{\alpha-2}\mathbf{j}$ and $z(i) \leftrightarrow \mathbf{z}_0\mathbf{z}_1\cdots\mathbf{z}_{\alpha-1}$. (Note that for each k, the strings $\mathbf{x}_k$ and $\mathbf{z}_k$ have length $\binom{l-1}{k}$; and for consistency the binary expansion $\mathbf{j}$ of the integer j must share the length of $\mathbf{x}_{\alpha-1}$ (and $\mathbf{z}_{\alpha-1}$).) Moreover, since the function f performs mod-2 vector addition on the binary expansions of its arguments, its action is *separable* in the following sense:

$$f(a,b) \quad \leftrightarrow \quad \mathbf{f}(a_0,b_0)\mathbf{f}(a_1,b_1)\cdots\mathbf{f}(a_K,b_K) \tag{4.2.1}$$

whenever $a \leftrightarrow \mathbf{a}_0\mathbf{a}_1\cdots\mathbf{a}_K$, $b \leftrightarrow \mathbf{b}_0\mathbf{b}_1\cdots\mathbf{b}_K$, and the substrings $\mathbf{a}_k$ and $\mathbf{b}_k$ share the same length. Of course, in this expression, consistency demands that the length of the substring $\mathbf{f}(a_k,b_k)$, the binary expansion of $f(a_k,b_k)$, equal the given length of the substrings $\mathbf{a}_k$ and $\mathbf{b}_k$ associated with its arguments. Thus, since the function f is separable, lemma 4.2.1 reproduces both the single *and* multiple productions of definition 3.6.1. □

Lemma 4.2.1 has two important practical consequences. First, the multitude of productions for the thinned Reed–Muller code $RM^{(n)}(p-\alpha,p)$ can be readily computed from a comparatively small set of stored integers—the $2(p+1)$ parameters $\{N^{(l)}, |I^{(l)}| | 0 \leq l \leq p\}$ and the set of $2\sum_{l=1}^{p} N^{(l)} \ll p2^{n+1}$ auxiliary x's and z's, one pair of integers for each state at each non-zero level. Second, as we now demonstrate, there exists a hierarchy of coarsened symbols that retains the essential structure of the original thinned Reed–Muller grammar.

Definition 4.2.1 Consider the thinned Reed–Muller code $RM^{(n)}(p-\alpha,p)$ for $1 \leq \alpha \leq p$

and $n \geq 0$. Fixing the level $0 \leq l \leq p$, *coarseness* $0 \leq q \leq \log_2 N^{(l)}$, and label $0 \leq i \leq N^{(l,q)} - 1$ (with $N^{(l,q)} \triangleq \lfloor N^{(l)}/2^q \rfloor$), define the *coarse symbol*

$$B_i^{l,n,q} \triangleq \bigcup_{j=0}^{2^q-1} B_{(i<<q)\wedge j}^{l,n} = \{B_k^{l,n} | \lfloor k/2^q \rfloor = i\}.$$

The corresponding level l *super-state* is denoted by the pair (q, i). (A comment on notation: the coarsening or refinement of a grammatical symbol is most intuitively represented by respectively right- or left-shifting its integer label. The left-shift operation $i << q$, which shifts the binary representation of i q bits to the left and inserts q zeros, is equivalent to multiplication by 2^q; similarly, the right-shift operation $i >> q$, which shifts the binary representation of i q bits to the right, is equivalent to $\lfloor i/2^q \rfloor$.)

The recursive structure of this hierarchy of coarsened symbols is analyzed in the following proposition. Notice that the *uniform coarseness productions* defined by equation 4.2.2 below retain the simple Reed–Muller structure and are easily computed from the set of stored auxiliary x's and z's that underpin the original grammar.

Proposition 4.2.1 The coarse symbols $\{B_i^{l,n,q} | 0 \leq i \leq N^{(l,q)} - 1, 0 \leq q \leq \log_2 N^{(l)}, 1 \leq l \leq p\}$ for the thinned Reed–Muller code $RM^{(n)}(p - \alpha, p)$ (with $1 \leq \alpha \leq p$ and $n \geq 0$) satisfy the recursive set identity:

$$B_i^{l,n,q} = \bigcup_{j=0}^{I^{(l,q)}-1} B_{f(\overline{x}\wedge j,\overline{z})}^{l-1,n,\overline{q}} \times B_{\overline{x}\wedge j}^{l-1,n,\overline{q}} \tag{4.2.2}$$

where the parameters

$$\begin{aligned} I^{(l,q)} &= 2^{q_x - q_z}, \\ \overline{q} &= q' + q_z, \\ \overline{z} &= z(i << q) >> (q' + q_z), \end{aligned}$$

and

$$\overline{x} = [x(i << q) >> (q' + q_x)] << (q_x - q_z)$$

are defined in terms of the quantities

$$q' = \begin{cases} 0 & k^* = K(l)+1 \\ \min(n, \sum_{k=k^*}^{K(l)} \binom{l-1}{k}) & 0 \le k^* \le K(l), \end{cases}$$

$$q_z = \begin{cases} q & k^* = K(l)+1 \\ \max(0, q - r_{k^*} - \binom{l-1}{k^*-1}) & 0 \le k^* \le K(l), \end{cases}$$

and

$$q_x = \begin{cases} \min(n, \binom{l-1}{K(l)}) & k^* = K(l)+1 \\ \min(q - r_{k^*}, \binom{l-1}{k^*-1}) & 0 \le k^* \le K(l). \end{cases}$$

Given the array

$$r_k = \begin{cases} 0 & k = K(l)+1 \\ \min(n, \binom{l-1}{K(l)}) & k = K(l) \\ r_{k+1} + 2\binom{l-1}{k} & 0 \le k \le K(l)-1, \end{cases}$$

the integer k^* is defined by the expression

$$k^* = \min\{0 \le k \le K(l)+1 | q \ge r_k\}.$$

(Recall that we define $\binom{l-1}{l} \stackrel{\Delta}{=} 0$.)

Proof . Applying the productions of lemma 4.2.1 to the coarse symbols of definition 4.2.1, we find that

$$\begin{aligned} B_i^{l,n,q} &= \bigcup_{j=0}^{2^q-1} B_{(i<<q)\wedge j}^{l,n} \\ &= \bigcup B_{f(x,z)}^{l-1,n} \times B_x^{l-1,n} \end{aligned} \tag{4.2.3}$$

where the integers $(i << q) \wedge j$, z, and x have the binary expansions:

$$\begin{aligned} (i << q) \wedge j &\leftrightarrow \mathbf{z}_0 | \mathbf{x}_0 \mathbf{z}_1 | \cdots | \mathbf{x}_{K(l)-2} \mathbf{z}_{K(l)-1} | \mathbf{x}_{K(l)-1} \mathbf{z}_{K(l)} \\ z &\leftrightarrow \mathbf{z}_0 \mathbf{z}_1 \cdots \mathbf{z}_{K(l)} \\ x &\leftrightarrow \mathbf{x}_0 \mathbf{x}_1 \cdots \mathbf{x}_{K(l)}. \end{aligned} \tag{4.2.4}$$

In the notation of lemma 4.2.1, x and z are the auxiliary integers $x((i << q) \wedge j) \wedge x_{K(l)}$ and $z((i << q) \wedge j)$ constructed from the length $\binom{l-1}{k}$ substrings $\mathbf{x}_k$ and $\mathbf{z}_k$. However, in contrast to the situation of lemma 4.2.1, $x_{K(l)}$ is not identically zero, but rather a parameter assuming values in the set $\{0, 1, \ldots, \min(2^n, m_{K(l)}^{l-1}) - 1\}$. The heretofore unspecified union in equation 4.2.3 is over both the $\min(n, \binom{l-1}{K(l)})$ bits of $\mathbf{x}_{K(l)}$ (labeling the underlying state productions) and the rightmost q bits of 4.2.4 (representing the coarsening). By successively unioning over first the z-bits and then the x-bits—using the separability of the function f (equation 4.2.1), we can independently coarsen first the left-hand and then the right-hand symbols in 4.2.3.

Following this approach, the remainder of the proof is straightforward. By the definition of k^*, all auxiliary x's and z's indexed by integers greater than or equal to k^* are to be unioned over. This coarsens both the left and right produced symbols in equation 4.2.3 to level q'. The remaining bits in the union are the rightmost q_x and q_z bits of $\mathbf{x}_{k^*-1}$ and $\mathbf{z}_{k^*-1}$ respectively. If $q_z > 0$, the left and right produced symbols can be further coarsened to level $\overline{q} = q' + q_z$, leaving exactly $q_x - q_z$ bits of the substring $\mathbf{x}_{k^*-1}$ to be unioned over. The number of super-state productions is then $I^{(l,q)}$. □

The uniform coarseness productions (4.2.2) defined in proposition 4.2.1 can be used to determine the super-state productions corresponding to a given set of state space partitions. If (q, i), (q_L, L), and (q_R, R) are super-states that respectively belong to the current state space partitions of a level l node and its left and right daughter nodes, then $(q, i) \rightarrow [(q_L, L), (q_R, R)]$ is an allowed super-state production (in the sense of Section 4.1) if and only if L shares a binary prefix with $f(\overline{x} \wedge j, \overline{z})$ and R shares a binary prefix with $\overline{x} \wedge j$ for some $0 \leq j \leq I^{(l,q)} - 1$; for if this condition is met, there is an allowed state production contained within the postulated super-state production. Since proposition 4.2.1 thus allows us to compute super-state productions by inspection, an efficient CTFDP implementation of maximum likelihood decoding might indeed exist for thinned Reed–Muller codes.

4.3 A CTFDP Decoding Algorithm for Thinned Reed–Muller Codes

Suppose a codeword from the thinned Reed–Muller code $RM^{(n)}(p - \alpha, p)$ is transmitted across a memoryless communications channel. The maximum likelihood decoding of the

received word $\mathbf{d}$ is the codeword

$$\hat{\mathbf{c}} = \arg \min_{\mathbf{c} \in RM^{(n)}(p-\alpha,p)} \sum_{i=1}^{2^p} -\ln p(d_i|c_i).$$

In this section, we formulate an explicit coarse-to-fine dynamic programming algorithm for performing this minimization that employs the thinned Reed–Muller super-states defined in Section 4.2.

As discussed in Section 4.1, a CTFDP decoding algorithm iterates the standard DP procedure (of Section 2.2) over a sequence of progressively finer super-state grammars. Since these grammars are induced by state space partitions, the *effective* level l state space for coarse-to-fine decoding is the entire tree of super-states $\{(q,i)|0 \leq i \leq N^{(l,q)} - 1, 0 \leq q \leq \log_2 N^{(l)}\}$ rather than the original linear array of ordinary states $\{0, 1, \ldots, N^{(l)} - 1\}$. Furthermore, at any node of the tree-template during a given DP iteration, we distinguish three disjoint sets of super-states. *Leaf* super-states are defined to be those super-states currently partitioning the node's state space; the cost of a leaf super-state is defined to be the minimum cost of all corresponding allowed super-state productions. *Interior* super-states strictly contain leaf super-states; their cost is the minimum cost of their constituent leaf super-states. *Exterior* super-states are strictly contained in leaf super-states and are not assigned costs. Despite, this apparent doubling in size of the problem's effective state spaces, only a small fraction of all super-states will be explored during a successful coarse-to-fine decoding trial.

The maximum likelihood decoding of the received word $\mathbf{d}$ is accomplished by the following CTFDP algorithm.

1. Compute the parameters $\{|I^{(l)}|, N^{(l)}|0 \leq l \leq p\}$ (proposition 3.6.1) and the auxiliary x's and z's (lemma 4.2.1) that together determine the thinned Reed–Muller grammar corresponding to $RM^{(n)}(p-\alpha, p)$.

2. Initialize the level 0 costs. The cost of the state j (super-state $(0,j)$) at the i-th level 0 node is $-\ln p(d_i|c_i = j)$ for $1 \leq i \leq 2^p$ and $0 \leq j \leq 1$. The super-state $(1,0)$ (i.e. the state set $\{0,1\}$) at each level 0 node is assigned the minimum nodal state cost and a pointer to the respective minimizing state.

3. Initialize the state space partitions. For each node at each level $0 \leq l \leq p$ of the

tree-template, choose the super-state $(\log_2 N^{(l)}, 0)$ as the sole leaf super-state. All other super-states (i.e. those of coarseness less than $\log_2 N^{(l)}$) are exterior.

4. Perform DP on the current set of leaf super-states. Proceeding sequentially from level 1 to level p, assign optimal costs and associated pointers first to the leaf super-states and then to the interior super-states at each of the 2^{p-l} nodes in level l:

   - Leaf super-states. If (q, i) is a leaf super-state at a level l node, compute the parameters $\overline{q}, \overline{x}, \overline{z}$ and $I^{(l,q)}$ that determine the uniform coarseness productions of proposition 4.2.1:

   $$(q, i) \rightarrow \{[(\overline{q}, f(\overline{x} \wedge j, \overline{z})), (\overline{q}, \overline{x} \wedge j)] | 0 \leq j \leq I^{(l,q)} - 1\}.$$

   Recall that these productions are not necessarily allowed super-state productions in the sense of Section 4.1, for the coarseness $\overline{q}$ super-states may not be leaf super-states at level $l-1$. However, the allowed super-state productions from (q, i) are readily obtained. For each uniform coarseness production, identify its left and right elements—respectively $(\overline{q}, f(\overline{x} \wedge j, \overline{z}))$ and $(\overline{q}, \overline{x} \wedge j)$—as leaf, interior, or exterior super-states. If an element is a leaf super-state, it must belong to an allowed super-state production. If it is exterior, there is a leaf super-state strictly containing it that belongs to an allowed super-state production. And if it is interior, it contains several constituent leaf super-states each of which belong to an allowed super-state production. Simultaneously applying these rules to both the left and right elements of a uniform coarseness production generates all the corresponding allowed super-state productions. By implicitly employing these rules and utilizing the interior super-state cost structure, the following procedure computes the minimum cost super-state production from (q, i) without explicitly determining the set of all such productions.

   Construct the subset $\{j_0, j_1, \ldots, j_{N-1}\}$ of $\{0, 1, \ldots, I^{(l,q)} - 1\}$ by the following procedure: beginning with $j_0 \stackrel{\Delta}{=} 0$, define $j_{k+1} = j_k + 2^{t(j_k)}$ where $t(j_k) = \max(t_L(j_k), t_R(j_k))$. $t_L(j_k)$ is the smallest nonnegative integer t for which there exists a non-exterior super-state of coarseness $\overline{q} + t$ (at the left daughter node) that contains the super-state $(\overline{q}, f(\overline{x} \wedge j_k, \overline{z}))$. Similarly, $t_R(j_k)$ is the smallest

nonnegative integer t for which there exists a non-exterior super-state of coarseness $\overline{q}+t$ (at the right daughter node) that contains the super-state $(\overline{q}, \overline{x} \wedge j_k)$. Naturally, the index sequence terminates at $N-1 = \max\{k \geq 0 | j_k < I^{(l,q)}\}$. For each $0 \leq k \leq N-1$, define the cost $c(j_k)$ to be the sum of the costs of the super-states $(\overline{q}+t(j_k), f(\overline{x} \wedge j, \overline{z}) >> t(j_k))$ and $(\overline{q}+t(j_k), \overline{x} \wedge j >> t(j_k))$ at the respective left and right level $l-1$ daughter nodes. $c(j_k)$ is the minimum cost of all allowed super-state productions associated with those uniform coarseness productions indexed by $\{j_k, j_k+1, \ldots, j_{k+1}-1\}$. Therefore, the minimum cost of all allowed super-state productions from (q,i) is $c(j^*) = \min_k c(j_k)$. Clearly, its determination requires $2N-1$ real number operations (N additions and $N-1$ comparisons). The corresponding optimal super-state production is the minimum cost pair of leaf super-states contained in the optimal pair:

$$[(\overline{q}+t(j^*), f(\overline{x} \wedge j^*, \overline{z}) >> t(j^*)), (\overline{q}+t(j^*), \overline{x} \wedge j^* >> t(j^*))].$$

Assign to the leaf super-state (q,i) the minimum cost $c(j^*)$ and a pointer to the optimal right super-state $(\overline{q}+t(j^*), \overline{x} \wedge j^* >> t(j^*))$.

- Interior super-states. Having calculated the optimal costs of a node's leaf super-states, we now recursively compute costs and pointers for its interior super-states. The interior super-state (q,i) is assigned the cost of and a pointer to its minimum cost component super-state—$(q-1, 2i)$ or $(q-1, 2i+1)$. Computing such a minimum requires a single real number operation (one comparison).

5. Determine the optimal derivation tree by applying the set of optimal superstate productions to the start super-state (0,0) (the start state 0) at the sole level p node. Suppose the leaf super-state (q,i) appears at a given level l node of the optimal derivation tree. We use the super-state pointers defined in step (4) above to construct the optimal super-state production $[(q_L, L), (q_R, R)]$ that assigns super-states to the given node's daughter nodes in the optimal derivation tree. If (q,i)'s associated pointer points to the optimal right super-state (q'_R, R'), then the corresponding optimal left super-state (q'_L, L') is by definition $(q'_R, f(R', z(i << q) >> q'_R)$. However, it is important to recognize that the pair $[(q'_L, L'), (q'_R, R')]$ is *not* necessarily a super-state production in the sense of Section 4.1, for these super-states may be

interior. In fact, the optimal left-produced super-state (q_L, L) is the minimum cost leaf super-state contained in (q'_L, L'); if (q'_L, L') is an interior super-state, (q_L, L) is obtained by following the sequence of pointers assigned to (q'_L, L') and its constituent interior super-states. (q_R, R) is defined analogously.

6. If the optimal derivation tree contains super-states of coarseness greater than zero, refine them and return to step (4). Otherwise, stop iterating and generate the maximum likelihood codeword $\hat{\mathbf{c}}$ (from the level 0 states of the optimal derivation tree). In the terminology of this section, we refine a super-state (q, i) by transferring it to the list of interior super-states, while replacing it (in the list of leaf super-states) with its components $(q-1, 2i)$ and $(q-1, 2i+1)$.

We discuss the relative performance of CTFDP and DP maximum likelihood decoding algorithms in Chapter 5.

Chapter 5

Discussion

5.1 Synopsis

As proved in Chapter 1, the maximum likelihood decoding scheme is optimal in the sense that it maximizes the probability of correcting channel errors. Of course, for all but the smallest codes, a sequential search for the maximum likelihood codeword is effectively impossible. Therefore, for those communications channels with factorizable likelihoods, dynamic programming is typically the only feasible strategy for exact maximum likelihood decoding.

The Viterbi algorithm is the traditional prototype dynamic programming algorithm for maximum likelihood decoding. For any trellis (e.g. convolutional or linear block) code, maximum likelihood decoding can be formulated as a shortest path search on the code's trellis diagram. As a graphical search, the Viterbi algorithm exploits the elementary dynamic programming principle that the shortest length path to a trellis node must be an extension of the shortest length path to some predecessor node. From the perspective of formal language theory, however, a trellis diagram is simply the graphical representation of a regular language and the Viterbi algorithm recursively parses a trellis code's regular grammar.

By further exploring the relationship between codes and formal languages, we aim to expand the range and applicability of dynamic programming decoding algorithms. The fundamental insight that informs our aim is expressed as follows. Given a factorizable likelihood function, only the existence of considerable grammatical structure within a code can facilitate decoding by dynamic programming. Just as hard decoding algorithms typically rely on underlying algebraic features, exact maximum likelihood decoding algorithms for general communications channels must necessarily exploit grammatical structures.

Following this strategy, we have constructed a large family of codes generated by context-free grammars and have presented three generalized Viterbi algorithms for their maximum likelihood decoding. The theoretical development proceeded in three successive stages. First, in Chapter 2 we designed dynamic programming algorithms for the maximum likelihood decoding of codes derived from context-free grammars and transmitted across either memoryless or Markov communications channels. In addition, we introduced similar algorithms to compute a useful reliability statistic—the posterior probability that a decoded word in fact matches the transmitted codeword. Second, by interpreting Forney's iterated squaring construction grammatically, we constructed in Chapter 3 a large class of CFG

representable codes, notably including the Reed–Muller codes. Moreover, by thinning these grammars, we created a family of codes having readily controllable decoding complexities. Finally, by exploiting the grammatical structure of thinned Reed–Muller codes, we derived a coarse-to-fine dynamic programming algorithm for maximum likelihood decoding in Chapter 4.

Having reviewed the theoretical content of this thesis, we now turn to the question of algorithmic performance.

5.2 DP and CTFDP Decoding Performance

We test our dynamic programming algorithms for maximum likelihood decoding on six representative thinned Reed–Muller codes in a series of simulated decoding trials. For each of the six codes—RM(2,5), RM(2,6), RM(2,7), RM(3,7), $RM^{(10)}(4,8)$, and $RM^{(8)}(6,10)$, the same fifty randomly selected codewords are transmitted across four (memoryless) bipolar communications channels with different degrees of additive gaussian noise. In other words, the bits 0 and 1 of each codeword are first converted to the respective signals -1 and 1 which are in turn corrupted by adding noise independently drawn from the distributions $\{N(0,\sigma^2)|\sigma = 0.3, 0.5, 0.8, 1.0\}$. A cross section of the decoding results is displayed in Appendix A; for each code, we select a sufficiently large value of σ (typically 0.8 or 1.0) so that there is some variability in decoding performance but avoid overwhelming noise levels.

Consider first the performance of the ordinary DP algorithm for maximum likelihood decoding (Section 2.2). Since DP decoding is infeasible for RM(3,7), the appendix details the outcome of 250 separate decoding trials for five different codes. Although the performance varies from code to code, overall the maximum likelihood decoding scheme recovers the transmitted codeword in 96% (240/250) of the trials. In contrast, consider the performance of a common approximation to this scheme, the *thresholded hard decoding scheme* (in which $\hat{\mathbf{c}}(\mathbf{d})$ is the minimum distance decoding of the binary string whose bits are *independently* most likely): the transmitted codeword is recovered in only 54% (136/250) of these trials. Thus, the theoretically optimal maximum likelihood decoding scheme performs extraordinarily well by absolute or relative empirical measures at moderate noise levels (i.e. $\sigma = 0.8$ or 1.0).

Furthermore, the posterior probability that the maximum likelihood codeword matches

the transmitted codeword provides an excellent measure of decoding reliability. In those trials for which the posterior probability exceeded 90%, the transmitted codeword was recovered 99% (220/222) of the time. When the posterior probability fell between 70% and 90%, the decoding success rate fell to 81% (13/16). And in those trials yielding posterior probabilities less than 70%, the success rate fell further to 75% (15/20).

At noise levels below those considered in the cross section, our maximum likelihood decoding algorithm's success rate is uniformly perfect with posterior probabilities exceeding 99%. In contrast, as σ increases past this threshold range of $0.8 \leq \sigma \leq 1.0$, the noise tends to overwhelm the signal producing increasingly mediocre decoding results.

Now consider the alternative CTFDP algorithm for maximum likelihood decoding (Section 4.3). Table 5.1 compares the performance of the CTFDP algorithm with that of the standard DP algorithm; for each set of simulated decoding trials, it displays the average ratio of CTFDP decoding operations to DP decoding operations. The results are striking. For all but the smallest code RM(2,5), the CTFDP algorithm computes the maximum likelihood codeword on average 5 to 100,000 times faster, depending on the particular code and noise level. Thus, even the tremendously complex code RM(3,7)—practically undecodable by the standard DP algorithm—becomes tractable at low to moderate noise levels (i.e. $\sigma \leq 0.8$).

		AVG(CTFDP/DP) Operations			
Code	DP Operations	$\sigma = 0.3$	$\sigma = 0.5$	$\sigma = 0.8$	$\sigma = 1.0$
RM(2,5)	3,007	0.8134	0.8136	1.0266	1.693
RM(2,6)	79,231	0.1199	0.1199	0.123	0.2127
RM(2,7)	4,606,719	0.006903	0.006904	0.007029	0.0099
RM(3,7)	4,425,388,799	1.136e-5	1.137e-5	1.214e-4	—
$RM^{(10)}(4,8)$	12,887,551	0.003262	0.003263	0.0052	0.0811
$RM^{(8)}(6,10)$	4,236,287	0.03261	0.03262	0.0663	0.3306

Table 5.1: The performance of CTFDP relative to DP.

The variation in CTFDP decoding performance is evident in the cross section of decoding trials displayed in the appendix. The immediate determinant of the number of CTFDP decoding operations is the number of dynamic programming iterations performed in any given trial. This statistic is particularly interesting because it also equals the number of level $p-1$ super-states participating in the terminal DP iteration. In principle, it can range from $\log_2 N^{(p-1)}+1 = \min(n, \binom{p-1}{\alpha-1})+1$ to $N^{(p-1)} = \min(2^n, m_{\alpha-1}^{p-1})$ for the code $RM^{(n)}(p-\alpha, p)$.

At low noise levels, this statistic is close to its lower bound, indicating that each successive optimal derivation tree tends to be a refinement of its predecessor. However, at higher noise levels successive optimal derivation trees are unlikely to be related, creating a surfeit of shattered super-states. For the moderate noise levels considered in our cross section of individual trials, the number of DP iterations remains well below its maximum and the CTFDP algorithm almost uniformly outperforms the standard DP algorithm.

5.3 Conclusion

In this thesis, we have demonstrated that efficient dynamic programming algorithms for maximum likelihood decoding can be designed for codes exhibiting considerable grammatical structure. Of course, much of the substance of this claim is not new. Viterbi [11] introduced his eponymous algorithm for decoding convolutional codes in 1967; Wolf [12] generalized it to the class of linear block codes in 1978. And Forney's [4] original strategy for decoding iterated squaring constructions is ultimately equivalent to our own algorithm (of Section 2.2) for decoding bent Reed–Muller codes. However, the grammatical approach to codes and decoding algorithms provides a powerful tool for both unifying and extending the range of dynamic programming techniques used in coding theory.

Code design is one important area that our grammatical approach might transform. Much of the current work ([1], [4], [7]) in this field involves the construction of minimal trellis representations for codes with specific properties (e.g. length, size, minimum distance, state space cardinality, etc.). Since a trellis diagram is simply the graphical representation of a formal language, one should focus on its underlying grammatical structure. Often grammatical production rules are far more easily specified and manipulated than their graphical analogs. For example, most trellis diagrams for iterated squaring constructions are too complicated to visualize in detail [4]. Moreover, the grammatical method transcends traditional algebraic techniques, facilitating the construction of new classes of nonlinear codes.

In addition, the grammatical approach is essential for designing efficient dynamic programming algorithms for exact maximum likelihood decoding. For communications channels with factorizable likelihoods, one can directly construct generalized Viterbi algorithms that exploit a code's grammatical structure. For example, in Chapter 2 we presented grammati-

cal algorithms to compute maximum likelihood codewords and their posterior probabilities for both memoryless and Markov channels. Moreover, within our grammatical framework, the computational complexity of these algorithms was effortlessly established. Finally, the most striking argument for the adoption of a grammatical approach to coding theory must be the performance of our coarse-to-fine decoding algorithm for thinned Reed–Muller codes; for a wide range of noise levels and all but the smallest of codes, this essentially grammatical algorithm outperforms the standard Viterbi (DP) algorithm by a wide margin.

Appendix A

Decoding Simulations

In this appendix, we present the results of simulated decoding trials for six representative thinned Reed–Muller codes: RM(2,5), RM(2,6), RM(2,7), RM(3,7), $RM^{(10)}(4, 8)$, and $RM^{(8)}(6, 10)$ (ordered by size). For each of these codes, fifty randomly selected codewords are transmitted across a simulated (memoryless) bipolar communications channel with additive gaussian noise. In other words, the bits 0 and 1 of each codeword are mapped first to the respective signals -1 and 1 which are in turn corrupted by adding noise independently drawn from the distribution $N(0, \sigma^2)$. The resulting channel outputs are thereupon decoded by three different procedures: DP maximum likelihood decoding (Section 2.2); CTFDP maximum likelihood decoding (Section 4.3); and thresholded hard decoding (in which $\hat{\mathbf{c}}(\mathbf{d})$ is the minimum distance decoding of the binary string whose bits are *independently* most likely). This third procedure is commonly used to approximate maximum likelihood decoding.

For each code, we display a table of the simulated decoding results headed by the code name, length, dimension, and minimum distance. We also display the number of required DP decoding operations (equation 2.2.1) and the fixed noise level σ. Column (A) simply labels the trials. Column (B) presents the distance $d(\mathbf{c}, \hat{\mathbf{c}})$ between the sent codeword $\mathbf{c}$ and the maximum likelihood codeword $\hat{\mathbf{c}}$ (computed by DP or CTFDP); if this distance is zero, the decoding scheme has corrected any transmission errors. Column (C) displays the posterior probability $p(\hat{\mathbf{c}}|\mathbf{d})$, providing a measure of decoding reliability. Column (D) shows the distance $d(\mathbf{c}, \hat{\mathbf{c}}')$ between the sent codeword and the result $\hat{\mathbf{c}}'$ of thresholded hard decoding. Column (E) presents the ratio CTFDP decoding operations to DP decoding operations. Column (F) displays the number of DP iterations required by a given CTFDP decoding trial. The average ratio of CTFDP decoding operations to DP decoding operations is displayed in each table's final line. Of course, since DP decoding is infeasible for RM(3,7), columns (C) and (D) remain blank.

In each case, the noise parameter σ is deliberately chosen to produce a sample of trials in which maximum likelihood decoding is beginning to diminish in reliability and the number of CTFDP operations is beginning to fluctuate. At noise levels below this threshold, maximum likelihood decoding performs perfectly and the CTFDP decoding algorithm is uniformly rapid. In contrast, above this threshold the performance of the maximum likelihood decoding scheme and the speed of its CTFDP implementation diminish rapidly as noise overwhelms the signal.

RM(2,5) [32,16,8] $\sigma = 0.80$
DP decoding operations = 3,007

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
1	0	0.9998	8	0.8188	7
2	0	0.9999	0	0.8134	7
3	0	0.9567	8	1.0589	8
4	0	0.8953	8	1.2551	9
5	0	0.8325	8	1.5284	10
6	0	0.9999	0	0.8134	7
7	0	0.9998	0	0.8134	7
8	0	0.5862	8	1.7918	11
9	0	0.9125	0	0.8380	7
10	0	0.4875	8	2.4360	13
11	0	0.4144	8	3.3818	16
12	0	0.9998	0	0.8161	7
13	0	1.0000	0	0.8161	7
14	0	0.9991	0	0.8134	7
15	0	0.6174	8	2.3788	13
16	0	0.9918	0	0.8354	7
17	0	1.0000	0	0.8134	7
18	0	0.7403	0	1.3119	9
19	0	0.9981	8	0.8134	7
20	0	0.9949	8	0.8161	7
21	0	0.9999	0	0.8134	7
22	8	0.7528	12	1.8224	11
23	0	1.0000	0	0.8161	7
24	0	1.0000	0	0.8161	7
25	0	0.9774	8	0.8168	7

RM(2,5) [32,16,8] $\sigma = 0.80$
DP decoding operations = 3,007

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
26	0	0.9534	8	0.8161	7
27	0	1.0000	0	0.8134	7
28	0	1.0000	0	0.8134	7
29	0	1.0000	0	0.8134	7
30	0	0.9998	0	0.8161	7
31	0	0.9998	0	0.8134	7
32	0	0.9984	8	0.8134	7
33	0	0.9944	0	0.8188	7
34	0	0.9894	8	0.8161	7
35	0	0.9988	0	0.8161	7
36	0	0.9926	0	0.8188	7
37	0	1.0000	0	0.8134	7
38	0	0.9998	0	0.8134	7
39	8	0.9611	0	1.0229	8
40	0	0.9354	8	1.0216	8
41	0	0.9999	0	0.8161	7
42	0	1.0000	0	0.8134	7
43	0	1.0000	0	0.8134	7
44	0	1.0000	0	0.8134	7
45	0	1.0000	0	0.8161	7
46	0	1.0000	0	0.8134	7
47	0	0.9998	0	0.8161	7
48	0	0.9999	0	0.8161	7
49	0	0.8855	0	1.3089	9
50	0	0.9999	0	0.8134	7
AVG				1.0266	

RM(2,6) [64,22,16] $\sigma = 1.00$
DP decoding operations = 79,231

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
1	0	1.0000	16	0.1426	12
2	0	0.9880	16	0.2834	17
3	16	0.3790	16	0.9728	38
4	0	1.0000	0	0.1439	12
5	0	0.9990	0	0.1425	12
6	0	0.9974	0	0.1941	14
7	0	1.0000	0	0.1201	11
8	0	0.9844	24	0.2870	17
9	0	0.9548	16	0.2238	15
10	0	0.9981	0	0.1649	13
11	0	0.9997	16	0.2015	14
12	0	0.9971	16	0.2241	15
13	0	0.9948	0	0.1207	11
14	0	1.0000	0	0.1201	11
15	0	1.0000	0	0.1203	11
16	0	0.9744	16	0.3068	18
17	0	0.9995	0	0.1405	12
18	0	1.0000	0	0.1203	11
19	0	1.0000	0	0.1205	11
20	0	0.9999	0	0.1379	12
21	0	0.9954	16	0.1201	11
22	0	0.9962	0	0.1706	13
23	0	0.9990	0	0.1224	11
24	0	0.9927	16	0.3211	19
25	0	1.0000	0	0.1221	11

RM(2,6) [64,22,16] $\sigma = 1.00$
DP decoding operations = 79,231

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
26	0	1.0000	0	0.1220	11
27	0	1.0000	0	0.1203	11
28	0	1.0000	0	0.1199	11
29	0	0.9910	0	0.1405	12
30	0	1.0000	0	0.1203	11
31	0	0.9979	0	0.1438	12
32	0	0.9994	0	0.1970	14
33	0	1.0000	0	0.1223	11
34	0	0.5866	24	1.0048	38
35	0	0.9983	0	0.1260	11
36	0	1.0000	0	0.1205	11
37	0	1.0000	0	0.1223	11
38	0	0.9512	24	0.4098	21
39	0	0.8996	16	0.4592	23
40	0	0.9994	16	0.2888	17
41	0	1.0000	0	0.1217	11
42	0	0.8790	16	0.3968	21
43	0	0.9999	0	0.1711	13
44	0	0.9968	0	0.1953	14
45	0	0.9998	0	0.1434	12
46	0	0.9997	16	0.1446	12
47	0	0.9999	0	0.1260	11
48	0	0.9987	16	0.1423	12
49	0	1.0000	0	0.1222	11
50	0	0.9688	0	0.3105	18
AVG				0.2127	

RM(2,7) [128,29,32] $\sigma = 1.00$
DP decoding operations = 4,606,719

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
1	0	1.0000	0	0.0092	18
2	0	1.0000	32	0.0375	39
3	0	1.0000	0	0.0080	17
4	0	1.0000	0	0.0069	16
5	0	1.0000	0	0.0070	16
6	0	1.0000	0	0.0095	18
7	0	1.0000	0	0.0070	16
8	0	1.0000	0	0.0069	16
9	0	1.0000	0	0.0095	18
10	0	1.0000	0	0.0178	25
11	0	1.0000	0	0.0069	16
12	0	1.0000	0	0.0070	16
13	0	1.0000	0	0.0080	17
14	0	1.0000	0	0.0093	18
15	0	1.0000	0	0.0080	17
16	0	1.0000	0	0.0133	21
17	0	1.0000	0	0.0069	16
18	0	1.0000	32	0.0093	18
19	0	1.0000	0	0.0110	19
20	0	1.0000	0	0.0154	23
21	0	1.0000	0	0.0111	19
22	0	1.0000	0	0.0079	17
23	0	1.0000	0	0.0080	17
24	0	1.0000	0	0.0069	16
25	0	1.0000	0	0.0070	16

RM(2,7) [128,29,32] $\sigma = 1.00$
DP decoding operations = 4,606,719

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
26	0	1.0000	0	0.0080	17
27	0	1.0000	0	0.0081	17
28	0	1.0000	0	0.0081	17
29	0	1.0000	0	0.0104	19
30	0	1.0000	0	0.0125	20
31	0	1.0000	0	0.0081	17
32	0	1.0000	0	0.0070	16
33	0	1.0000	0	0.0070	16
34	0	1.0000	32	0.0227	29
35	0	1.0000	0	0.0070	16
36	0	1.0000	0	0.0080	17
37	0	1.0000	0	0.0071	16
38	0	1.0000	0	0.0144	22
39	0	1.0000	0	0.0106	19
40	0	1.0000	0	0.0091	18
41	0	1.0000	0	0.0069	16
42	0	1.0000	0	0.0095	18
43	0	1.0000	0	0.0071	16
44	0	1.0000	0	0.0123	20
45	0	1.0000	0	0.0072	16
46	0	1.0000	0	0.0157	23
47	0	1.0000	0	0.0071	16
48	0	1.0000	0	0.0095	18
49	0	1.0000	0	0.0071	16
50	0	1.0000	0	0.0099	19
AVG				0.0099	

RM(3,7) [128,64,16] $\sigma = 0.80$
DP decoding operations = 4,425,388,799

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
1	0			0.0001048	72
2	0			0.0009635	255
3	0			0.0003482	144
4	0			1.579e-05	24
5	0			2.306e-05	30
6	0			9.181e-05	66
7	0			2.292e-05	29
8	0			1.377e-05	23
9	0			1.198e-05	21
10	0			9.823e-05	67
11	0			1.18e-05	21
12	0			1.458e-05	23
13	0			1.539e-05	24
14	0			1.739e-05	25
15	0			1.26e-05	22
16	0			1.538e-05	24
17	0			0.0001487	89
18	0			3.172e-05	35
19	0			1.251e-05	22
20	0			0.0001047	70
21	0			4.471e-05	42
22	0			1.876e-05	26
23	0			3.494e-05	37
24	0			1.169e-05	21
25	0			3.169e-05	34

RM(3,7) [128,64,16] $\sigma = 0.80$
DP decoding operations = 4,425,388,799

(A) Trial	(B) $d(\mathbf{c}, \hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c}, \hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
26	0			0.0004268	161
27	0			8.96e-05	64
28	0			5.287e-05	48
29	0			4.906e-05	45
30	0			3.303e-05	36
31	0			2.125e-05	28
32	0			0.0001984	103
33	0			2.507e-05	30
34	0			2.147e-05	28
35	0			1.863e-05	26
36	0			5.773e-05	50
37	0			1.154e-05	21
38	0			0.002489	411
39	0			3.066e-05	34
40	0			3.537e-05	37
41	0			1.239e-05	22
42	0			7.26e-05	57
43	0			1.819e-05	26
44	0			1.575e-05	24
45	0			1.531e-05	24
46	0			1.687e-05	25
47	0			1.974e-05	27
48	0			1.966e-05	27
49	0			2.668e-05	32
50	0			6.965e-05	56
AVG				0.0001214	

$RM^{(10)}(4,8)$ [256,118,16] $\sigma = 0.80$
DP decoding operations = 12,887,551

(A) Trial	(B) $d(\mathbf{c},\hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c},\hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
1	0	0.9489	60	0.0085	20
2	0	1.0000	0	0.0043	13
3	0	1.0000	24	0.0037	12
4	0	0.9976	0	0.0043	13
5	0	0.9999	32	0.0048	14
6	0	1.0000	0	0.0033	11
7	0	1.0000	0	0.0033	11
8	0	1.0000	16	0.0033	11
9	0	1.0000	0	0.0038	12
10	0	0.9838	0	0.0097	23
11	0	0.9957	0	0.0052	15
12	0	0.9996	0	0.0048	14
13	0	0.9827	28	0.0089	21
14	0	0.7814	60	0.0124	25
15	0	0.9993	24	0.0049	14
16	0	1.0000	16	0.0043	13
17	0	1.0000	0	0.0033	11
18	0	0.9989	0	0.0033	11
19	0	0.9997	36	0.0060	16
20	0	0.9993	0	0.0082	20
21	0	1.0000	28	0.0033	11
22	0	1.0000	0	0.0038	12
23	0	1.0000	0	0.0038	12
24	0	1.0000	0	0.0033	11
25	0	0.9664	16	0.0128	27

$RM^{(10)}(4,8)$ [256,118,16] $\sigma = 0.80$
DP decoding operations = 12,887,551

(A) Trial	(B) $d(\mathbf{c},\hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c},\hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
26	0	1.0000	24	0.0033	11
27	0	1.0000	16	0.0037	12
28	0	1.0000	0	0.0033	11
29	0	0.9989	16	0.0052	15
30	0	0.9995	36	0.0071	18
31	0	0.6291	40	0.0129	27
32	0	1.0000	24	0.0033	11
33	0	1.0000	0	0.0037	12
34	0	1.0000	32	0.0065	17
35	0	1.0000	32	0.0038	12
36	0	0.8970	44	0.0120	26
37	0	0.9988	16	0.0048	14
38	0	1.0000	36	0.0048	14
39	0	1.0000	0	0.0033	11
40	0	0.9972	44	0.0054	15
41	0	0.9999	0	0.0043	13
42	0	0.9926	16	0.0054	15
43	0	1.0000	0	0.0033	11
44	0	1.0000	0	0.0033	11
45	0	0.9948	40	0.0033	11
46	0	1.0000	36	0.0038	12
47	0	1.0000	0	0.0033	11
48	0	1.0000	40	0.0033	11
49	0	0.9929	40	0.0060	16
50	0	1.0000	0	0.0033	11
AVG				0.0052	

$RM^{(8)}(6,10)$ [1024,440,16] $\sigma = 0.80$
DP decoding operations = 4,236,287

(A) Trial	(B) $d(\mathbf{c},\hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c},\hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
1	0	0.9961	52	0.0329	10
2	0	0.9968	40	0.0327	10
3	0	0.9999	24	0.0328	10
4	0	0.9960	88	0.0330	10
5	24	0.8850	52	0.0327	10
6	0	0.9736	80	0.0425	12
7	24	0.4450	64	0.1926	39
8	0	0.9953	68	0.0426	12
9	0	0.9948	64	0.0327	10
10	0	0.7854	64	0.0525	14
11	0	0.9999	52	0.0328	10
12	16	0.3720	112	0.1104	25
13	0	0.9999	56	0.0328	10
14	0	0.9838	56	0.0423	12
15	16	0.8022	92	0.0889	21
16	0	0.9997	52	0.0328	10
17	0	0.6635	68	0.1308	29
18	16	0.9576	56	0.0677	17
19	0	0.9996	64	0.0327	10
20	0	0.9993	84	0.0329	10
21	0	0.9923	88	0.0526	14
22	0	0.3587	48	0.1445	31
23	0	0.9994	64	0.0329	10
24	0	0.9710	108	0.0330	10
25	0	0.9997	60	0.0379	11

$RM^{(8)}(6,10)$ [1024,440,16] $\sigma = 0.80$
DP decoding operations = 4,236,287

(A) Trial	(B) $d(\mathbf{c},\hat{\mathbf{c}})$	(C) $p(\hat{\mathbf{c}}\|\mathbf{d})$	(D) $d(\mathbf{c},\hat{\mathbf{c}}')$	(E) CTFDP/DP	(F) DPIs
26	0	0.2748	64	0.2516	48
27	0	0.9937	40	0.0425	12
28	0	0.4483	80	0.1155	26
29	0	0.9352	80	0.0576	15
30	16	0.4291	84	0.0887	21
31	0	0.9979	76	0.0327	10
32	0	0.8818	48	0.0329	10
33	0	0.9988	60	0.0475	13
34	0	0.9990	44	0.0377	11
35	0	0.6631	68	0.1367	30
36	0	0.8959	68	0.0775	19
37	16	0.5427	80	0.0623	16
38	0	0.9998	64	0.0328	10
39	0	0.9927	104	0.0473	13
40	0	0.9936	44	0.0520	14
41	0	0.9237	48	0.0627	16
42	0	0.4745	76	0.0779	19
43	0	0.5098	72	0.0832	20
44	0	0.8833	32	0.1030	24
45	0	0.9745	60	0.0476	13
46	0	0.5249	72	0.0729	18
47	0	0.7052	52	0.0476	13
48	0	0.9491	64	0.0676	17
49	0	0.5803	52	0.1680	35
50	0	0.9656	60	0.0376	11
AVG				0.0663	

Bibliography

[1] Esmaeli, M., Gulliver, A., and Secord, N.. "Quasi-cyclic structure of Reed–Muller codes and their smallest regular trellis diagram." *IEEE Trans. Inform. Theory* 43(1997):1040–52.

[2] Forney, G. D.. "The Viterbi algorithm." *Proc. IEEE* 61(1973): 168–78.

[3] Forney, G. D.. "Convolutional codes II: maximum likelihood decoding." *Information and Control* 25(1974): 222–66.

[4] Forney, G. D.. "Coset codes—Part II: binary lattices and related codes." *IEEE Trans. Inform. Theory* 34(1988):1152–87.

[5] Geman, S.. "Codes from production rules, and their maximum-likelihood decoding." In preparation (1997).

[6] Hopcroft, J. and Ullman, J.. *Introduction to Automata Theory, Languages, and Computation*. Reading, Mass.: Addison-Wesley,1979.

[7] Muder, D. J.. "Minimal trellises for block codes." *IEEE Trans. Inform. Theory* 34(1988):1049–53.

[8] Raphael, C.. "Coarse-to-fine dynamic programming." In preparation (1997).

[9] Raphael, C. and Geman, S.. "A grammatical approach to mine detection." In *Detection and Remediation Technologies for Mines and Minelike Targets II*. Eds. Dubey, A. C. and Barnard, R. L.. *Proceedings of SPIE* 3079(1997):316–332.

[10] Roman, S.. *Coding and Information Theory*. New York: Springer-Verlag, 1992.

[11] Viterbi, A. J.. "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm." *IEEE Trans. Inform. Theory* 13(1967):260–9.

[12] Wolf, J. K.. "Efficient maximum likelihood decoding of linear block codes using a trellis." *IEEE Trans. Inform. Theory* 24(1978):76–80.